UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO DE CIÊNCIAS DA COMPUTAÇÃO

DIEGO LUIZ MARAFON

INTEGRAÇÃO JAVASERVER FACES E AJAX

ESTUDO DA INTEGRAÇÃO ENTRE AS TECNOLOGIAS JSF E AJAX

FLORIANÓPOLIS, 2006

UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO DE CIÊNCIAS DA COMPUTAÇÃO

DIEGO LUIZ MARAFON

INTEGRAÇÃO JAVASERVER FACES E AJAX

ESTUDO DA INTEGRAÇÃO ENTRE AS TECNOLOGIAS JSF E AJAX

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Ciências da Computação.

Orientador:
Prof. Dr. Leandro José Komosinski

Membros da Banca:
Prof. Dr. João Candido Dovicchi
Profa. Dra. Patrícia Vilain

FLORIANÓPOLIS, 2006

Diego Luiz Marafon

INTEGRAÇÃO JAVASERVER FACES E AJAX

ESTUDO DA INTEGRAÇÃO ENTRE AS TECNOLOGIAS JSF E AJAX

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Ciências da Computação.

Orientador: _____
Prof. Dr. Leandro José Komosinski

Banca examinadora

_____
Prof. Dr. João Candido Dovicchi

_____
Profa. Dra. Patrícia Vilain

À minha família.

Agradecimentos

Agradeço à minha família, especialmente ao meu pai Moacir Antônio Marafon, minha mãe Isabel Silva Marafon e meu irmão Thiago Antônio Marafon, os responsáveis por muitos ensinamentos e conquistas obtidas ao longo da minha vida.

Aos meus amigos e colegas de trabalho que sempre me incentivaram e auxiliaram, ajudando-me a crescer pessoal e profissionalmente. À Ana Christina da Silva Florão pelas suas valiosas dicas na revisão deste trabalho.

Gostaria de agradecer também ao orientador deste trabalho, o professor Leandro José Komosinski, pelo apoio, incentivo e ensinamentos, fundamentais para a realização deste trabalho.

Resumo

O desenvolvimento de aplicativos Web vem crescendo cada vez mais principalmente com a consolidação da utilização da linguagem de programação Java com o padrão J2EE.

Com o uso constante desta linguagem de programação muitos investimentos foram realizados em projetos e tecnologias a fim de aprimorar e padronizar o processo de desenvolvimento de sistemas que trabalham na rede mundial de computadores.

Um desses projetos que conta com grande destaque atualmente é o *framework JavaServer Faces*(JSF), o qual atua principalmente nas camadas de Visão e Controle do modelo MVC e possui como principais características facilitar o desenvolvimento de componentes personalizados e adotar um modelo de programação orientada a eventos, tentando se assemelhar ao desenvolvimento *desktop*.

Uma outra característica do JSF é a utilização apenas de requisições síncronas, ou seja, a cada evento invocado pelo cliente é criada uma requisição que segue até o servidor para realizar algum processamento. Em seguida uma nova página é totalmente construída, mesmo que esta seja igual à anterior, e exibida no *browser* do cliente, que por sua vez espera por todas essas ações sem poder interagir com o sistema.

Um outro projeto que vem obtendo igual sucesso é o que desenvolve uma tecnologia, chamada AJAX, e que utiliza principalmente *JavaScript*. O uso do AJAX proporciona significativos benefícios visto que permite que o sistema possa alterar alguns dos dados exibidos aos usuários sem a necessidade de reconstrução de toda a página novamente, sendo modificado apenas o conteúdo dos campos alterados, tornando os sistemas mais dinâmicos, interativos e atraentes aos seus usuários.

Este trabalho dedica-se a estudar 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.

PALAVRAS-CHAVE: *Java, J2EE, JavaServer Faces, AJAX, MVC, JavaScript, aplicativos Web.*

Abstract

The web development has been increasing a lot in the last years. Many investments have been applied in projects and technologies to improve the development of applications that run on the internet, especially the ones that use the Java programming language and the platform J2EE.

One of these projects is the JavaServer Faces framework, that acts mainly in the View and Control in the MVC design pattern and has the followings objectives: facilitate components development and use event-oriented programming like in desktop systems development.

Another project that is making a lot of success is the one that develop a technology that uses Javascript named AJAX. One of the benefits that AJAX brings is that the application can change some data that are showed to the user without rebuilding the whole page. Using AJAX, the application becomes more dynamic, because the interface that is showed to the user doesn't have to "blink" every time that the system changes a data.

Since both technologies could be integrated, this work is going to show: the ways that both could work together, some projects that are working with this objective and study if in the future these can be totally integrated.

KEYWORDS: *Java, J2EE, JavaServer Faces, AJAX, MVC, JavaScript, Web applications.*

Índice de figuras

Lista de Gráficos

Lista de Tabelas

Lista de Abreviaturas e Siglas

AJAX – Asynchronous JavaScript and XML

DOM – Document Object Model

IDE – Integrated Development Environment

HTML – HyperText Markup Language

HTTP – HyperText Transfer Protocol

JSF – JavaServer Faces

JSP – Java Server Pages

J2EE – Java 2 Enterprise Edition

XML – eXtensible Markup Language

WWW – World Wide Web

W3C – World Wide Web Consortium

Sumário

1 INTRODUÇÃO

1.1 Contexto

O advento da rede mundial de computadores no século passado vem desde então revolucionando o mundo: mudando conceitos e hábitos, e tornando o computador cada vez mais útil e necessário na vida das pessoas.

Com a Internet, programas e informações puderam ser compartilhados por usuários, de forma on-line e simultânea, vencendo grandes distâncias e tornando o mundo uma única aldeia. Essa revolução de comportamento motiva, entre outras ações, o desenvolvimento de aplicativos Web, e proporciona a todos os segmentos que trilham o caminho da informatização um crescimento gigantesco. Fábulas 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, vem se tornando padrão, não somente para aplicações Web, mas no contexto geral, a adoção da arquitetura MVC (Modelo – Visão – Controle) [FOWLER, 2003].

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.

Com essa divisão do problema o modelo MVC tem como seus principais objetivos facilitar: o reuso do código, a modularização dos sistemas e a manutenção do programa, devido à melhor compreensão do código e do fluxo da aplicação.

Com a popularização da internet muitas linguagens de programação para Web já foram inventadas, tais como: **Asp**, **Php**, **Java** (utilizando o padrão J2EE, uma plataforma de programação que faz parte da plataforma Java e utilizada para desenvolvimento de aplicações executadas em um servidor [WIKIPEDIA, 2006]), entre outras.

A linguagem Java, criada em 1992, teve franca expansão, 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**.

Com o intuito de facilitar ainda mais a construção de aplicativos com esta linguagem foram e estão sendo desenvolvidos muitos projetos para a elaboração de *frameworks* e tecnologias, tais como: *Struts*[1] , *JavaServer Faces*[2], *Spring*[3], 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, há 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&DOLLIMORE&KINDBERG, 1994], 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.

---

[1] http://struts.apache.org/
[2] http://java.sun.com/javaee/javaserverfaces/
[3] Spring: O Spring é um *framework* open source criado por Rod Johnson e atua principalmente na camada de Controle do modelo MVC. A página oficial do projeto é http://www.springframework.org/

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 a 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.

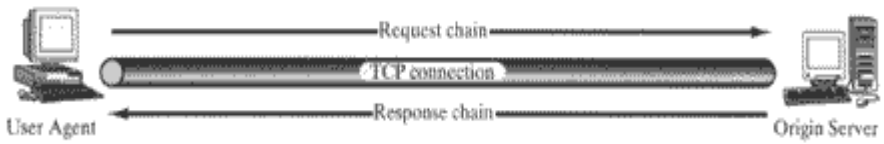

Ilustração 1–Funcionamento do protocolo HTTP. Fonte: THE WEB DESIGNER 5 HTTP PRIMER, 2006]

Finalmente, 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[4], GoogleMap[5] entre outras.

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

[4] http://www.gmail.com/
[5] http://maps.google.com/

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 5.0*[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.

Para complementar o estudo, um componente foi desenvolvido possibilitando a integração das duas tecnologias de maneira simplificada e eficiente, servindo como exemplo para desenvolvedores.

Um 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 Objetivos

1.2.1 Objetivo Geral

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

1.2.2 Objetivos Específicos

- Verificar se o framework JSF pode integrar a tecnologia AJAX nas suas próximas versões de maneira nativa.
- Desenvolver um componente utilizando a melhor forma de integração das tecnologias comentadas, de acordo com as conclusões do estudo realizado.

[6] http://java.sun.com/javaee/technologies/javaee5.jsp

1.2.3 Metodologia

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.

Será então desenvolvido um componente que utilizará a melhor técnica pesquisada.

2 BASE TEÓRICA

2.1 Introdução

Para que se possa compreender o assunto central deste trabalho é necessário definir inicialmente alguns conceitos relevantes ao tema.

2.2 JavaScript

É uma linguagem de programação desenvolvida pelo Netscape em 1995, cujo principal objetivo era tornar os sistemas Web mais interativos. Com esta linguagem os programas podem, por exemplo, fazer a validação de valores inseridos pelo usuário no browser do cliente, sem a necessidade de efetuar uma requisição ao servidor [GOODMAN, 2001].

As principais características do *JavaScript* são:

- As variáveis oferecem tipagem dinâmica, ou seja, não precisam ter seus tipos declarados.
- Esta é uma linguagem interpretada, portanto, o seu código-fonte não precisa ser compilado.
- As funções desenvolvidas utilizando a linguagem *JavaScript* são executadas no *browser* do cliente.
- *JavaScript* é orientada a eventos, ou seja, as funções são executadas quando um evento associado a função ocorre.

Porém, uma das grandes desvantagens da utilização de *JavaScript* é o prejuízo causado na manutenção dos sistemas, haja visto que com a utilização de funções dessa linguagem, parte da lógica é colocada na camada de Visão do modelo MVC, diminuindo assim a compreensão sobre o código fonte dos programas.

2.3 XML

O acrônimo XML significa Linguagem de Marcação Extensível (*eXtensible Markup Language*). Esta linguagem é usada para descrever documentos e dados de maneira padrão, definida pelo W3C, que possam ser transportados utilizando a Internet [DURANT&BENZ, 2003].

É atualmente utilizada em inúmeras aplicações, inclusive para programas *desktops*, devido à facilidade de uso e a diversidade de funções disponíveis.

Os principais benefícios da utilização desta linguagem são:

- O desenvolvedor pode estendê-la, ou seja, pode adicionar novas *tags*.
- Formal e fácil de ser usada.
- Os dados de diversas fontes podem ser integrados, ou seja, pode-se efetuar o compartilhamento de dados entre programas diferentes.

2.4 DOM

O Modelo de Objeto de Documento *(Document Object Model*) é uma plataforma definida pelo W3C a qual armazena as informações de uma página numa estrutura de dados conhecida como árvore, permitindo que programas e *scripts* acessem e alterem o conteúdo, a estrutura e o estilo dos documentos [W3C, 2006a].

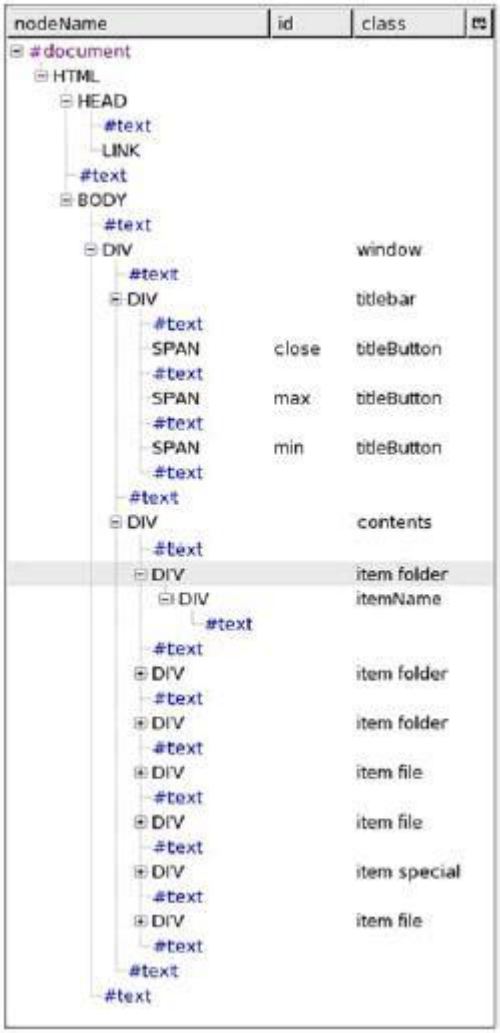

Ilustração 2 - DOM - Árvore que contém os dados de uma página HTML. Fonte: [CRANE&PASCARELLO, 2006]

O DOM é uma API de programação de documentos e foi desenvolvido para ser utilizado em qualquer linguagem de programação.

O DOM pode ser dividido atualmente em duas partes:

- DOM *Core*: define como serão feitos os acessos e as modificações dos valores dos modelos de objetos de documentos.
- DOM *Html*: estabelece quais consultas e operações podem ser realizadas sobre o documento HTML.

3 JAVASERVER FACES

3.1 Introdução

Este *framework* é o resultado de um projeto apoiado pela Sun e teve sua primeira versão apresentada em setembro de 2002 [GEARY&HORSTMANN, 2005].

O *JavaServer Faces* atua principalmente nas camadas de Visão e Controle do modelo MVC [FOWLER, 2003], conforme visualizado na figura abaixo, e tem como principal objetivo facilitar o trabalho dos desenvolvedores na construção das páginas dos sistemas Web. Para isso, disponibiliza aos usuários uma biblioteca de componentes prontos para serem utilizados e permite que novos componentes sejam criados de maneira simples. Além disso, possui um modelo de programação dirigido a eventos, tentando se aproximar das aplicações *desktops*.

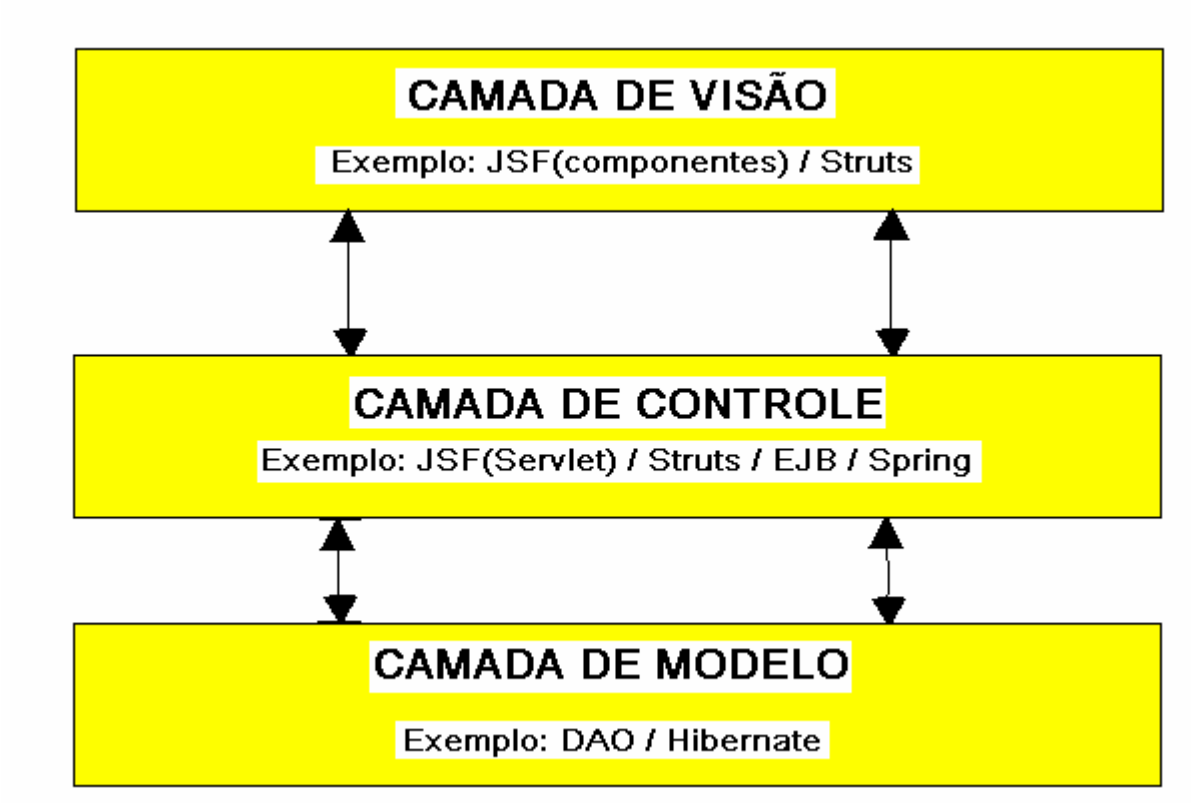

Ilustração 3 - Modelo MVC e seus frameworks

3.2 Contextualização JSF

Nesta seção serão abordados alguns tópicos que compõem o *framework* JSF e que são essenciais para a compreensão das seções posteriores.

3.2.1 Ciclo de Vida das Requisições JSF

É o caminho percorrido por uma requisição JSF, desde a sua criação, com a realização de um evento, até a renderização[7] da página a ser exibida para o usuário.

Este ciclo é divido em seis fases, sendo que os desenvolvedores podem acrescentar novas etapas ou implementar artifícios para que as requisições não executem alguma determinada fase.

3.2.2 FacesServlet

Conhecido como "Controlador Frontal" do *framework* JSF, visto que este é o ponto pelo qual devem passar todas as requisições, assim que forem instanciadas.

Depois de passar pelo *FacesServlet* a requisição começa a executar a primeira fase do ciclo de vida das requisições JSF.

3.2.3 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.

[7] Renderizar: neologismo originado do verbo 'to render" do inglês e que é muito usado em livros técnicos da

- 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 V*iew* de uma página já existe e precisa apenas ser restaurada para o usuário.

3.2.4 FacesContext

Este objeto é o responsável por armazenar as *View ID* e todas as informações necessárias para o *framework* JSF gerenciar os componentes de uma página.

3.2.5 Backing Beans

São classes as quais contém parte ou todas as informações dos valores de uma página [GEARY&HORSTANN, 2005]. É a representação do formulário HTML na forma de um objeto Java.

3.2.6 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.

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

área da computação. Significa a conversão de um dado para ser exibido.

3.3 Ciclo de Vida

3.3.1 Fases do Ciclo de Vida

Para que não haja erros básicos nos sistemas é extremamente importante que os desenvolvedores tenham pleno conhecimento do ciclo de vida das aplicações JSF.

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.

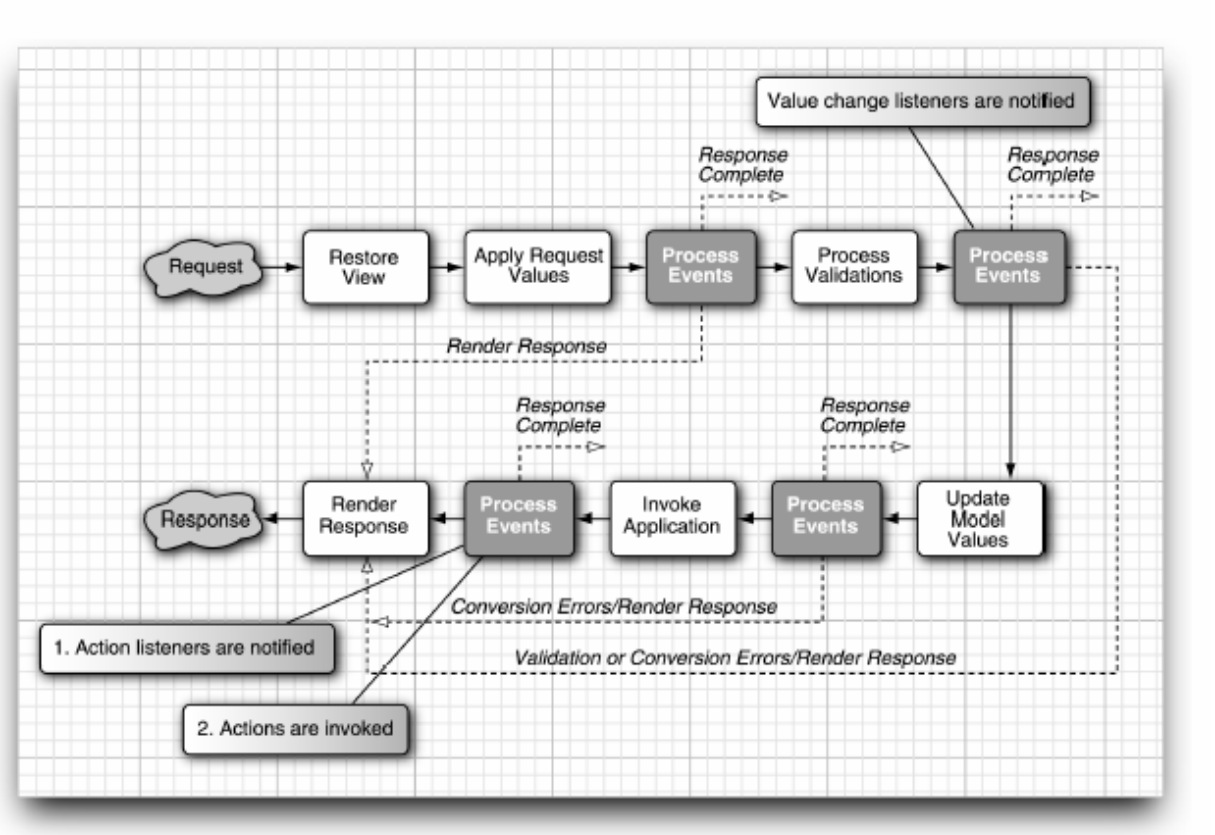

Ilustração 4 Ciclo de vida das requisições JSF – fonte: [GEARY&HORSTMANN, 2005]

Como se pode observar na ilustração anterior, o ciclo de vida de uma requisição *JavaServer Faces* pode ser dividido em seis fases:

3.3.1.1 Restaurar visão

Esta é a primeira fase do ciclo, iniciada quando o *FacesServlet* recebe uma requisição. A partir dessa ação o controle examina a requisição e a página que solicitou o serviço.

O *FacesServlet* então verifica se já possui uma V*iew ID* da página solicitante no *FacesContext*. Caso ainda não exista, uma nova V*iew ID* é criada.

3.3.1.2 Aplicar Valores de Requisição

O objetivo inicial desta fase é a recuperação do estado corrente de cada componente.

Caso a propriedade *immediate* do componente não for igual a *true*, o valor do dado do componente é apenas convertido para o tipo da propriedade em questão, como por exemplo: Integer, Boolean e String.

No entanto, se a propriedade *immediate* do componente possuir o valor igual a *true*, o valor do componente é convertido e validado já nesta fase.

3.3.1.3 Processar Validações

Nesta fase cada componente irá verificar se os valores que estão sendo recebidos podem ser aceitos sem que haja nenhum problema de dados, como por exemplo: um atributo numeral receber um valor contendo letras.

Caso seja encontrado um erro, uma mensagem de erro é adicionada no *FacesContext* e o controle da aplicação vai para a última fase do ciclo de vida, chamada “Renderizar Resposta”.

3.3.1.4 Atualizar Valores de Modelo

Aqui o objetivo único é atualizar os valores dos *backing beans*. Como esta fase é sucessora à de validação e esta somente é chamada caso ocorra sucesso na validação dos dados, pode-se efetuar a atualização dos valores do modelo com segurança.

3.3.1.5 Invocar Aplicação

São executados os *listeners* de ação (lógica de interface) e as ações (lógica de negócio), respectivamente.

Também nesta fase do JSF a aplicação deve definir, dependendo das respostas das ações acima citadas e do arquivo *faces-config.xml*, a página a ser mostrada na seqüência ao usuário.

3.3.1.6 Renderizar Resposta

Última fase do ciclo de vida do *framework JavaServer Faces*, tem como objetivo principal o de reconstruir a página que deve ser exibida ao usuário.

3.3.2 Variações do Fluxo do Ciclo de Vida

O ciclo de vida das requisições que utilizam o *framework JavaServer Faces*, representado na Ilustração 4 pode sofrer alterações no fluxo, como por exemplo: “saltar” fases caso execute alguns eventos, como os de ação, de mudança de valor, etc.

3.3.2.1 Eventos de Mudança de Valor

Os eventos de mudança de valor ocorrem, por exemplo, quando o usuário seleciona um valor de uma lista ou marca um *checkbox* ou um *radio button*.

Quando um evento de mudança de valor ocorre, o ciclo de vida de uma aplicação *JavaServer Faces* é executado normalmente até a terceira fase chamada de “Processar Validações” (*Process Validations*).

Logo em seguida, se não houver erros, a função definida para ser executada ao ocorrer um evento de mudança de valor é processada.

Depois dessa ação, as fases restantes do ciclo de vida são executadas normalmente até que uma nova página seja exibida ao usuário.

3.3.2.2 Eventos de Ação

Eventos de ação são executados quando o usuário clica em um botão ou um link de uma aplicação.

Quando um evento de ação ocorre, o ciclo de vida de uma requisição JSF é executado normalmente até a penúltima etapa, conhecida como "Invocar Aplicação" (*Invoke Application*). Nesta fase a ação definida para ser invocada é executada.

Depois do cumprimento dessas etapas, a última fase do ciclo de vida das requisições JSF é executada, finalizando o ciclo.

3.3.2.3 Tags Event Listener

Esses eventos são representados pelas classes *ActionListener* e *ValueChangeListener* e são análogos respectivamente aos eventos de ação e às mudanças de valor. A diferença entre esses eventos é que, enquanto nos primeiros deve-se representar o evento numa classe, nos últimos deve-se utilizar uma *tag*.

Mesmo com esta pequena diferença, o fluxo do ciclo de vida entre os eventos de ação e mudança de valor com seus respectivos *tag event listeners* são iguais.

3.3.2.4 Componentes Imediatos

São utilizados quando se deseja executar uma ação numa página sem que haja a necessidade de conferir o preenchimento de todos os campos obrigatórios.

Os componentes imediatos podem ser divididos em dois tipos:

- Componentes de Entrada Imediatos:

  Fazem parte deste tipo de componentes: os de entrada de texto e os que permitem ao usuário selecionar um determinado dado.

  O fluxo do ciclo de vida para estes componentes ocorre normalmente até a fase de "Aplicar Valores de Requisição" (*Apply Request Values*). Logo após esta etapa, a conversão e validação do dado que sofreu a ação é realizada.

  Depois de realizadas todas essas ações o fluxo se encaminha então para a última fase do ciclo de vida: "Renderizar Resposta" (*Render Response*).

- Componentes de Comando Imediatos:

  Os links e botões podem fazer parte dos componentes de comandos imediatos. O fluxo do ciclo de vida de uma requisição de uma aplicação JSF para este tipo de componente é idêntico ao dos Componentes de Entrada Imediatos apresentados anteriormente, ou seja, segue normalmente até a segunda fase, "Aplicar Valores de Requisição" (*Apply Request Values*). A requisição então segue diretamente para a última fase "Renderizar Resposta" (*Render Response*).

3.3.2.5 Eventos de Fase

O *framework* JSF permite que seus desenvolvedores criem novas fases do ciclo de vida das requisições. Nessas novas etapas o sistema poderia, por exemplo, verificar os erros e a permissão do usuário para execução de certas ações.

3.4 Vantagens da utilização do framework JSF

- Permitir que os programadores criem interfaces usando os componentes do *framework* ou componentes desenvolvidos por algum desenvolvedor.
- 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.

3.5 Desvantagens da utilização do framework JSF

- Conter pouca documentação de qualidade disponível visto o pouco tempo de existência em comparação a outros *frameworks*.
- Executar apenas requisições síncronas.

3.6 Aplicação Exemplo

Nesta seção será apresentado um exemplo de módulo de aplicativo cujo código-fonte está disponibilizado na seção “10.1 Exemplo JSF”, o qual utiliza o *framework* JSF. Entretanto, o referido módulo não aproveita as vantagens da tecnologia AJAX e, conseqüentemente, apresenta o problema citado anteriormente, em que a cada ação executada há a necessidade da reconstrução de uma nova página, mesmo que esta seja igual à anterior.

O exemplo criado para demonstração foi um programa de cadastro de dados de pessoas, como pode ser visto na figura abaixo.

Ilustração 5 - Exemplo de programa usando JSF

Este aplicativo possui algumas particularidades, como por exemplo:

- Número do CPF:

Para verificar se o número do CPF informado é válido usando a técnica de dígito verificador, foi criada uma classe *CPFValidator* que implementa a interface *Validator* disponibilizada pelo *framework* JSF.

Utilizando o *framework* JSF, o implementador pode efetuar a validação do número do CPF de duas maneiras:

- validação após preenchimento do CPF. Neste caso, a tela toda deveria ser atualizada mesmo que a maioria de seus dados permanecesse os mesmos, causando tal desconforto ao usuário, além da performance da aplicação ficar prejudicada;
- outra alternativa seria aguardar o final do preenchimento de todos os campos e o clique no botão “OK” para que tal validação fosse realizada. Esta técnica (utilizada neste exemplo) também necessita que a página seja totalmente reconstruída após este evento.

Caso ocorra um erro de validação do número do CPF o sistema reconstrói a página de cadastro e exibe a mensagem de erro, como pode ser visto na figura abaixo:

Ilustração 6 - Exemplo de programa usando JSF - Erro CPF

- Seleção do estado:

Todas as vezes em que o usuário selecionar um estado diferente na tela de cadastro, conforme Ilustração 7, o sistema faz uma requisição ao servidor, que executa um método, no caso *populeListaCidades* da classe *PessoaForm*.

Com a invocação desse serviço a página de cadastro é reconstruída, disponibilizando ao usuário todas as cidades do programa que estão associadas ao estado selecionado.

Ilustração 7 - Exemplo de programa usando JSF - Disponibilizando cidades

3.7 Conclusão

Com o estudo sobre *JavaServer Faces*, um *framework* que atua nas camadas de Visão e Controle dos aplicativos Web, foi possível observar a sua grande contribuição para a implementação de aplicações para uso na Internet.

No entanto, como foi demonstrado no módulo do aplicativo descrito na seção “3.6 Aplicação Exemplo”, implementado usando JSF, uma de suas deficiências é a inexistência de recursos que permitam que as páginas não precisem ser totalmente reconstruídas ao sofrerem algum evento, mesmo que apenas partes dos dados tenham sido modificadas. Tal deficiência, além de degradar a performance dos sistemas, gera desconforto para o usuário.

Dessa forma, na continuidade deste trabalho são buscados recursos para a integração da tecnologia AJAX com o JSF objetivando eliminar as deficiências apresentadas neste tópico sobre o JSF.

4 AJAX

4.1 Introdução

Neste capítulo pretende-se comentar algumas características importantes do conjunto de tecnologias tão em voga nos últimos meses, chamado de AJAX.

Esse conjunto de tecnologias não era muito conhecido até a Google, uma das maiores empresas do mundo, com grande destaque no oferecimento de recursos de busca de conteúdos na Web, começar a investir e a utilizar o AJAX em seus aplicativos como: Gmail, Google Suggest, GoogleMap, entre outros.

A partir de então, o AJAX começou a ser objeto de estudo de muitos desenvolvedores e centros de pesquisa, e começou a ser integrado a diversos aplicativos no mundo inteiro.

4.2 Histórico

O Internet Explorer 5 teve sua versão lançada no final do século passado e contou com um objeto novo - até então chamado ActiveX - que implementava Microsoft.XMLHTTP. Tal objeto permitia a realização de requisições assíncronas.

Com o sucesso do lançamento do *MicrosoftXMLHTTP*, o W3C (World Wide Web Consortium), conjunto de empresas formado pela Sun Microsystems, Microsoft Corporation, Mozilla Foundation, entre outras, que desenvolvem e definem padrões do WWW (World Wide Web), padronizou a implementação nos *browsers* de um objeto similar ao *MicrosoftXMLHTTP*, chamado de *XMLHTTPRequest*.

O documento que formalizou este padrão pode ser visto em: http://www.w3.org/TR/XMLHttpRequest/.

Contudo, a Microsoft manteve o uso de seu componente (*MicrosoftXMLHTTP*) e não aderiu aos padrões W3C. Como conseqüências, existem várias diferenças entre Internet Explorer da Microsoft e os demais navegadores, como por exemplo, a criação de um objeto *XMLHTTPRequest*.

Para criar um objeto *XMLHTTPRequest* no Internet Explorer deve-se utilizar o JavaScript das seguintes formas:

1 - *var xmlhttp = new ActiveXObject("Microsoft.XMLHttp");*

2- *var xmlhttp = new ActiveXObject("MSXML2.XMLHttp");*

Já nos demais *browsers*, como no Mozilla Firefox, Konqueror e Netscape, deve-se inicializar o objeto *XMLHTTPRequest* desta maneira:

var xmlhttp = new XMLHttpRequest();

Esta é uma das inúmeras diferenças de comportamento entre os navegadores de internet que aceitam *JavaScript*, o que torna difícil a implementação de soluções genéricas para todos os *browsers* de mercado.

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 nos aplicativos para atualizar alguns dados de uma página sem ter que reconstruí-la integralmente, tarefa esta que só era possível de ser feita através de artifícios de programação (utilizando *iFrames*).

Com isso, utilizando o *XMLHTTPRequest*, as novas informações que aplicativos deveriam mostrar ao seu usuário poderiam ser colocadas em algum ramo da árvore e a página poderia renderizar 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. [ADAPTATIVE PATH, 2006].

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*.

4.3 Características

Segundo o livro AJAX in Action [CRANE&PASCARELLO, 2006], a tecnologia AJAX segue três princípios:

- 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.

- Servidores enviam dados:

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

- 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.

4.4 Funcionamento

As interações entre o cliente e o servidor utilizando a tecnologia AJAX podem ser vistas na figura abaixo:

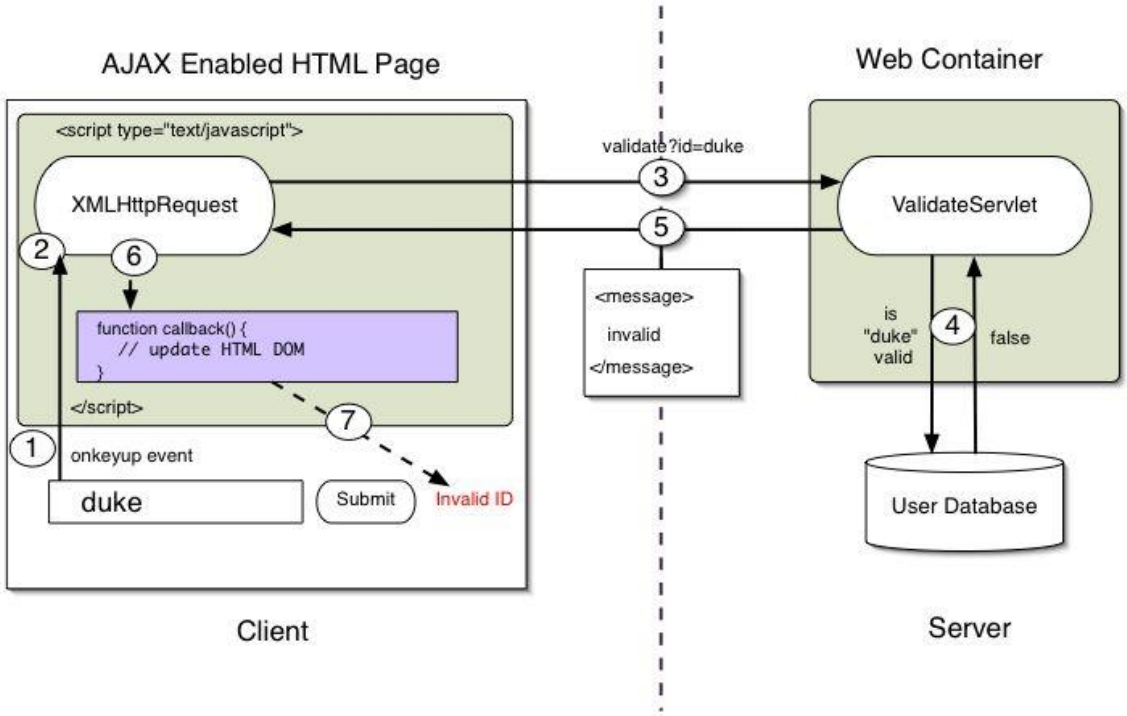

Ilustração 8 - Fases do AJAX - fonte: [SUN, 2006a]

- Passo 1: o usuário executa um evento na página, como por exemplo, digitar alguma palavra num determinado campo.

- Passo 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.

- Passo 3: o objeto *XMLHTTPRequest* criado no passo 2 executa a ação configurada também na etapa anterior.

- Passo 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.

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

- Passo 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.

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

4.5 Vantagens da utilização da tecnologia AJAX

As grandes vantagens da utilização desta tecnologia são:

- Aumento da performance:

Por meio de diversos estudos consagrados foi atestada que a utilização do AJAX de maneira correta pode aumentar a performance dos aplicativos, já que algumas páginas não precisam ser totalmente recarregadas a todo o momento [CRANE&PASCARELLO, 2006], diminuindo assim a quantidade de dados trafegados na rede, conforme figura abaixo.

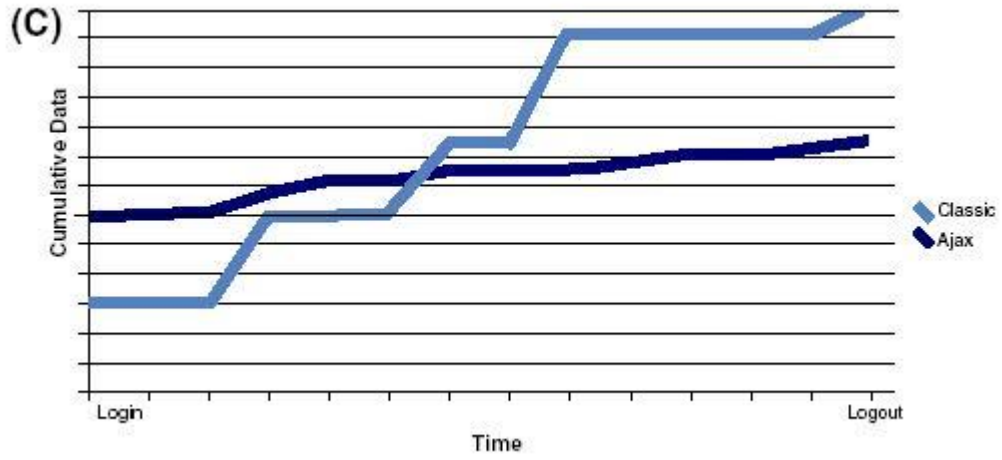

Gráfico 1 - Comparação dos dados enviados do servidor para o cliente. Fonte [CRANE&PASCARELLO, 2006]

Como observado no gráfico, apesar de o número de dados que trafegam entre o cliente e o servidor de uma aplicação que utiliza o conjunto de tecnologias AJAX ser 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:

Com a utilização do AJAX pode-se perceber 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.

4.6 Desvantagens da utilização da tecnologia AJAX

- Aumento de uso de *JavaScript*:

Uma das conseqüências da utilização do AJAX é o aumento significativo do uso da tecnologia *JavaScript*, muito eficiente, porém causadora de muitos transtornos na manutenção de sistemas. Esta degradação acontece porque com o uso dessa linguagem uma parte da lógica é colocada na camada de Visão do sistema, dificultando assim a compreensão do código do programa.

Como se não bastasse, ainda há diferentes formas em que os navegadores trabalham com esta linguagem, o que funciona no Internet Explorer pode não funcionar em outro *browser* como no Mozzila Firefox, por exemplo,

Adicionalmente, o usuário pode desabilitar o serviço de *JavaScript* de seu *browser* e com isso qualquer página que contiver AJAX não funcionará corretamente.

- Botão voltar do navegador:

Utilizando o AJAX é possível modificar apenas alguns valores mostrados em uma página, sem alterar o endereço (URL). Podem então ocorrer problemas ao se clicar no botão **voltar** do navegador, já que este irá mostrar a página do último endereço contido no *browser*. Por este mesmo motivo algumas páginas podem não aparecer no histórico do navegador.

4.7 Aplicação Exemplo

Nesta seção será apresentado o mesmo exemplo da seção “3.6 Aplicação Exemplo”, porém usufruindo da tecnologia AJAX e tentando assim obter um aplicativo mais interativo do ponto de vista do usuário.

O código-fonte deste exemplo encontra-se no capítulo “Anexo” na seção “10.2 Exemplo JSF + AJAX”.

As principais vantagens obtidas pelo uso deste conjunto de tecnologias no exemplo construído foram:

Ilustração 9 - Exemplo de programa usando JSF e AJAX - Verificando senha “fraca”

- Senha:

Neste exemplo, quando o usuário tirar o foco do campo de senha, ocorre a execução de um método Java utilizando o AJAX para verificar se a senha informada possui números e letras que a tornam uma senha mais adequada do ponto de vista de segurança computacional.

Depois da invocação deste método, caso a senha não atenda aos requisitos definidos, o sistema exibe ao lado do campo **Senha** uma mensagem em vermelho: “A senha deve conter letras e números”, como mostrado na figura referenciada.

- CPF:

A validação não ocorre mais em uma classe *CPFValidator*, como no exemplo da seção “3.6 Aplicação Exemplo” Agora a validação ocorre quando o usuário tira o foco do campo

invocando um método usando AJAX.

Caso o CPF não seja válido, o sistema imediatamente mostra ao usuário a mensagem de que o valor informado para este campo está irregular.

- E-mail:

A validação do valor de E-mail é invocada toda vez que o usuário tira o foco deste campo. Caso o usuário não tenha colocado o caractere "@", o sistema mostra uma mensagem de "E-mail inválido".

- Escolha do estado e da cidade:

Toda vez que o usuário selecionar um estado (UF), o sistema executa um método, o qual disponibiliza ao usuário na segunda *selectOneListbox* todas as cidades associadas ao estado selecionado, sem a necessidade de renderizar toda a tela novamente.

Como pôde ser visto, em todas as situações citadas no exemplo, com o uso do conjunto de tecnologias AJAX, a página não precisa ser totalmente reconstruída para mostrar a mensagem de erro, executar validações ou disponibilizar as cidades do estado selecionado.

4.8 Conclusão

Conclui-se que o grande benefício desta tecnologia, tão comentada, utilizada e pesquisada, é a interatividade aplicada aos 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. Adicionalmente, o AJAX permite também a atualização de apenas alguns dados nas telas dos sistemas, sem a necessidade de renderização de toda página.

5 FORMAS DE INTEGRAÇÃO

5.1 Introdução

Neste capítulo será realizada uma análise aprofundada das formas de integração da tecnologia AJAX e do *framework* JSF. Serão analisadas uma série de publicações, entre as quais se destacam o artigo "Using JavaServer Faces Technology with AJAX" [MURRAY&NORBYE&BURNS, 2005] de Greg Murray, Tor Norbye e Ed Burns e o livro "Pro JSF and AJAX" [JACOBI&FALLOWS, 2006a], escrito por Jonas Jacobi e John R.Fallows.

A integração entre a tecnologia AJAX e o *framework JavaServer Faces* pode ser dividida inicialmente de duas maneiras: com o uso dos componentes JSF, adicionando na página as funcionalidades AJAX ou com a confecção de componentes personalizados.

5.2 Sem a construção de componentes personalizados

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

5.2.1 Integrando os componentes JSF com AJAX por intermédio de funções JavaScript

É comentada a seguir a forma de integração apresentada no artigo "Using JSF with AJAX" [MURRAY&NORBYE&BURNS, 2005].

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.

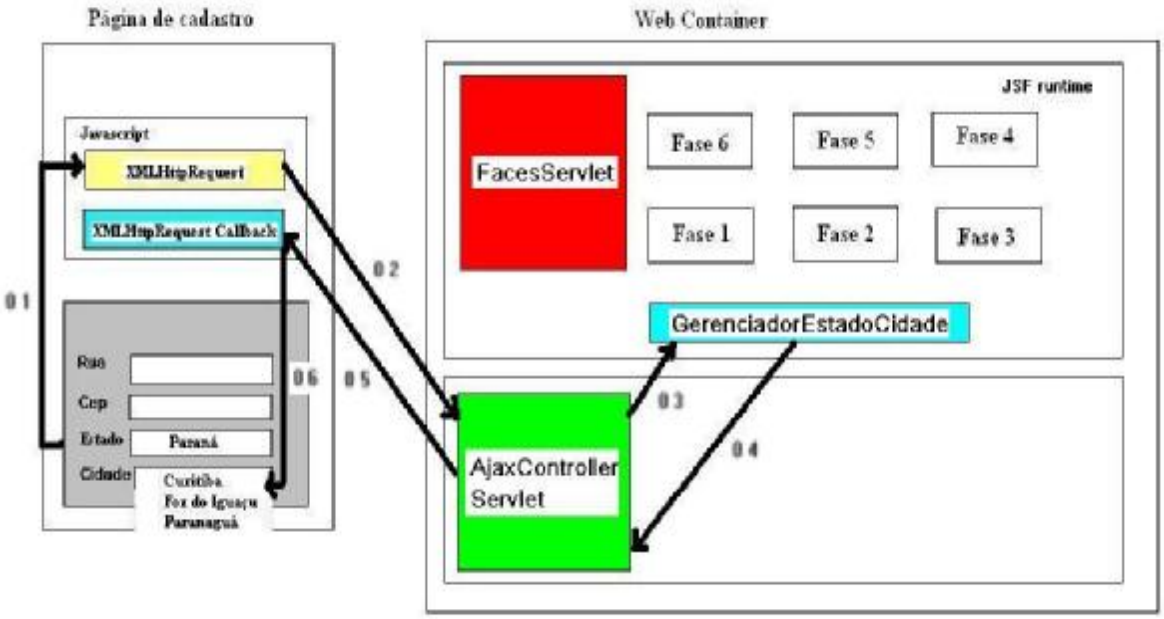

Ilustração 10 - Representação da Forma 3 de integração AJAX e JSF do artigo 1

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 (que será apresentada na seção 6.2.6 DWR deste trabalho). 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.

5.2.2 Modificando a especificação JSF

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

O principal objetivo é atualizar apenas os dados da página que foram modificados, não necessitando para tal renderizar toda a página novamente, nem criar classe adicional para a utilização de requisições assíncronas.

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.

Esta idéia tem como principais princípios o de reimplementar algumas classes da especificação JSF, entre elas: *UIComponent, UIComponentBase, UIViewRoot*. Assim, as atualizações da árvore DOM ocorreriam utilizando-se a tecnologia AJAX e só seriam atualizados os ramos que sofreram alguma alteração.

Modificando as classes citadas da especificação JSF, todos os componentes herdariam então a capacidade de utilizar a tecnologia AJAX, sem nenhum esforço adicional do desenvolvedor.

Testes iniciais desenvolvidos por Jacob Hookom demonstram a melhoria da eficiência com a adoção dessa idéia, de acordo com a tabela abaixo:

Tabela 1 - Comparação entre JSF e JSF com Avatar [HOOKOM, 2006]

	JSF	JSF + Avatar
Tempo de renderização	12.51ms	0.17ms
Quantidade de dados	17KB	89bytes

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.

Apresentada no último JavaOne [JAVAONE, 2006], maior conferência mundial sobre tecnologia Java, a idéia teve uma grande repercussão e aceitação por partes dos desenvolvedores que utilizam o *framework* JSF.

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

5.3 Desenvolvendo componentes personalizados

Neste modo de integração AJAX e JSF, o desenvolvedor é obrigado a construir componentes personalizados, o que inicialmente necessita de mais trabalho que a forma anterior (apresentada na seção **Erro! Fonte de referência não encontrada.**).

Entretanto, personalizando seus próprios componentes, o desenvolvedor obtém algumas vantagens: reutilizar componentes e dificultar a visualização, por parte dos usuários do sistema, das funções *JavaScript*.

Atualmente existem duas formas de integração da tecnologia AJAX com o *framework* JSF que envolvem esta idéia, dependendo do número de *Servlets* usados para isso, conforme explicado nas seções abaixo:

5.3.1 Usando apenas um Servlet

A idéia básica se sustenta no fato de que todas as requisições que utilizam ou não o AJAX devam passar pelo *Servlet* disponibilizado pelo *framework JavaServer Faces*, conhecido como *FacesServlet*, e modificar em algum momento o fluxo do ciclo de vida das requisições JSF para que o sistema usufrua dos benefícios da tecnologia AJAX.

A seguir serão apresentadas inúmeras idéias de integração entre AJAX e JSF que fazem uso de apenas um *Servlet*:

5.3.1.1 Utilizando um *PhaseListener* após a quinta fase do ciclo de vida

Esta é uma forma de integração escrita no artigo "Using JSF with AJAX" [MURRAY&NORBYE&BURNS, 2005] e pode ser representada pela figura abaixo:

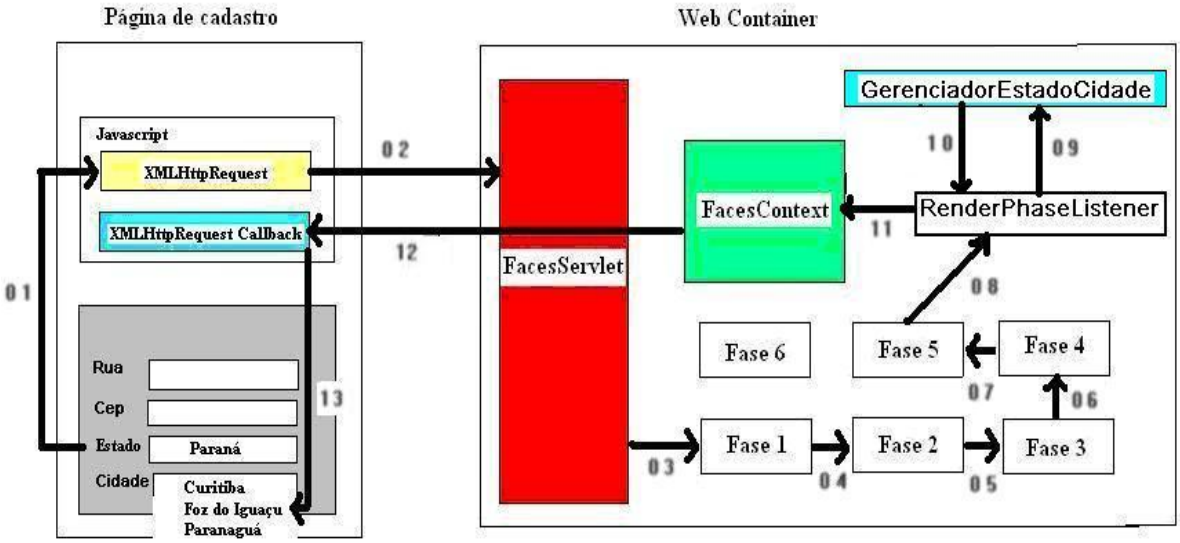

Ilustração 11 - Representação da Forma 1 de integração AJAX e JSF do artigo 1

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 e 10) 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&FALLOWS, 2006b].

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.

Um 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.

5.3.1.2 Reimplementando os renderizadores dos componentes

Esta é uma das formas de integração AJAX e JSF apresentadas no artigo "Super-Charge JSF AJAX Data Fetch" [JACOBI&FALLOWS, 2006b], conhecida como "The Renderer Approach".

São adicionadas funcionalidades ao Renderizador, que é disponibilizado pelo *framework* JSF, a fim de que consiga renderizar 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.

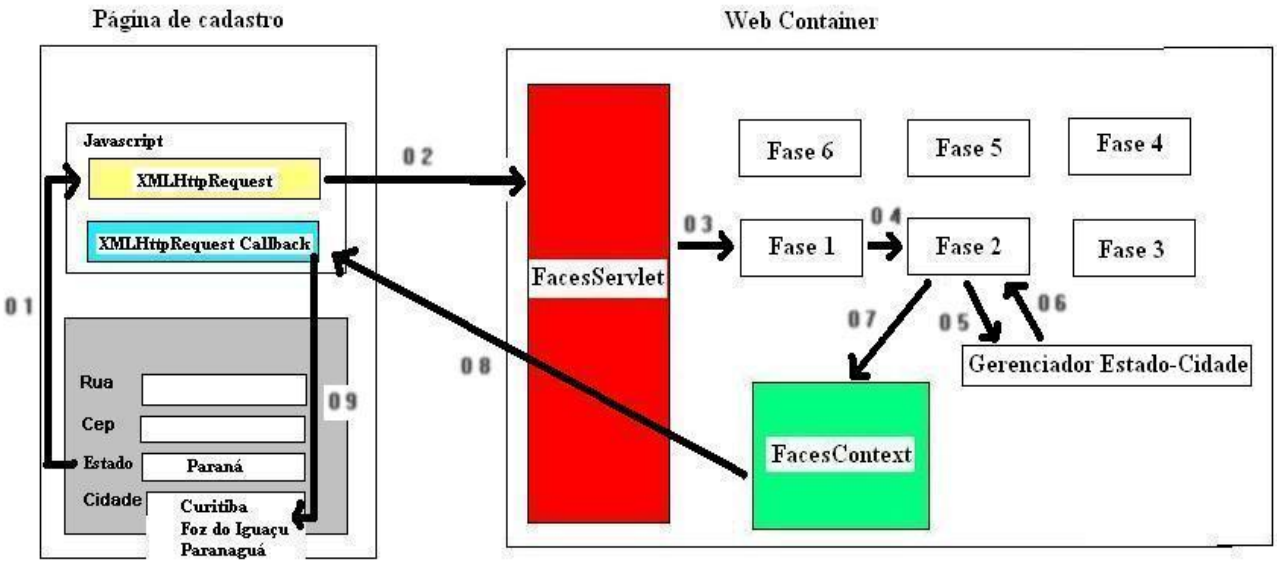

Ilustração 12 - Representação da Forma 1 de integração AJAX e JSF do artigo 2

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 e 6) 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.

5.3.1.3 Utilizando um *PhaseListener* após a primeira fase do ciclo de vida

Apresentada por Mark Basler [BASLER, 2006a], 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.

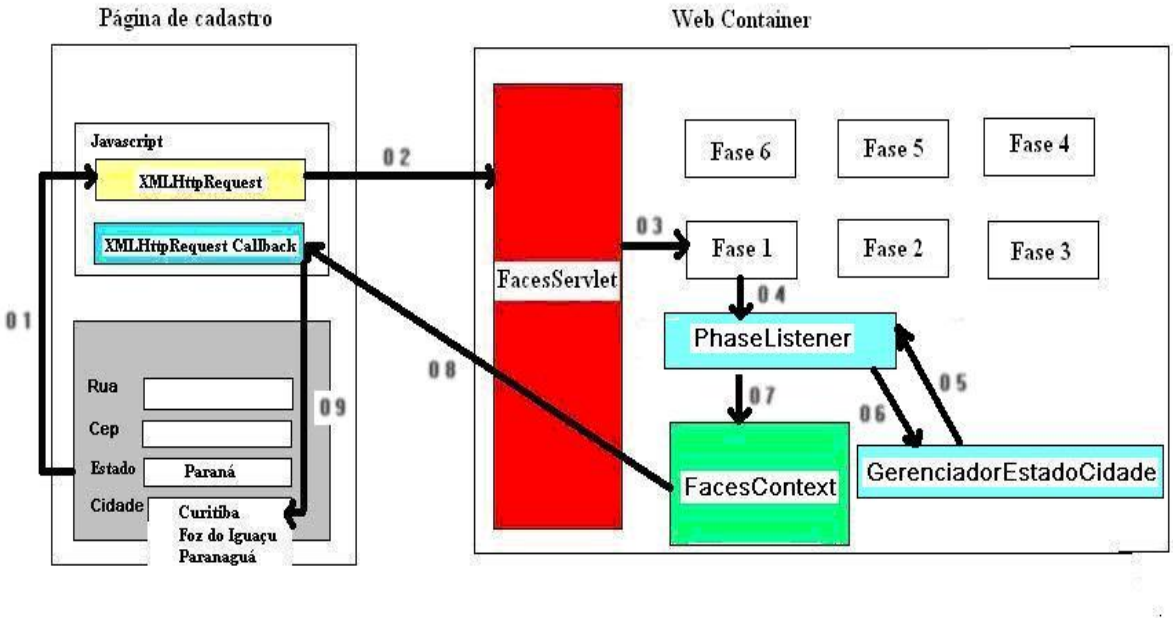

Ilustração 13 - Representação da Forma 1 de integração AJAX e JSF segundo Mark Basler

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 e 6) 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 “5.3.1.1 Utilizando 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 “5.3.1.1 Utilizando 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&BURNS&HOLMES, 2006].

5.3.1.4 Utilizando um *PhaseListener* antes da primeira fase do ciclo de vida

O artigo “Including AJAX Functionality in a Custom JavaServer Faces Component” [MURRAY&BALL, 2006] 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.

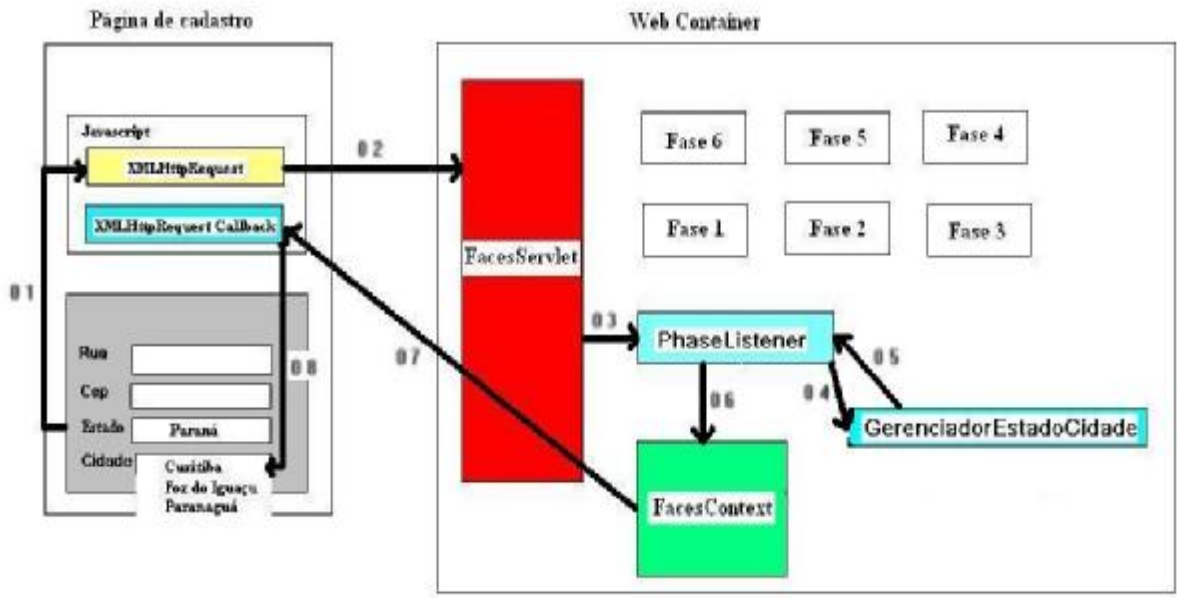

Ilustração 14 - Representação da Forma 1 de integração AJAX e JSF de “Super-Charge Ájax Data Fetch”

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 e 5) 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 “5.3.1.1 Utilizando um *PhaseListener* após a quinta fase do ciclo de vida” e “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.

5.3.1.5 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.

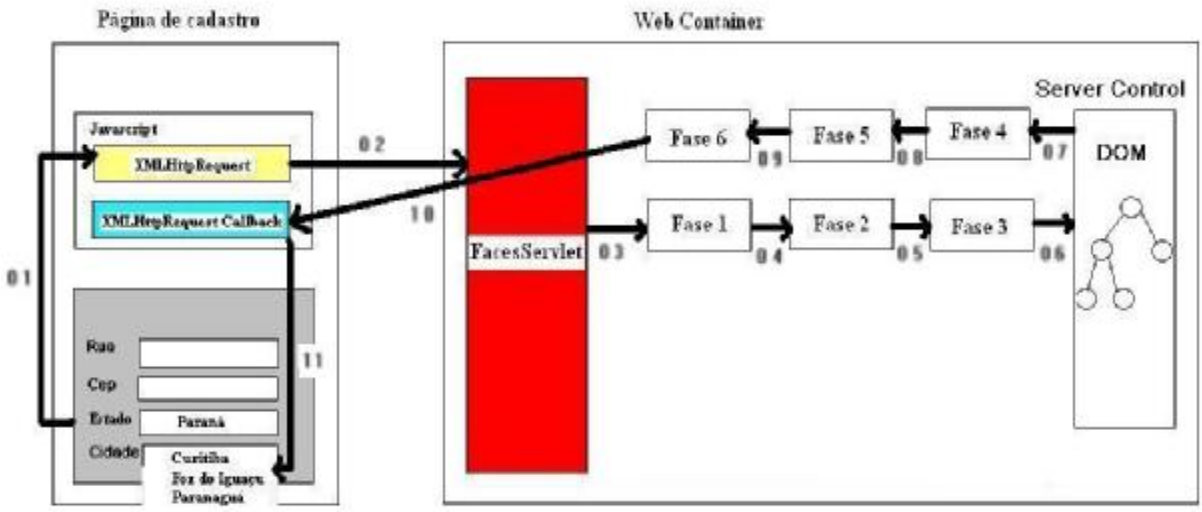

Ilustração 15 - Representação da forma de integração AJAX e JSF da empresa Backbase [BACKBASE,2003]

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.

5.3.1.6 Utilizando *PhaseListener* em três fases do ciclo de vida

Esta é a forma de integração AJAX e JSF utilizada pela empresa Exadel em sua biblioteca conhecida como Ajax4Jsf [EXADEL, 2006].

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.

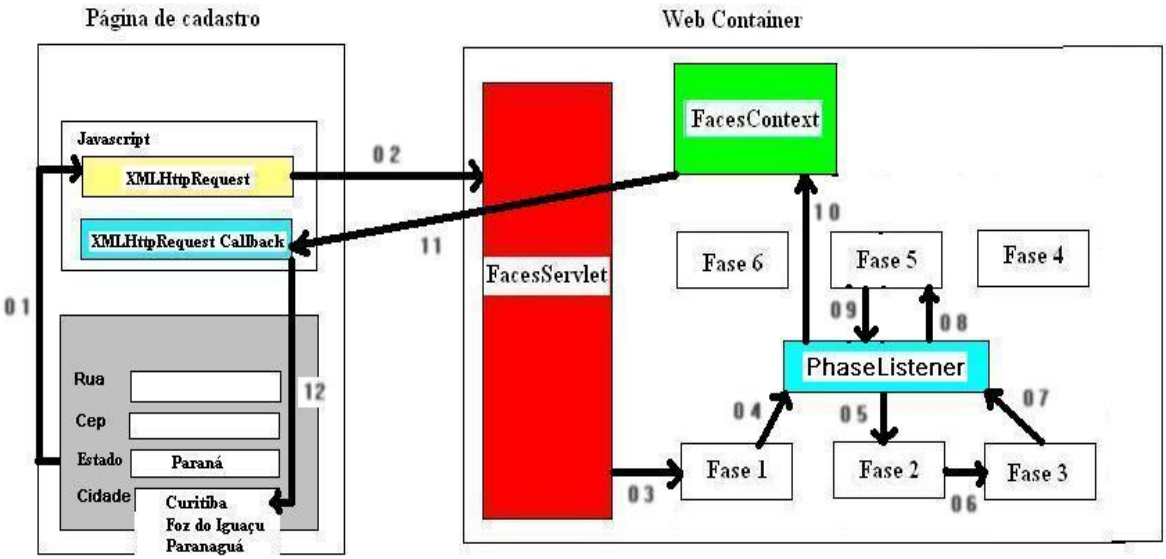

Ilustração 16 - Representação da forma de integração AJAX e JSF da empresa Exadel [EXADEL, 2006]

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 "5.3.1.1 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.

5.3.2 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 "5.3.1 Usando apenas um Servlet").

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

Segundo a apresentação de Jacob Hookom no último JavaOne [BURNS&HOOKOM&WINER, 2006], 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.

Segundo Philip Breau [BREAU, 2006], um dos desenvolvedores do projeto ICEfaces [ICESOFT, 2006], neste projeto 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.

5.3.2.1 Criando um *Servlet* para gerenciar requisições AJAX

Apresentada no artigo "Using JSF with AJAX" [MURRAY&NORBYE&BURNS, 2005], as ações executadas por uma requisição que utiliza esta idéia são apresentadas na figura a seguir.

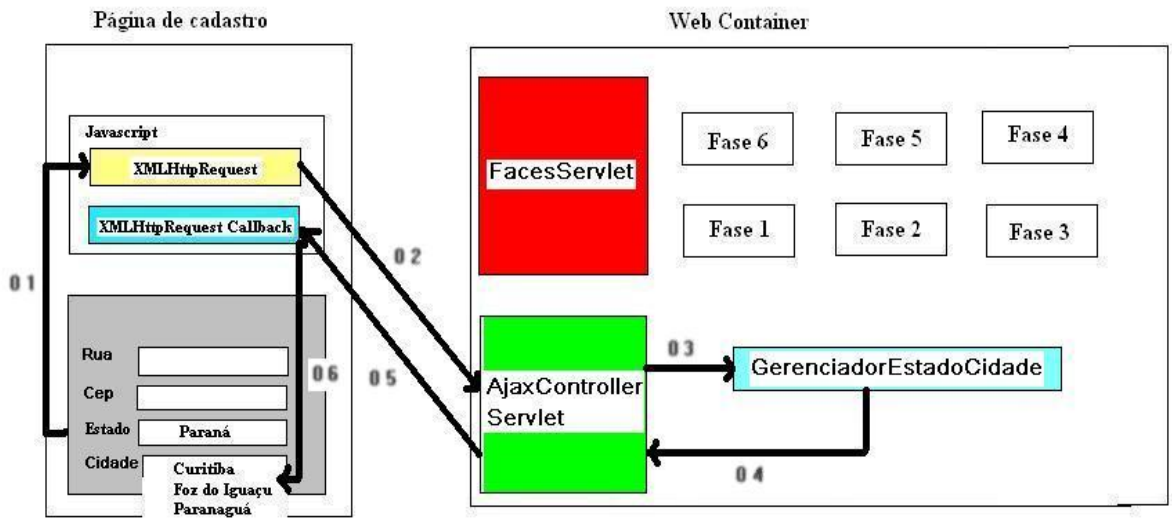

Ilustração 17 - Representação da Forma 2 de integração AJAX e JSF do artigo 1

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 C*allback* 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.*

5.3.2.2 Criando um *Servlet* para gerenciar as requisições AJAX e implementando um novo ciclo de vida para requisições AJAX

A terceira forma de integração apresentada pelo artigo “Super-Charge JSF AJAX Data Fetch” [JACOBI&FALLOWS, 2006b] 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.

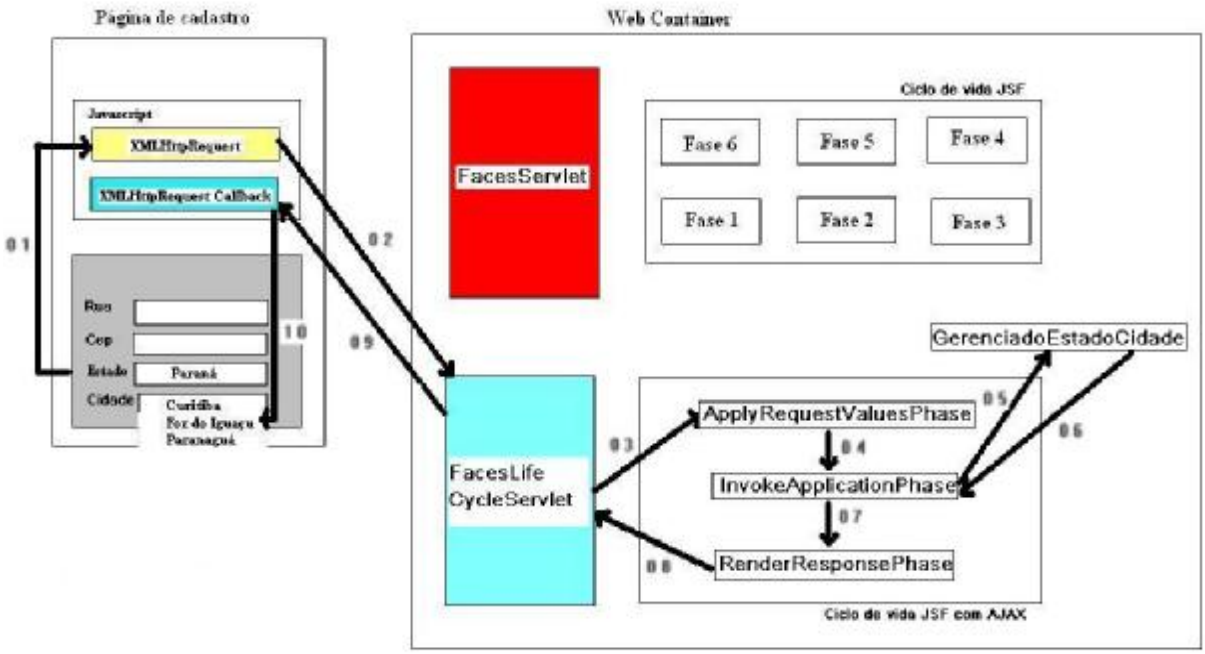

Ilustração 18 - Representação da Forma 3 de integração AJAX e JSF do artigo 2

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 e 6) 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.

Uma 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.

5.3.2.3 Criando um *Servlet* para gerenciar requisições AJAX e reimplementando os renderizadores dos componentes

A segunda técnica para integração AJAX e JSF apresentada por Mark Basler [BASLER, 2006b] 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:

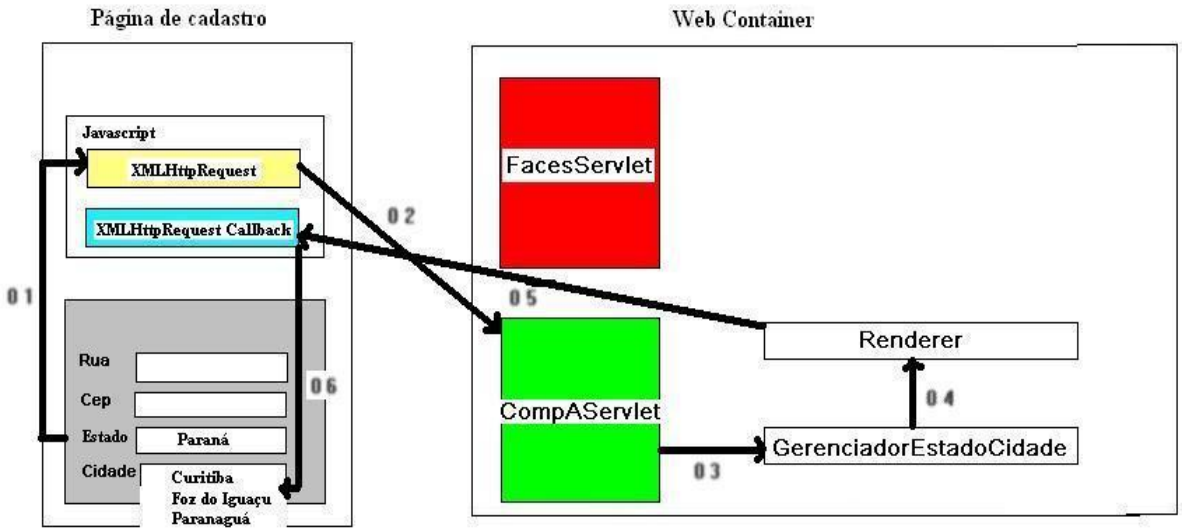

Ilustração 19 - Representação da Forma 2 de integração AJAX e JSF segundo Mark Basler

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.

Uma limitação está relacionada ao fato de que em todos os componentes, dos quais o desenvolvedor deseja obter os benefícios da tecnologia AJAX, deve ser desenvolvida uma nova classe *Renderer* para cada componente, para que nesta sejam empacotados os dados retornados ao cliente. Por essa razão, esta idéia é pouco aceita e utilizada pelos desenvolvedores.

5.4 Conclusão

No estudo deste capítulo se pôde perceber que há inúmeras idéias para a realização da integração entre AJAX e JSF. Foi verificado que muitas dessas idéias possuem defeitos, porém, ainda não há um consenso a respeito da melhor maneira de como efetivar tal integração.

Atualmente a idéia de integração AJAX e JSF 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) no JavaOne e que foi discutida na seção "**Erro! Fonte de referência não encontrada.**2 Modificando a especificação JSF".

A idéia básica é modificar a especificação JSF para que todos os componentes possam usufruir de requisições assíncronas sem a necessidade de nenhum trabalho adicional do desenvolvedor.

Devido a grande procura dos desenvolvedores para integrar AJAX aos aplicativos que utilizam o framework *JavaServer Faces*, os responsáveis pela especificação JSF pretendem incorporar essa tecnologia na versão 2.0 desse framework. A idéia apresentada acima é a mais cotada para ser utilizada, porém, não está definida a forma de integração a ser usada.

A segunda melhor forma de integração AJAX e JSF é a apresentada na seção "5.3.2.1

Criando um *Servlet* para gerenciar requisições AJAX" que, apesar de possuir alguns defeitos, ainda é muito utilizada devido à facilidade de uso e eficiência.

6 BIBLIOTECAS DE COMPONENTES DE INTEGRAÇÃO EXISTENTES

6.1 Introdução

Neste capítulo serão apresentadas algumas bibliotecas que fornecem componentes capazes de fazer a integração entre o *framework* JSF e o conjunto de tecnologias AJAX de maneira simples e eficiente.

Ao final será elaborado um estudo comparativo entre as referidas bibliotecas considerando alguns quesitos, tais como: custo, facilidade de uso, eficiência, documentação, forma de integração de JSF com AJAX, entre outros.

6.2 Bibliotecas de componentes existentes

6.2.1 ICEfaces

6.2.1.1 Introdução

O *framework ICEfaces* [ICESOFT,2006] pertence à empresa Rich Web Company que tem como principal foco desenvolver soluções para tornar as aplicações para a Web mais ricas em funcionalidades. Entre os principais clientes dessa empresa pode-se citar: IBM, BEA, Oracle, Borland, Cisco e Nokia.

Segundo seu próprio site, esta empresa é líder no mercado mundial de desenvolvimento de componentes que utilizam a tecnologia AJAX.

Ao ser criado, a empresa disponibilizava esse *framework* em duas versões:

- *ICEfaces Community Edition*:

  Esta era uma versão gratuita do *framework* e que possuía algumas limitações, como por exemplo, não poder solicitar suporte à empresa e também não poder ser usado para sistemas com fins lucrativos.

- *ICEfaces Enterprise Edition*:

Esta já era uma versão paga do *framework* (valores variavam de $1.750,00 a $5.000,00 dependendo do pacote de suporte) cuja licença poderia ser usada comercialmente.

Porém, no dia 14 de novembro de 2006, a empresa ICEsoft tornou *open source* essa biblioteca de componentes, cobrando apenas serviços de suporte, caso seja de interesse dos clientes.

As vantagens deste projeto se tornar *open source* já poderão ser observadas pelos usuários do *framework JavaServer Faces*, tendo em vista que algumas idéias e alguns membros do ICEfaces irão participar do desenvolvimento da especificação JSF2.0, na qual pretende-se fazer a integração com o AJAX.

6.2.1.2 Aplicação Exemplo

Com o estudo dessa biblioteca foi elaborada uma aplicação de cadastro de dados de pessoas, semelhante aos apresentados em seções anteriores, com o intuito de assimilar e analisar melhor as idéias propostas pela *ICEfaces*.

O código fonte do exemplo criado pode ser visualizado no Anexo na seção “10.3 Exemplo *ICEfaces*”.

A seguir serão demonstradas as telas do exemplo criado juntamente com suas respectivas particularidades:

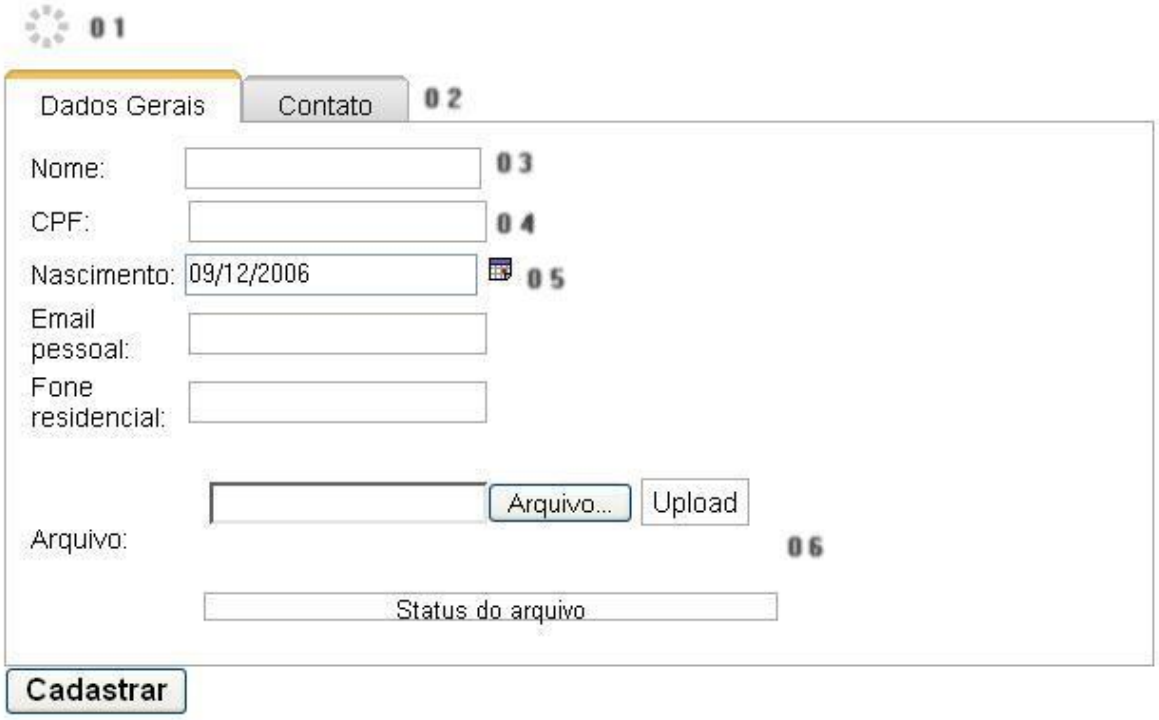

Ilustração 20 - Tela 1 do exemplo *ICEfaces*

01 – *Connection Status*:

Este componente, disponibilizado pela biblioteca *ICEfaces*, pode ser utilizado por meio da *tag* <*ice:outputConnectionStatus* />, cuja principal função é demonstrar ao usuário o status da conexão entre o *browser* e o servidor. O status dessa conexão pode ser visto de quatro diferentes maneiras:

- Ativo: indica que a conexão está ativa e que existe uma requisição sendo executada.
- Inativo: demonstra que apesar da conexão estar ativa não existe nenhuma requisição sendo executada naquele momento.
- Atenção: acusa ao usuário que a latência da conexão está alta.
- Desconectado: indica que a conexão foi perdida, podendo ser causada por falhas na rede ou na aplicação.

02 – *TabSet*:

Já este componente *TabSet*, utilizado por meio da *tag* <*ice:panelTabSet*>, possibilita ao usuário dividir a tela de cadastro de dados em várias abas, como no exemplo acima: Dados Gerais e Contato.

Apesar de ser um componente *ICEfaces*, que possui objetivo inicial de integrar AJAX e JSF, o *TabSet* foi desenvolvido utilizando apenas JavaScript.

03 – *AutoComplete*:

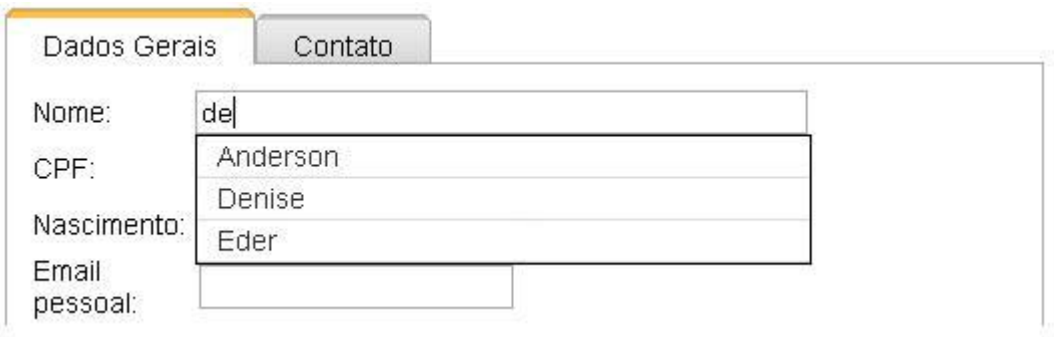

Ilustração 21 - Autocomplete exemplo *ICEfaces*

Este componente, disponibilizado pela biblioteca *ICEfaces* demonstra um dos maiores benefícios que podem ser obtidos com os componentes que utilizam a tecnologia AJAX.

A cada caractere digitado pelo usuário o sistema disponibiliza uma lista com valores que contenham a palavra digitada.

No exemplo acima demonstrado, quando o usuário digitou a palavra “de” no campo **Nome**, o sistema sugere todos os nomes que contenham o conjunto de caracteres inseridos. Nesse caso são sugeridos os nomes: Anderson, Denise e Eder.

04 – Validação:

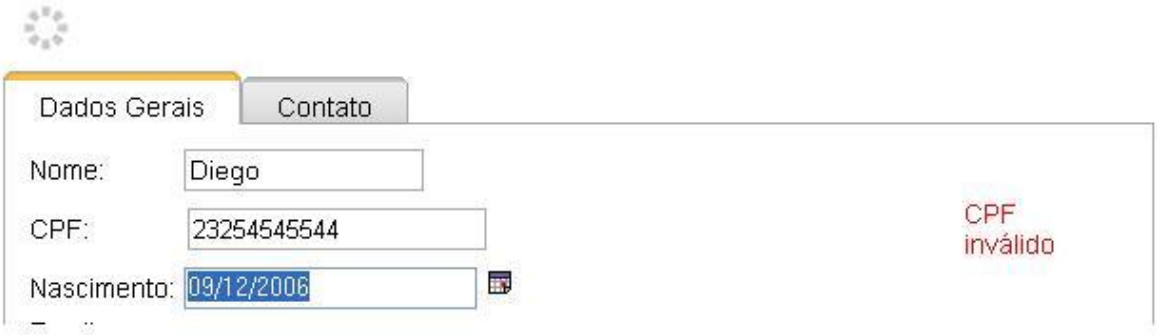

Ilustração 22 – Validação do campo CPF

Utilizando-se da característica *Partial Submit* da biblioteca *ICEfaces* juntamente com uma subclasse de *Validator* é possível fazer a validação dos dados que o usuário está preenchendo na página.

Como mostrado na figura acima, no exemplo desenvolvido é feita a validação do valor preenchido para o CPF quando o usuário tira o foco desse mesmo campo. Caso o valor não seja válido é mostrada então ao usuário uma mensagem de alerta.

05 – *InputDate*:

A biblioteca de componentes *ICEfaces* disponibiliza também um componente no qual o usuário pode escolher uma data pela janela pop-up, como pode ser visto na ilustração a seguir:

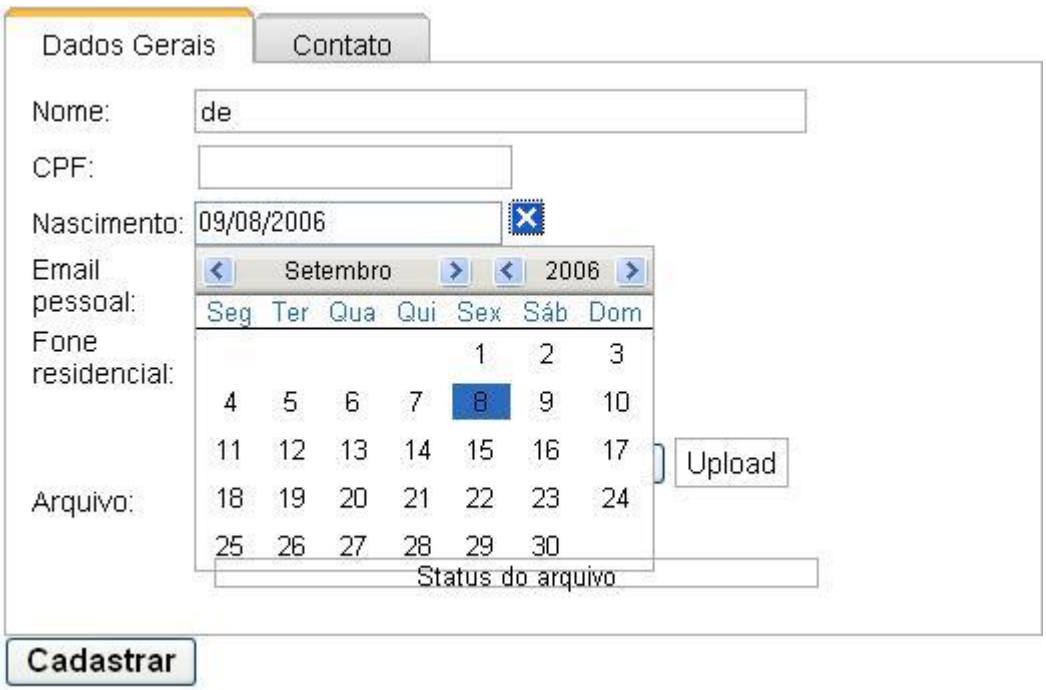

Ilustração 23 - inputDate exemplo ICEfaces

Este é outro componente da biblioteca *ICEfaces* desenvolvido utilizando-se somente a tecnologia JavaScript.

06 – *InputFile* e *Progress*:

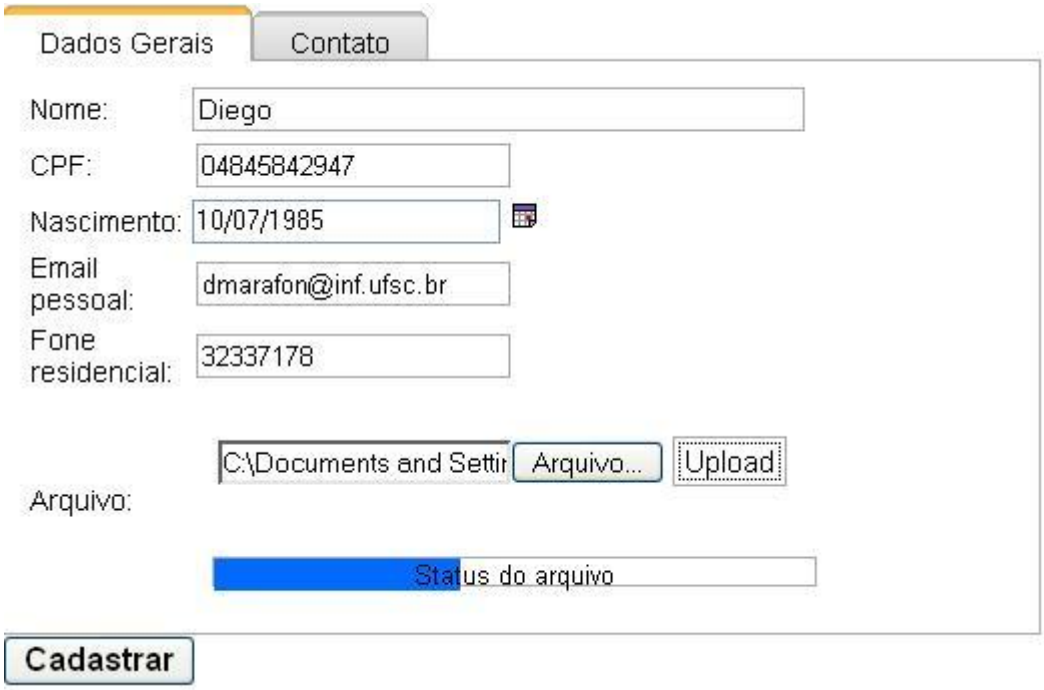

Ilustração 24 - inputFile e outputProgress exemplo ICEfaces

O componente *inputFile*, disponibilizado pela biblioteca *ICEfaces* por intermédio da *tag* <*ice:inputFile*> pode ser utilizado para que o usuário salve arquivos no servidor, recebendo os dados de maneira assíncrona.

Este componente pode ser acompanhado por uma *tag* <*ice:outputProgress*> que mostra ao usuário o progresso da ação de salvar o arquivo no servidor conforme a figura acima.

07 – *SelectOneMenu*:

Ilustração 25 - Tela 2 do exemplo *ICEfaces*

Na segunda tela do exemplo demonstra-se o componente *SelectOneMenu*, que permite que, por intermédio da tecnologia AJAX, os valores de uma lista sejam modificados quando o valor escolhido numa outra lista seja igualmente alterado.

No exemplo acima os valores da lista de cidades se modificam toda vez que o usuário selecionar um novo valor na lista de estados. Assim, a lista de cidades somente conterá os valores pertencentes ao estado selecionado.

08 e 10 – *DataPaginator:*

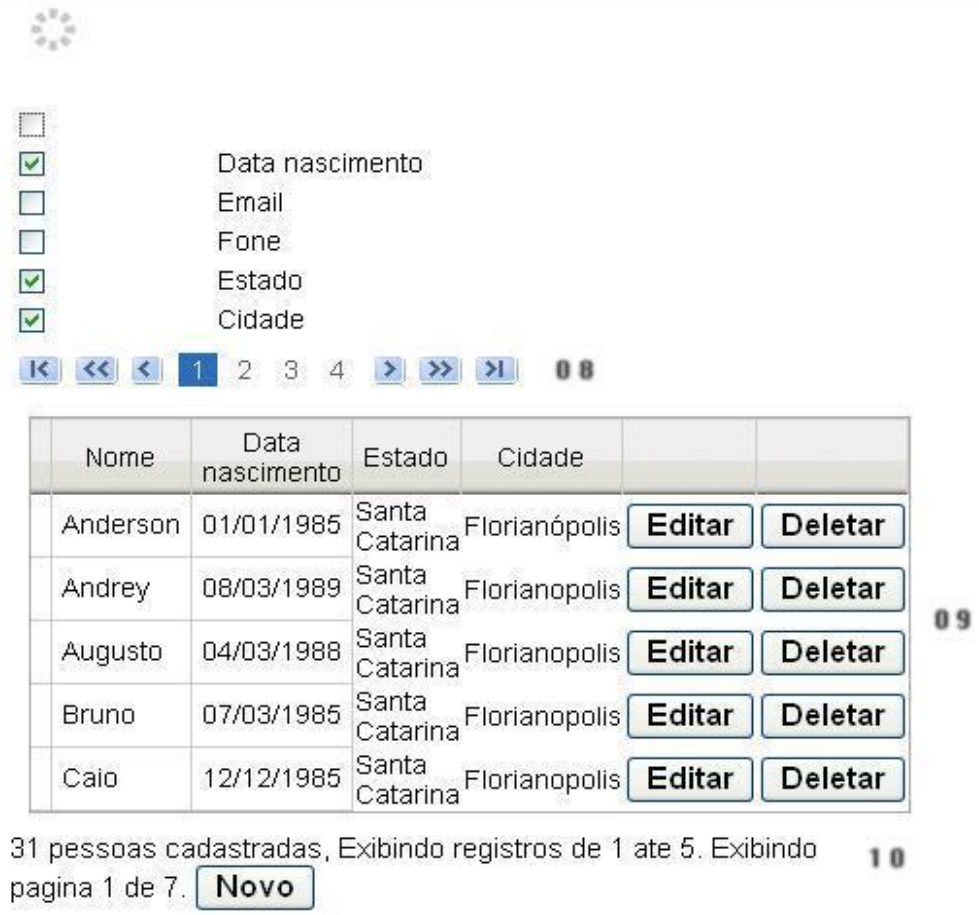

Ilustração 26 - Tela 3 do exemplo *ICEfaces*

O componente *DataPaginator* (dados paginados) da biblioteca *ICEfaces*, por meio da *tag* <*ice:dataPaginator*>, disponibiliza aos usuários botões, vistos ao lado do número 08 da figura acima, que permitem alterar a página da tabela de dados.

Também se pode exibir uma mensagem ao usuário com alguns dados da tabela, como pode ser observado ao lado do número 10. As mensagens que aparecem na figura acima são: “31 pessoas cadastradas. Exibindo registros de 1 até 5. Exibindo página 1 de 7”. Isso significa que:

- Existem trinta e um registros na tabela;
- Cada página conterá cinco registros;
- Existem sete páginas, porém está sendo exibida ao usuário a primeira;

09 – *DataTable*:

Este componente *DataTable*, disponibilizado pela *tag* <*ice:dataTable*>, é muito similar ao <*h:dataTable*> desenvolvido pelo JSF.

Uma peculiaridade do exemplo é que o usuário escolhe dinamicamente as colunas da tabela as quais serão renderizadas por meio dos *checkbox*, localizados no exemplo acima da tabela de dados.

6.2.1.3 Forma de Integração *ICEfaces*

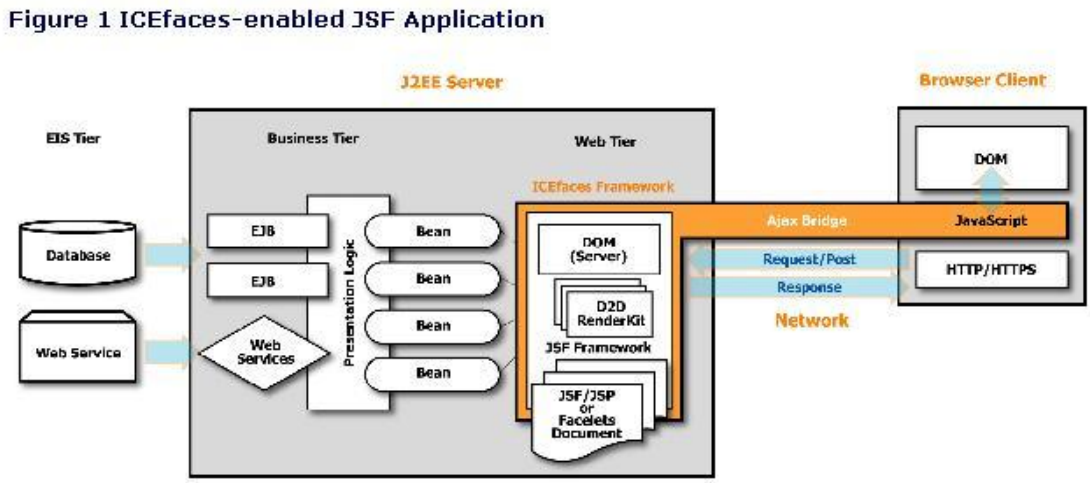

Ilustração 27 - Integração das tecnologias JSF e AJAX utilizada pela biblioteca *ICEfaces*. Fonte: [ICESOFT,2006]

Esta biblioteca integra a parte de *JavaScript* no *browser* do cliente com as camadas de Visão e Controle do modelo MVC do servidor por meio de um módulo *ICEfaces* chamado de *Ajax Bridge*.

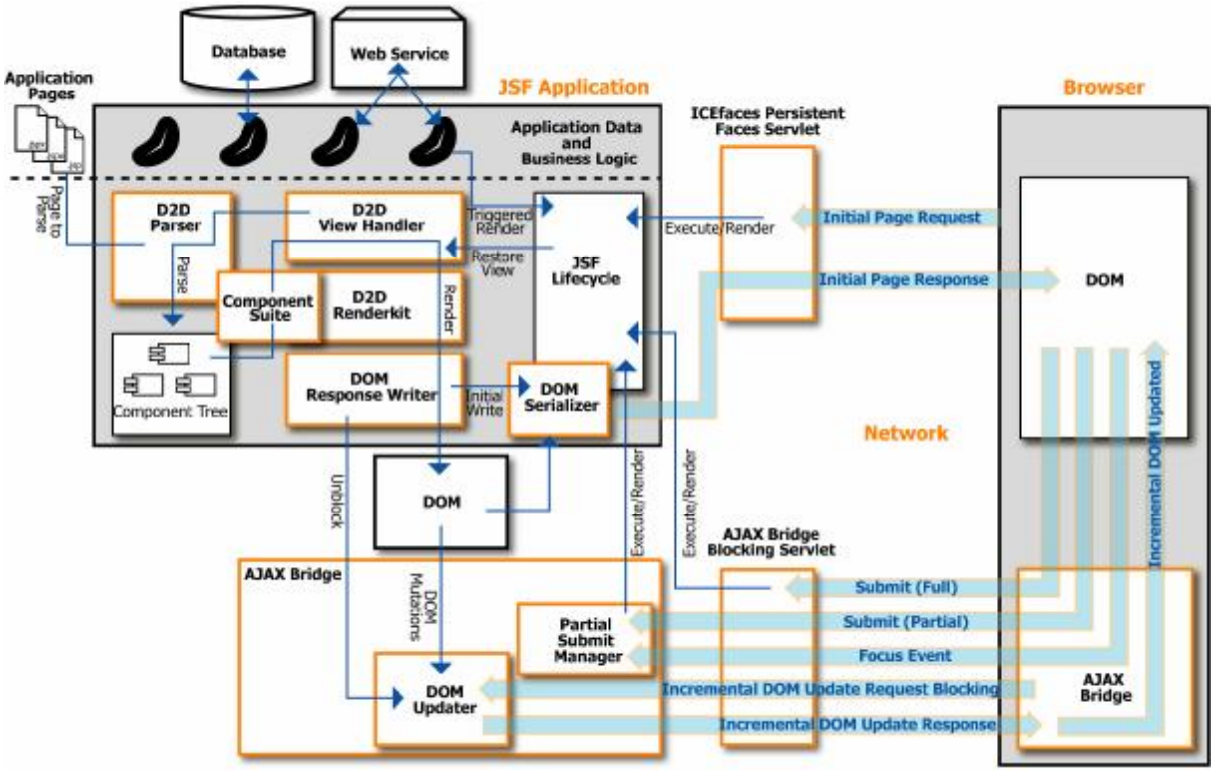

Ilustração 28 - Arquitetura *ICEfaces*. Fonte: [ICESOFT,2006]

As principais classes que pertencem à arquitetura *ICEfaces* são:

1 – *Persistent Faces Servlet*:

Todas as requisições iniciais advindas de arquivos ***.iceface** são encaminhadas para a classe *PersistentFacesServlet*. Esta fica responsável por repassar a requisição ao servidor que executará todas as etapas do ciclo de vida de uma requisição JSF.

2 – *Blocking Servlet*:

Depois de a página ser renderizada pela primeira vez, todas as requisições do *browser* passam pelo AJAX *Bridge* do cliente e são encaminhadas para AJAX *Blocking Bridge Servlet*. Esta classe tem a função de repassar a requisição caso esta seja um *submit* diretamente ao servidor; caso contrário, a requisição deve ser encaminhada ao AJAX *Bridge* do servidor.

3 – *D2D View Handler*:

Classe responsável por estabelecer a renderização direta com o DOM, que inclui entre outras atividades: a inicialização de um objeto da classe DOM *Response Writer* e a invocação de um objeto da classe D2DParser na primeira vez em que uma página é renderizada a fim de que seja feito o *parser* da árvore de componentes JSF.

4 – D2D Parser:

Cada vez que uma nova página é renderizada um objeto da classe D2D P*arser* é chamado para que execute o *parser* do arquivo e construa a árvore de componentes JSF.

5 – D2D *RenderKit*:

A principal função dessa classe é renderizar a árvore de componentes dentro do DOM.

6 – DOM *Response Writer*:

Esta classe é responsável por escrever a árvore de componentes no DOM e encaminhar essas alterações ao DOM *Updater*.

7 – DOM *Serializer*:

A principal função desta classe é a de "serializar" o objeto DOM para as páginas na primeira vez que é renderizada.

8 – DOM *Updater*:

A classe DOM *Updater* é responsável por capturar o objeto DOM atualizado e enviar esta instância para o AJAX *Bridge* localizado no browser do cliente para que seja renderizado.

Enquanto a tela com a árvore de componentes não for repassada ao AJAX *Bridge* do *browser*, o DOM *Updater* bloqueia qualquer requisição advinda da classe *DOM Response Writer*.

9 – *Client-side AJAX Bridge*:

Esta característica é implementada utilizando-se *JavaScript* e carregada automaticamente quando o *browser* renderiza uma página.

As principais funções do *Client-side AJAX Bridge* são: repassar requisições (*submit* ou *Partial submit*) para a classe AJAX *Bridge Blocking Servlet* e receber objetos DOM atualizados para serem renderizados para o usuário.

10 – *Server-side AJAX Bridge*:

Já as classes que pertencem ao módulo AJAX *Bridge* que ficam no servidor fazem parte da biblioteca *ICEfaces* e portanto são desenvolvidas em Java.

As principais atividades desenvolvidas por este módulo são: receber as requisições do *Client-side AJAX Bridge*, como por exemplo, *submit* e *partial submit*, que devem ser repassados à aplicação JSF; e obter objetos DOM atualizados que devem ser enviados para o AJAX *Bridge* do cliente.

Modos de Atualização:

A biblioteca *ICEfaces* disponibiliza ao usuário dois modos de atualização de dados: Síncrono e Assíncrono, porém, seja qual for o modo escolhido, não há modificação nas fases do ciclo de vida da aplicação JSF.

- Atualização Síncrona

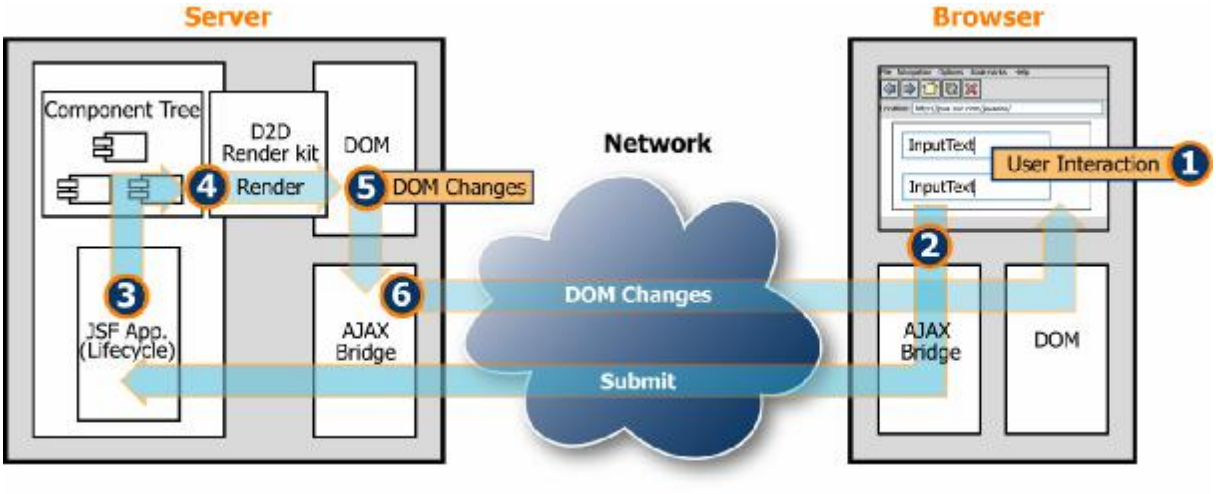

Ilustração 29 - Modo de atualização Síncrono. Fonte: [ICESOFT,2006]

Como a maneira de atualização síncrona não é o padrão da biblioteca *ICEfaces*, para utilizá-la deve-se selecioná-la no arquivo *web.xml*.

Como se pode observar na ilustração acima, as requisições, quando utilizam o modo de atualização síncrono, não usufruem do AJAX *Bridge* no *browser*, portanto, é necessário que toda a página seja totalmente recarregada a cada evento do usuário.

ü Atualização Assíncrona:

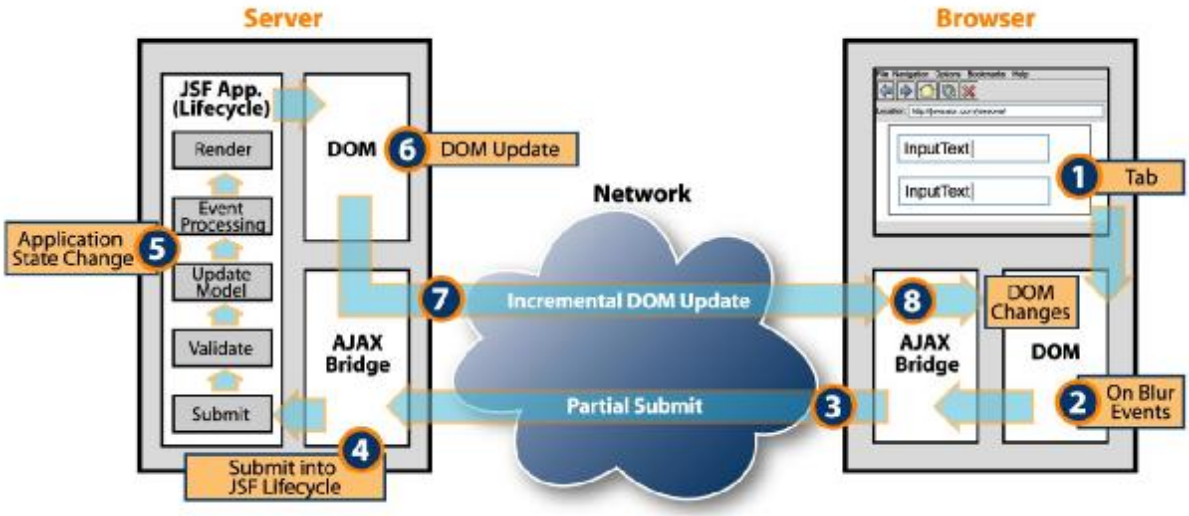

Ilustração 19 - Modo de atualização Assíncrono. Fonte: [ICESOFT,2006]

Neste modo de atualização padrão da biblioteca *ICEfaces* são utilizadas as AJAX *Bridges* tanto do servidor quanto do cliente e, portanto, é possível renderizar somente alguns campos da página exibida ao usuário.

6.2.1.4 Considerações Finais *ICEfaces*

Observa-se que esta biblioteca dispõe a seus usuários inúmeros componentes, além de documentação vasta e vários exemplos.

A forma de integração utilizada pela *ICEfaces* é similiar à apresentada na seção "**Erro! Fonte de referência não encontrada.** Criando um *Servlet* para gerenciar requisições AJAX."

6.2.2 AjaxFaces

6.2.2.1 Introdução

AjaxFaces [CYBERXP.NET ,2005] é uma biblioteca de uma empresa norte-americana chamada de *CyberXP.Net*, fundada em 2005, que possui como principal objetivo disponibilizar alguns componentes que integram *JavaServer Faces* e AJAX.

A primeira versão desta biblioteca foi liberada em 2006, porém, somente os seguintes

componentes fazem parte desta liberação:

- *Tree*

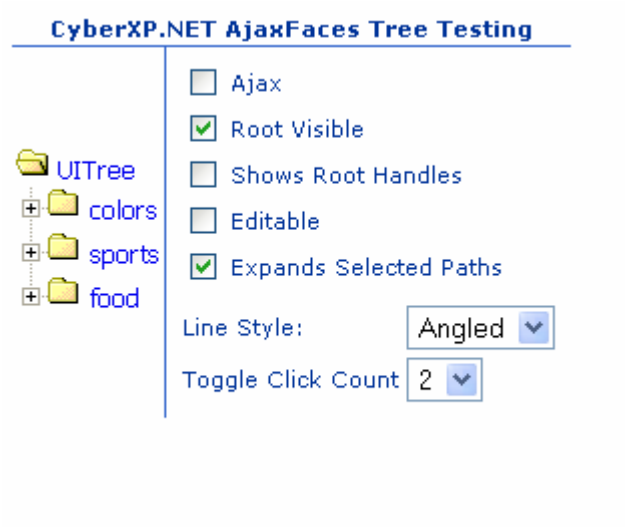

Ilustração 30 - Componente *tree* da biblioteca *AjaxfFaces*. Fonte: [CYBERXP.NET ,2005]

- *TabbedPane*
- *Calendar*

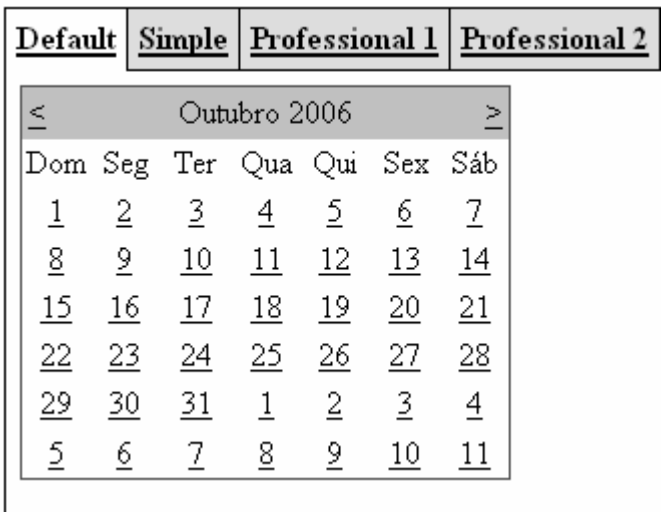

Ilustração 31-Componentes *TabbedPane* e *Calendar* da biblioteca *AjaxFaces*. Fonte: [CYBERXP.NET,2005]

Alguns outros componentes estão planejados para fazerem parte da próxima versão, dentre os quais estão:

- *Menu Bar*
- *PopupMenu*

- *PanelBar*
- *CommonTaskBar*

Esta biblioteca possui uma versão para demonstração, que funciona por até noventa e nove chamadas de requisição ao *AjaxServlet* para cada componente do *AjaxFaces*. Depois deste número é necessário pagar para continuar utilizando-a.

6.2.2.2 Aplicação Exemplo

A aplicação exemplo disponibilizada pela empresa *CyberXP.Net* apresenta problemas ao ser executada. Mesmo com as várias tentativas de contato (e-mail e fórum) realizadas desde julho de 2006 até a presente data, não se obteve resposta.

6.2.2.3 Forma de Integração *Ajaxfaces*

A solução de integração entre AJAX e JSF utilizada pela biblioteca Ajaxfaces possui os seguintes componentes:

- *HTMLAjaxScript*:

Conjunto de funções *JavaScript* que possuem a responsabilidade de invocar requisições do *AjaxServlet* e renderizar elementos na página exibida ao usuário utilizando para isso a tecnologia AJAX.

- *Servlet AjaxServlet*:

Este é uma classe Java que pertence à biblioteca *Ajaxfaces* e sua função é instanciar as requisições advindas do *HTMLAjaxScript* e repassá-las para o *FacesServlet* processá-las.

- *UIAjaxData*:

A biblioteca *Ajaxfaces* permite seu uso com ou sem recursos da tecnologia AJAX. Para ativar tais recursos em determinada página, esta deve conter em seu código a *tag* *<af_c:ajaxData>*.

A classe *UIAjaxData* é uma subclasse de *javax.faces.component.UIComponent* que é disponibilizada pelo *framework JavaServer Faces*.

- *UIAjaxItem*:

Esta classe também é uma subclasse de *javax.faces.component.UIComponent* fornecida pelo *framework* JSF e sua principal função é mapear quais componentes irão invocar alguma função por meio do *XMLHttpRequest* (utilizado pelo AJAX) ao sofrerem algum evento e quais componentes serão modificados devido à execução dessa função citada anteriormente.

Este mapeamento deve ser realizado fazendo uso das seguintes *tags* na página:

```
<af_c:ajaxItem
    sourceId="source_ui_component_id"
    targetId="target_ui_component_id"
    action="setOuterHTML"/>
```

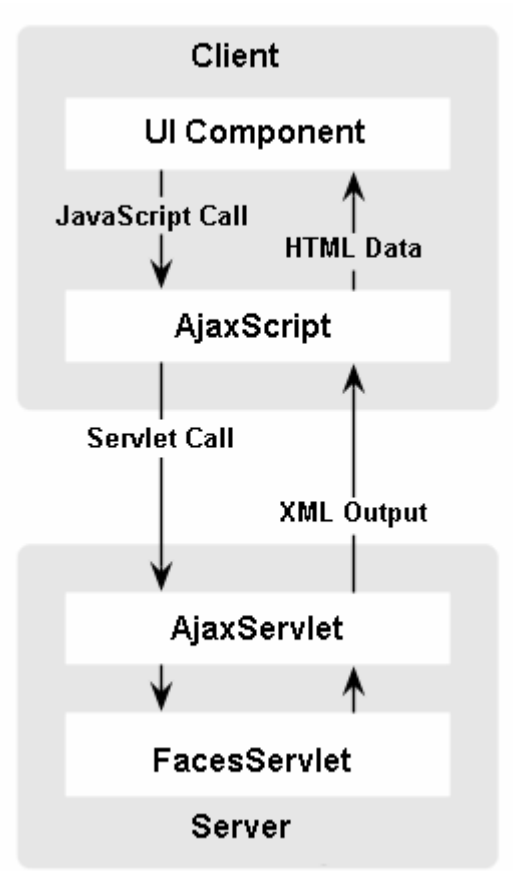

Ilustração 32 - Integração *Ajaxfaces*. Fonte: [CYBERXP.NET, 2005]

A forma de integração utilizada por esta biblioteca é similar à apresentada na seção "**Erro! Fonte de referência não encontrada.** Criando um *Servlet* para gerenciar requisições AJAX".

6.2.2.4 Considerações Finais *Ajaxfaces*

Finalmente, em relação à biblioteca *Ajaxfaces*, constata-se:

- apresenta um número reduzido de componentes e de documentação a respeito de seu uso;
- o exemplo disponibilizado não funciona;
- o site não está sendo atualizado;
- a empresa não responde aos questionamentos dos interessados.

Esses fatores têm gerado insatisfação dos usuários, conforme constatado nas listas de discussão da própria empresa, fazendo com que seus clientes busquem alternativas de mercado.

6.2.3 Ajax4jsf

6.2.3.1 Introdução

Lançada a versão 1.0 em 02 de agosto de 2006, o *framework Ajax4jsf* é um projeto patrocinado pela empresa Exadel [EXADEL ,2006] sediada na cidade de Concord, Califórnia.

O diferencial do *Ajax4JSF* é a facilidade de desenvolvimento de sistemas devido à total integração desse *framework* com a IDE(*Integrated Development Enviroment*), chamada de Exadel Studio.

Exadel também desenvolveu uma biblioteca de componentes JSF que utilizam a tecnologia AJAX, chamada de *RichFaces* [EXADEL, 2006a], a qual é uma extensão do *framework Ajax4jsf*. O valor do licenciamento de tal biblioteca é de $799,00 (setecentos e noventa e nove dólares) no primeiro ano e $399,00 (trezentos e noventa e nove dólares) a partir de então.

6.2.3.2 Aplicação Exemplo

Para avaliar a facilidade de uso, qualidade dos componentes disponibilizados, entre outros, foi desenvolvido neste trabalho um aplicativo-exemplo com o *framework Ajax4jsf* e a biblioteca *RichFaces*, cujo código fonte encontra-se no capítulo “Anexo”, na seção “10.4 Exemplo AJAX4JSF”.

Na seqüência são apresentados os componentes deste exemplo e suas respectivas particularidades.

- Status:

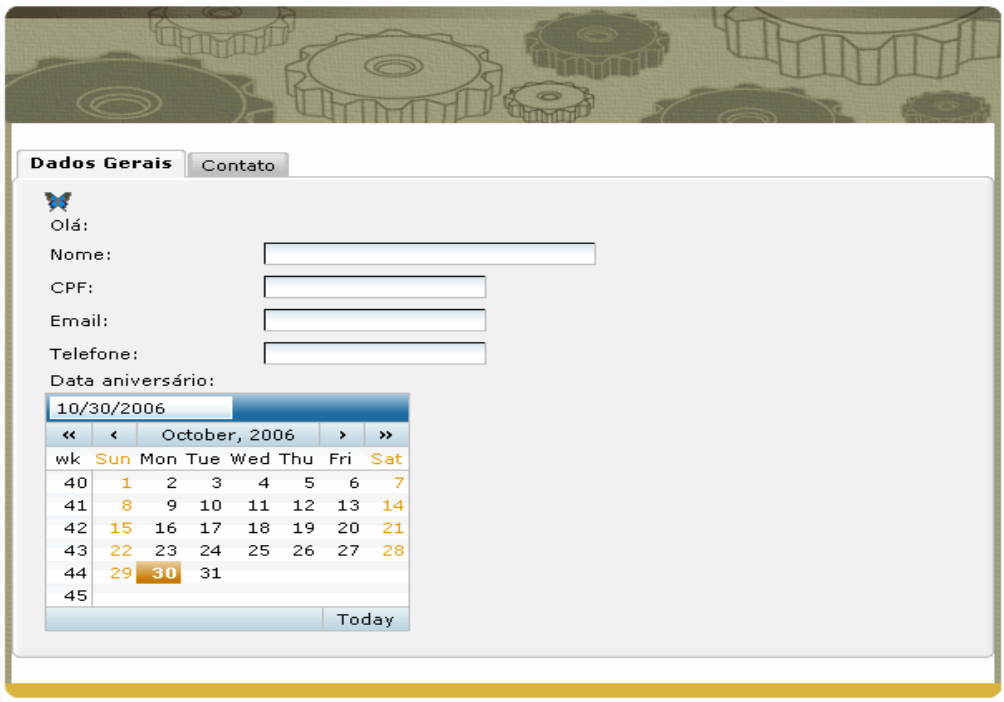

Ilustração 33 - Tela 1 do exemplo Ajax4Jsf

O componente <*a4j:status*> é representado inicialmente por uma borboleta, conforme observado na figura acima. Enquanto houver alguma requisição assíncrona sendo executada, tal figura é trocada pela animação de um planeta rodando.

ü *OutputText*:

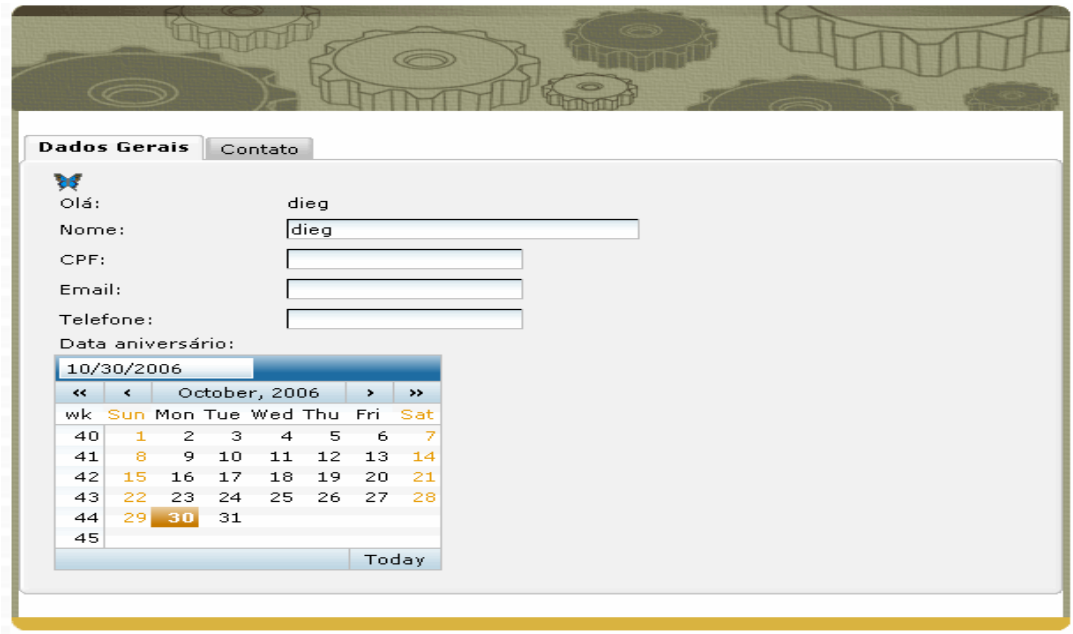

Ilustração 34 - Tela 2 do exemplo Ajax4Jsf

A cada tecla digitada pelo usuário no campo **Nome** o aplicativo-exemplo renderiza o conteúdo digitado ao lado da mensagem “Olá:”. Entretanto, este componente faz uso apenas de recursos de *JavaScript* e não da tecnologia AJAX.

ü *Suggestion Box*:

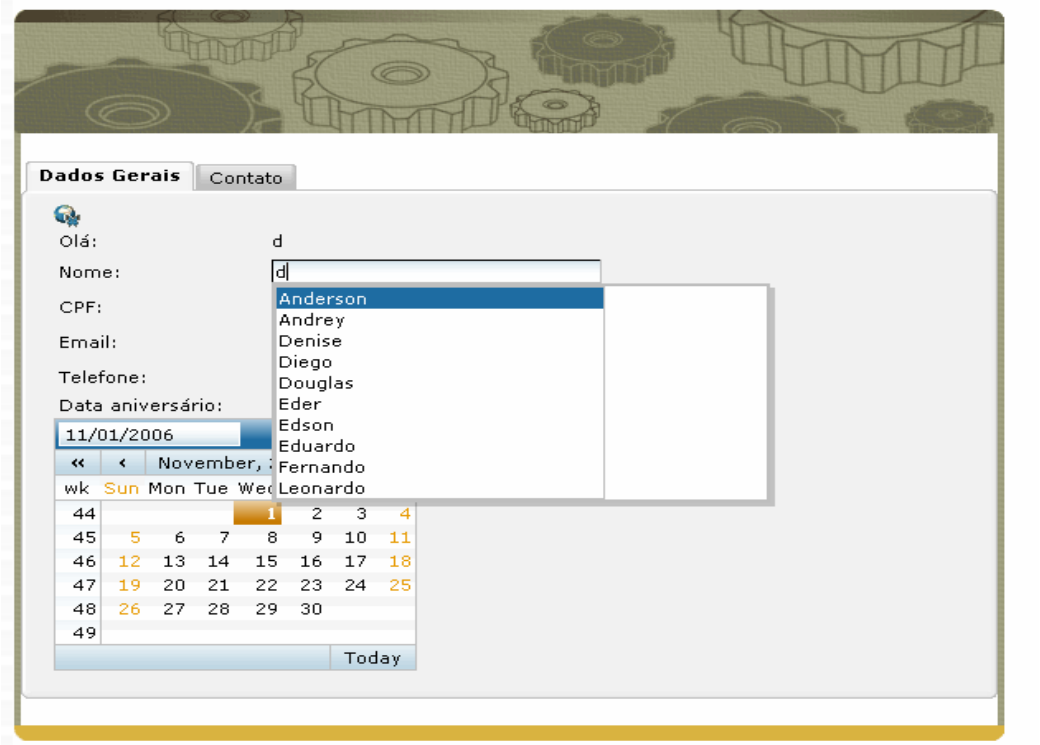

Ilustração 35 - Tela 3 do exemplo Ajax4Jsf

A cada tecla digitada no campo **Nome** uma requisição assíncrona é criada e invoca o método *suggestionAction* da classe *Autocomplete* para que o sistema exiba uma lista com sugestão de nomes que contenham o conteúdo digitado pelo usuário.

ü Validação:

A validação do CPF é realizada quando o foco sai deste campo, emitindo a mensagem “CPF inválido”, se aplicável.

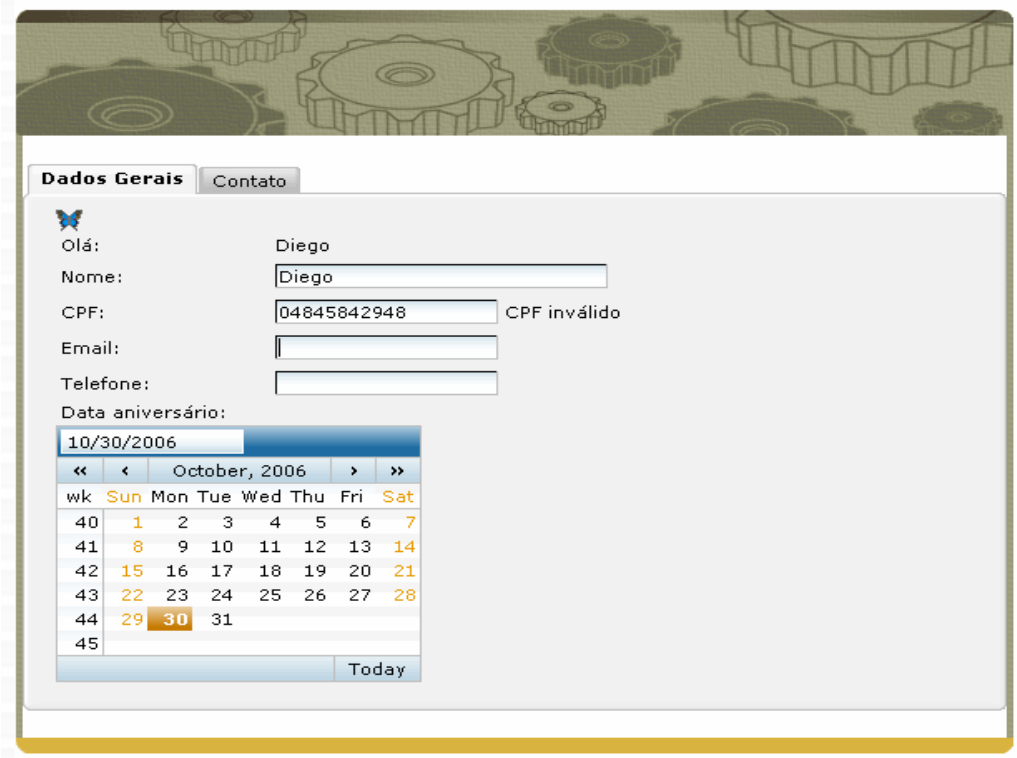

Ilustração 36 - Tela 4 do exemplo Ajax4Jsf

Esta mesma funcionalidade foi implementada também no campo **E-mail**, para verificar se o caractere “@” está no conteúdo digitado. Caso tal condição não seja atendida, quando o foco for retirado desse campo, o sistema emite a mensagem “E-mail deve conter o caractere‘@’”.

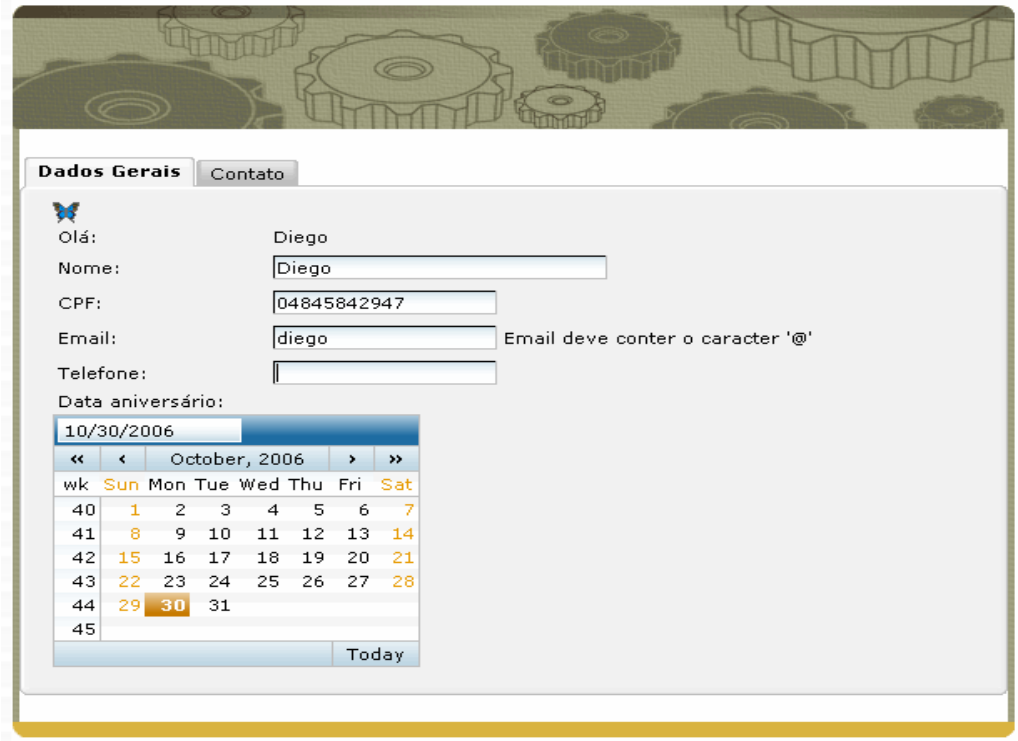

Ilustração 37 - Tela 5 do exemplo Ajax4Jsf

ü *Calendar*:

O componente *Calendar*, desenvolvido apenas com JavaScript, permite que o usuário selecione uma data de maneira simples e eficiente.

Ao selecionar o dia, o conteúdo dia/mês/ano é preenchido no campo **Data de Nascimento**, sem, no entanto necessitar que a página seja totalmente renderizada.

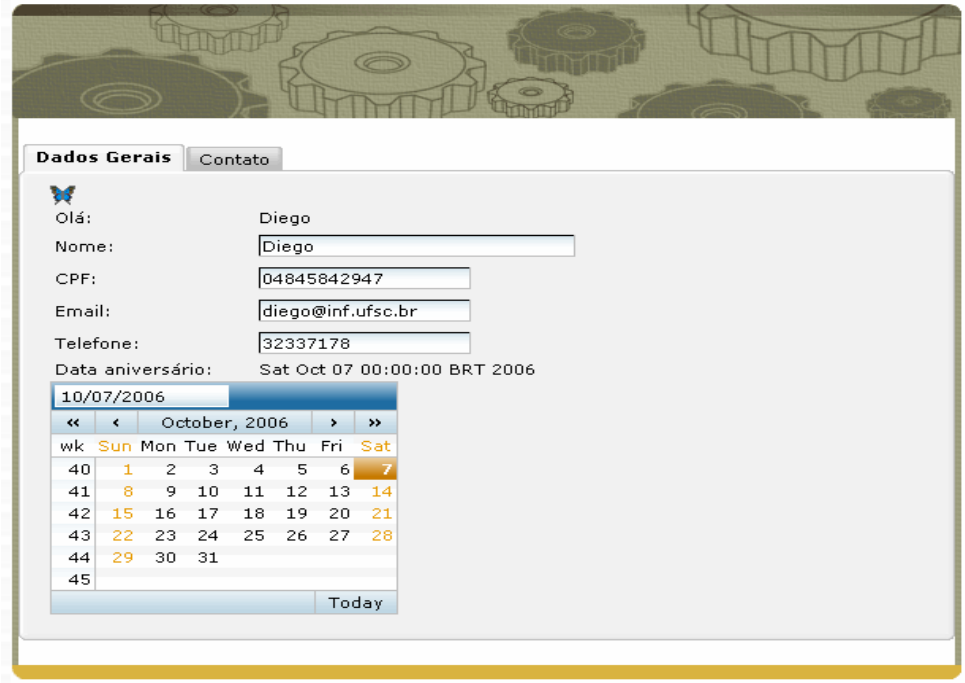

Ilustração 38 - Tela 6 do exemplo Ajax4Jsf

ü *SelectItems*:

Quando o usuário seleciona um determinado estado uma requisição assíncrona é criada para invocar o método *updateList* da classe *GerenciadorEstadoCidade* a fim de que sejam sugeridas as cidades que pertencem ao referido estado.

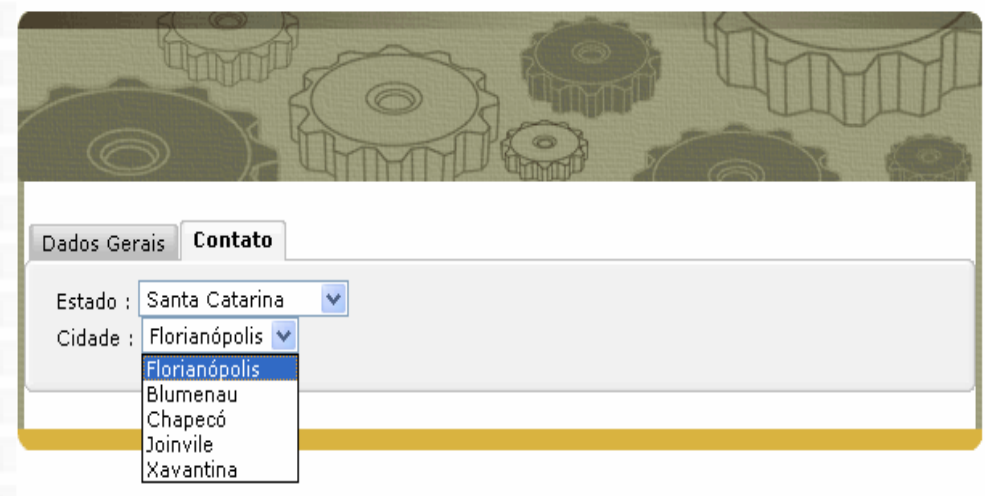

Ilustração 39 - Tela 7 do exemplo Ajax4Jsf

ü *TabPane:*

Este componente permite dividir uma página do sistema em várias abas. No exemplo são apresentadas duas abas: “Dados Gerais” e “Contato”. Este é outro componente que pode ser desenvolvido utilizando-se apenas de *JavaScript*, não necessitando do uso da tecnologia AJAX.

6.2.3.3 Forma de integração *Ajax4jsf*

Na figura abaixo os principais elementos que pertencem ao *framework Ajax4jsf* são:

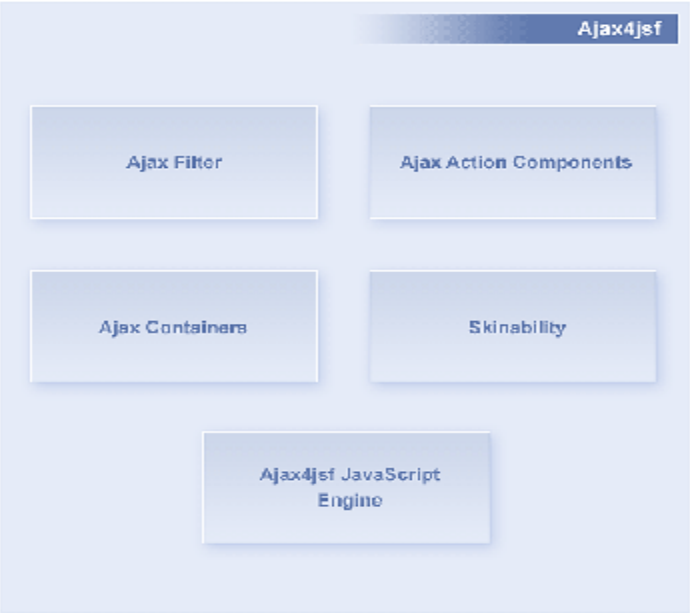

Ilustração 40 – Representação dos principais elementos da biblioteca Ajax4jsf. Fonte: [EXADEL, 2006]

ü *Ajax Filter*:

Este tipo de filtro tem como principal função a de receber as requisições e, de acordo com o tipo, encaminhá-las para serem executadas de maneira correta.

Caso seja uma requisição JSF, o *Ajax Filter* deve reconstruir toda a página que deve ser exibida ao usuário; caso seja uma requisição AJAX somente devem ser atualizados os componentes desejados pelo desenvolvedor.

ü *Ajax Container*:

É uma interface que contém quais partes da página JSF devem ser renderizadas após a execução de uma requisição.

ü *Ajax4jsf JavaScript Engine*:

Este é o responsável por administrar quais campos da página JSF do cliente deverão sofrer modificações quando uma requisição é executada.

ü *Skinability*:

Esta é uma característica desta biblioteca, que permite que algumas características gráficas sejam definidas com o uso de parâmetros do sistema; com isso o usuário pode definir a cor de fundo, o estilo e a fonte das letras, entre outros atributos, durante a execução do programa.

ü *Ajax Action Components*:

Existem basicamente três componentes que podem enviar requisições AJAX a partir do *browser* do cliente: *AjaxCommandButton*, *AjaxCommandLink* e *AjaxSupport*.

A figura a seguir demonstra o caminho seguido por uma requisição *Ajax4jsf* desde a ocorrência da ação do usuário até a página ser renderizada.

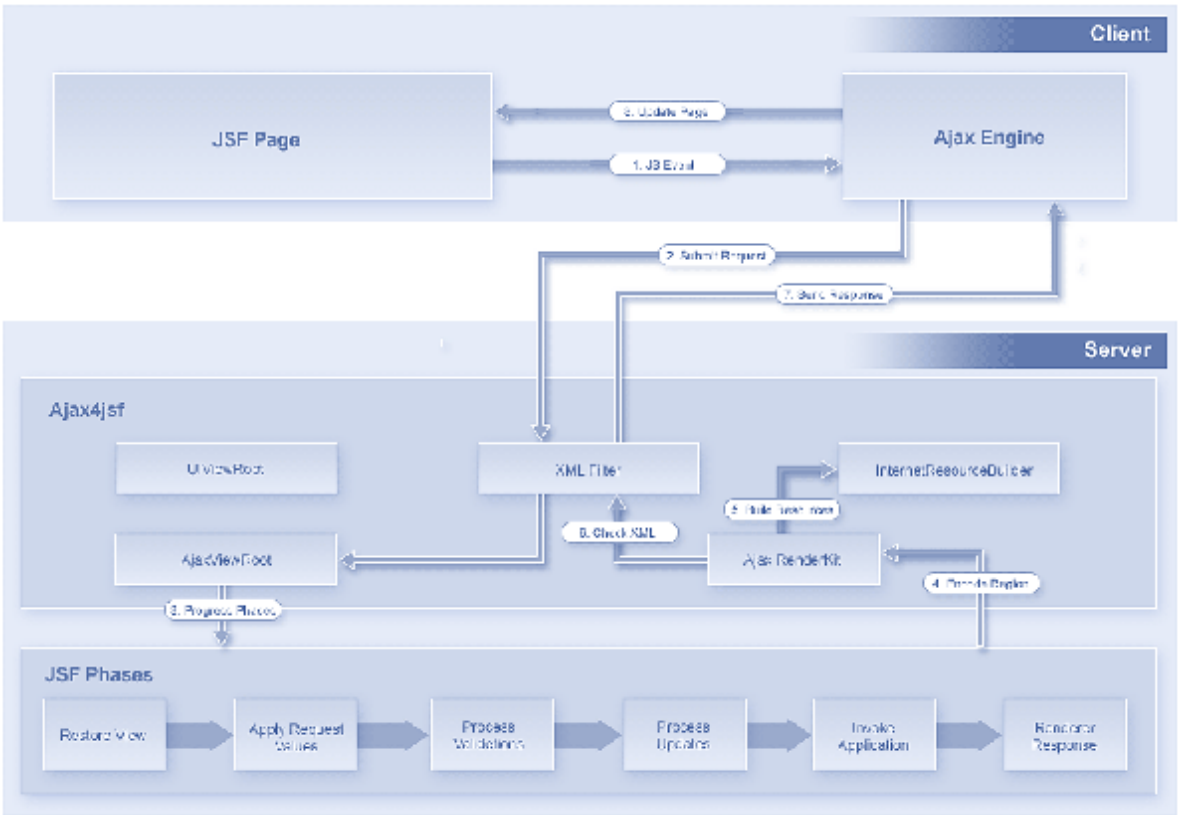

Ilustração 41 - Etapas realizadas por uma requisição *Ajax4jsf*. Fonte: [EXADEL,2006]

A idéia adotada por esta biblioteca é semelhante à apresentada na seção "**Erro! Fonte de referência não encontrada.**.6: Utilizando *PhaseListener* em três fases do ciclo de vida", visto que utiliza-se de apenas um *Servlet*, disponibilizado pelo JSF e conhecido como *FacesServlet*, e um *PhaseListener* em três fases deste ciclo.

6.2.3.4 Considerações Finais *Ajax4jsf*

Pode-se observar então que nem este *framework* nem sua biblioteca disponibilizam um componente muito procurado pelos desenvolvedores, conhecido como "Grid Paginada" e que foi apresentado na seção "6.2.1.2 Aplicação Exemplo ICEfaces", no item 8.

Uma falha encontrada na versão utilizada do Ajax4Jsf é que este *framework* não está ainda totalmente integrado aos diversos *browsers* (principalmente Internet Explorer e Mozzila Firefox) considerando que no exemplo desenvolvido os componentes ficam perfeitamente alinhados em um browser, enquanto em outro acontece o desalinhamento.

6.2.4 Backbase

6.2.4.1 Introdução

A empresa Backbase [BACKBASE,2003], que possui sede em Nova Iorque e Amsterdã, também dispõe de uma biblioteca de componentes que integram o *framework* JSF e a tecnologia AJAX.

Na página da referida empresa encontram-se exemplos e o código fonte de todos os componentes que são oferecidos. Interessante citar que em todos os exemplos disponíveis os usuários podem modificar o código fonte de tais componentes na própria página e em seguida testar as alterações promovidas.

Uma versão de demonstração para trinta dias é oferecida para quem desejar elaborar testes. O valor da licença anual é de US$ 7.000 (sete mil dólares).

6.2.4.2 Aplicação Exemplo

O código fonte do exemplo desenvolvido com esta biblioteca de componentes encontra-se no capítulo de “Anexo” na seção “10.5 Exemplo *Backbase*”.

A tela inicial do exemplo construído pode ser visualizada na figura a seguir.

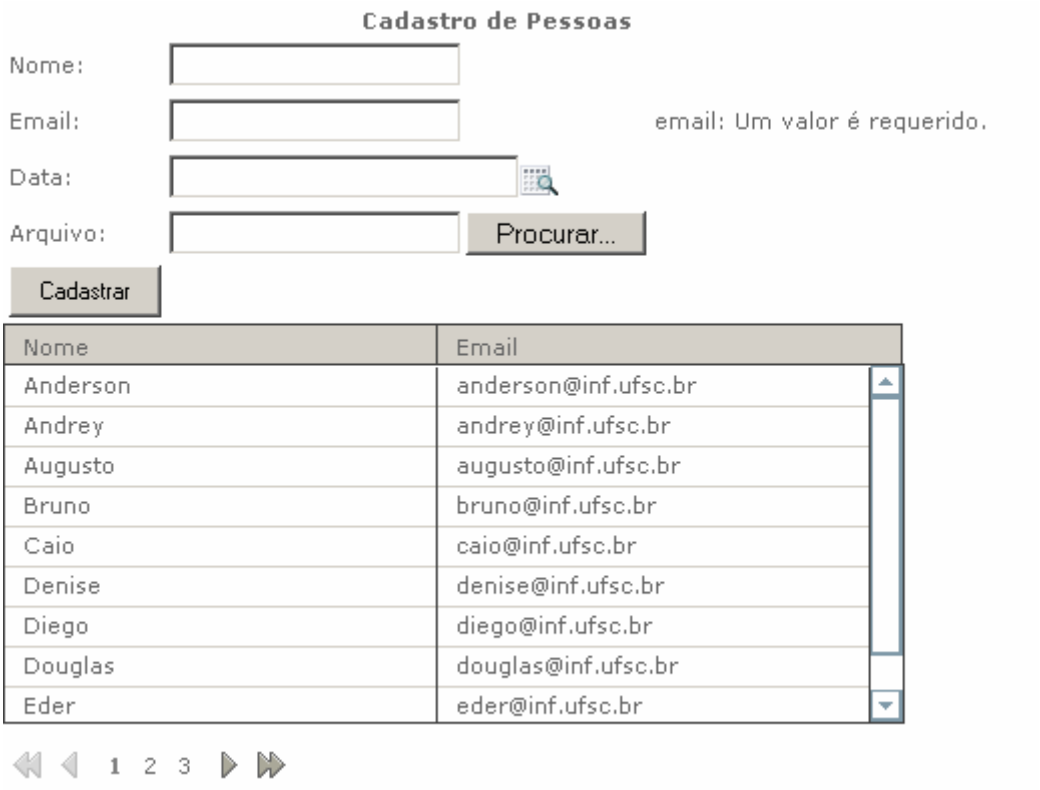

Ilustração 42 - Tela inicial do exemplo utilizando *Backbase*

O exemplo desenvolvido com esta biblioteca possui as seguintes funcionalidades:

- *Validator*

A biblioteca *Backbase* possui um componente que faz a verificação e a validação do conteúdo digitado pelo usuário no campo de **E-mail** utilizando a tecnologia AJAX.

Caso o valor do e-mail esteja em branco, o sistema exibe a seguinte mensagem: “E-mail: um valor é requerido”. Se o conteúdo digitado não for válido, o programa exibe a mensagem “O valor informado não é um endereço de e-mail válido”, conforme conferido na figura que segue.

Ilustração 43 – Componente exibindo mensagem de valor inválido no exemplo *Backbase*

- *DatePicker*

É um componente que permite ao usuário selecionar uma determinada data a partir do calendário.

A figura abaixo demonstra a ação deste componente, que é ativada quando o usuário clica no botão ao lado do campo **Data**.

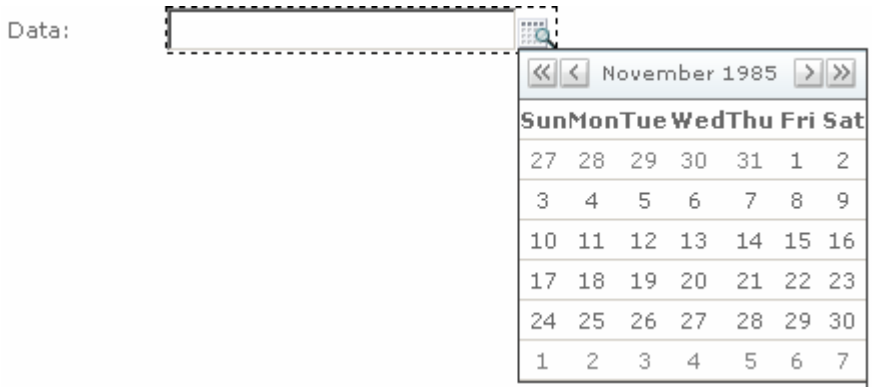

Ilustração 44 - Componente *DatePicker* da biblioteca *Backbase*

- *FileUploader*

Este componente permite ao usuário armazenar um arquivo utilizando para isso a tecnologia AJAX.

Ilustração 45 – Componente *FileUploader* da biblioteca *Backbase*

- *ListGrid*:

O *ListGrid* é um componente encontrado em poucas das bibliotecas que utilizam recursos de AJAX e suas funcionalidades são muito úteis, conforme a seguir comentado.

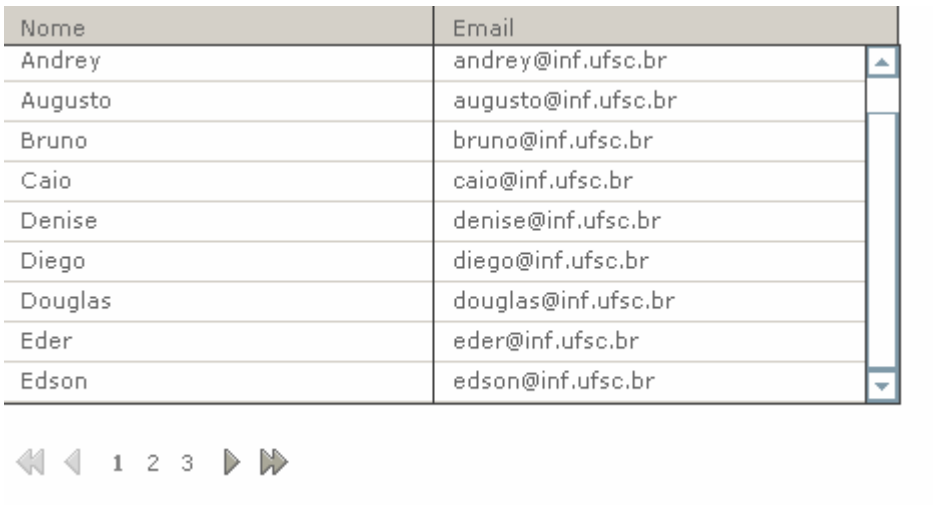

Ilustração 46 – Componente *ListGrid* da biblioteca *BackBase*

O componente permite listar uma série de registros em forma de tabela, dividida em páginas.

Uma outra funcionalidade é a ordenação dos elementos. No exemplo acima, ao clicar sobre as células da tabela "Nome" e "E-mail", o programa exibe os registros em ordem alfabética levando em consideração os conteúdos do nome e e-mail de cada registro.

6.2.4.3 Forma de Integração *Backbase*

A biblioteca *Backbase* estende o *framework* JSF para permitir que seus componentes ofereçam os benefícios AJAX.

A seguir são detalhadas as implementações desenvolvidas por meio da biblioteca *Backbase* para integrar AJAX e JSF:

- *Backbase* Presente no Cliente (BPC):

Este é um mecanismo que se aloja no *browser* do cliente, carregado na primeira vez que o usuário entra na página, e que contém a árvore com os estados de todos os componentes da página.

Ao receber a resposta de uma requisição, o BPC compara as árvores DOM recebidas pela requisição com a antiga e renderiza no *browser* do cliente apenas os componentes que tiveram alguma alteração.

ü *BackViewHandler*:

Esta é a classe responsável por desenvolver a renderização dos componentes, atualizando seus estados na árvore DOM.

Na biblioteca *Backbase* todos os componentes são responsáveis por guardar seu estado e em caso de mudança de seu conteúdo, o próprio componente deve notificar a instância *BackViewHandler* para que a mesma atualize a árvore DOM.

A figura demonstra o caminho seguido por uma requisição *Backbase* desde a ocorrência da ação do usuário até a página ser renderizada.

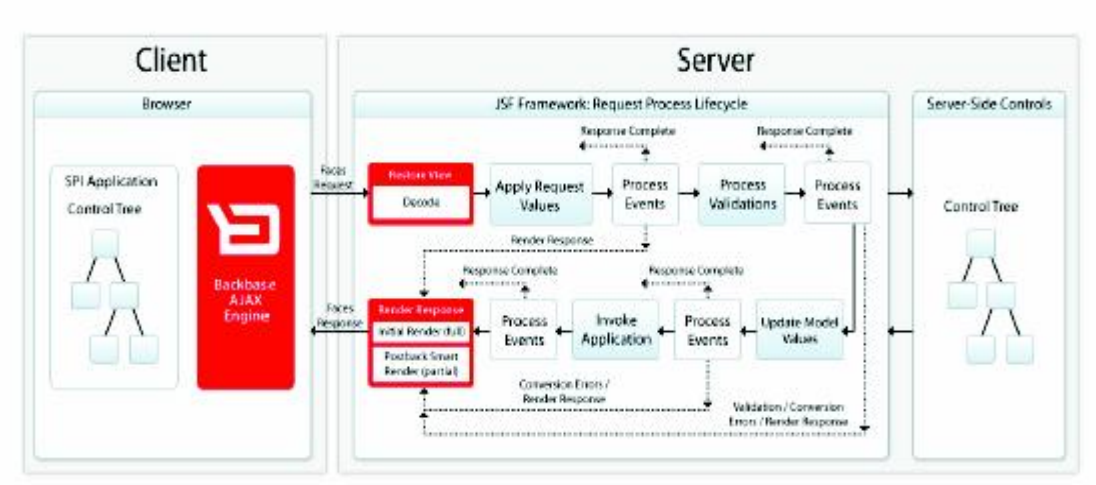

Ilustração 47 - Etapas realizadas por uma requisição Backbase. Fonte : [BACKBASE,2003]

Essa forma de integração é semelhante à apresentada na seção "**Erro! Fonte de referência não encontrada.**5: Implementando um módulo de controle após a terceira fase do ciclo de vida", porque utiliza apenas um *Servlet*, conhecido como *FacesServlet*, possui um módulo de controle após a terceira fase do ciclo, e reimplementa a última fase, conhecida como "Restaurar Visão".

6.2.4.4 Considerações Finais *Backbase*

Uma das deficiências observadas nesta biblioteca é a inexistência de um componente que permita facilidades para sugerir palavras a partir de partículas inseridas pelo usuário, funcionalidade esta semelhante às apresentadas nas seções: "6.2.1.2 Aplicação Exemplo no item *AutoComplete*" e "6.2.3.2 Aplicação Exemplo no componente *Suggestion Box*".

6.2.5 Java BluePrints

6.2.5.1 Introdução

Java BluePrints [SUN DEVELOPER NETWORK, 2005] é um projeto que reúne várias empresas, coordenado pela Sun Microsystems e que tem como principal objetivo o de disponibilizar componentes e informações das melhores formas de integração da tecnologia AJAX com o *framework* JSF.

A seguir será apresentado um exemplo que faz uso dos principais componentes já disponíveis, resultado do referido projeto.

6.2.5.2 Aplicação Exemplo

O exemplo desenvolvido, similar aos anteriormente apresentados, utiliza uma *IDE* desenvolvida pela SUN, conhecida como *Sun Java Studio Creator*, a qual auxilia os desenvolvedores a construírem suas aplicações fazendo uso dos componentes *Blueprints.*

Na seqüência é apresentada a página do exemplo, cujo código fonte pode ser visualizado no capítulo “Anexo”, na seção “10.6 Exemplo *BluePrints*”.

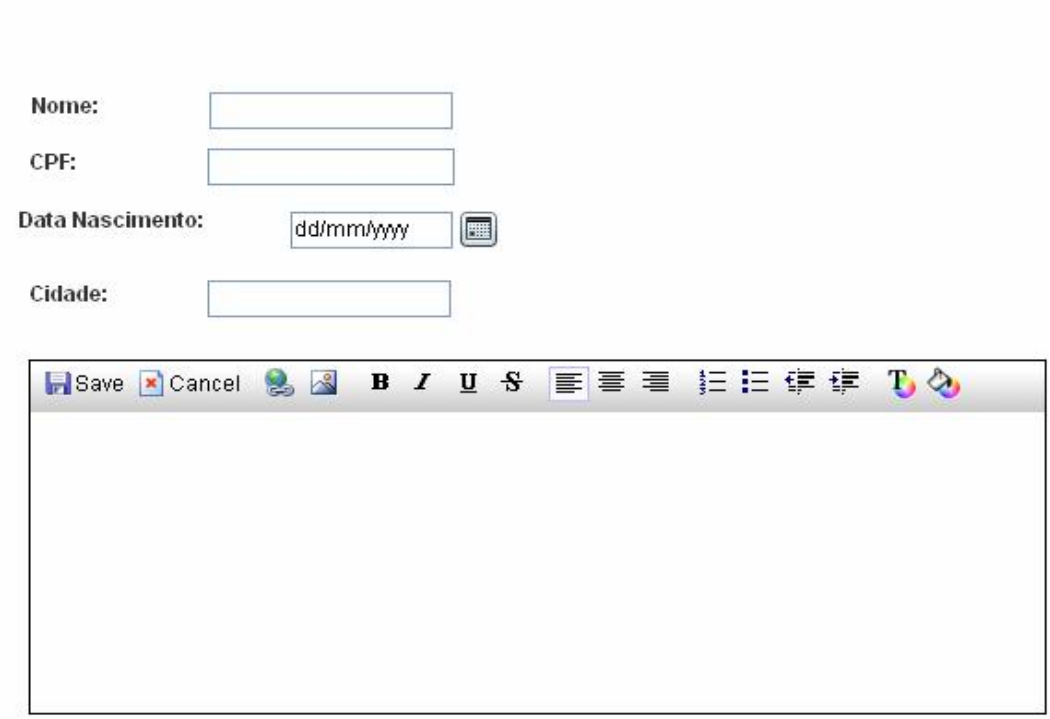

Ilustração 48 - Tela do exemplo utilizando *Blueprints*

Os componentes usados para construção do exemplo foram:

- *Autocomplete Text field*:

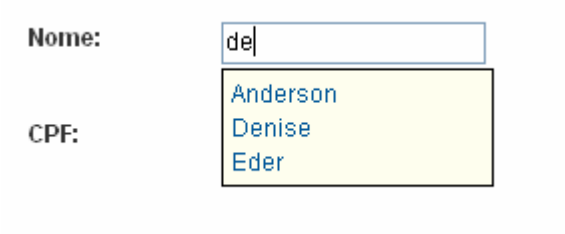

Ilustração 49 - Exemplo *Blueprints* componente *Autocomplete*

Este componente foi utilizado no campo **Nome** para que a cada letra digitada o sistema procure o conteúdo numa determinada lista e sugira os nomes que contenham a partícula inserida pelo usuário.

- *Rich Textarea Editor*:

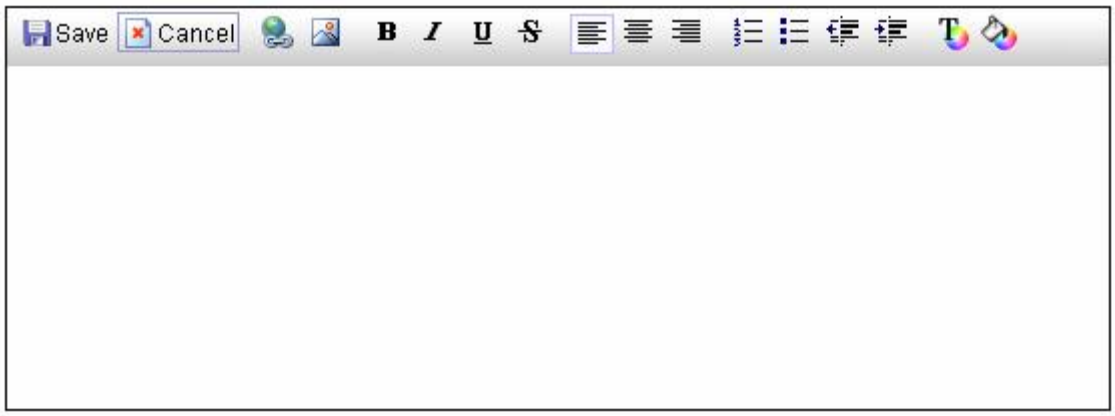

Ilustração 50 - Exemplo *Blueprints* componente *Rich Textarea Editor*

O objetivo deste componente, entre outros, é implementar funcionalidades de um editor de texto, com recursos de centralizar o texto e colocar o negrito em algumas palavras.

O *Rich Textarea Editor* ainda não utiliza AJAX, porém o projeto pretende incorporar essa tecnologia ao componente permitindo, entre outras funcionalidades, verificar a ortografia das palavras no momento que estas são inseridas, alertando o usuário sobre possíveis erros.

ü *Popup Calendar*:

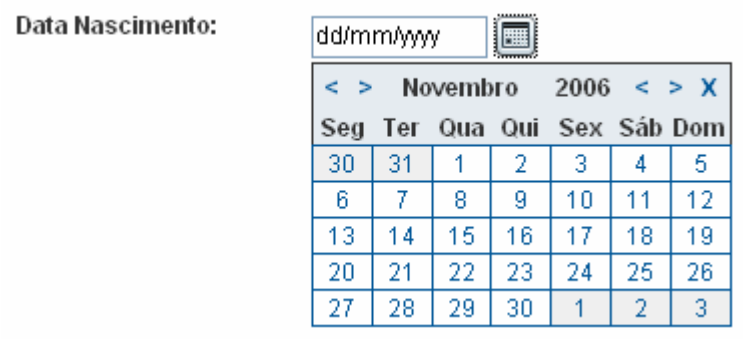

Ilustração 51 - Exemplo *Blueprints* componente *Calendar*

De forma similar ao apresentado por outras bibliotecas, este é um componente utilizado para facilitar ao usuário a seleção de datas.

Destaca-se, porém, que para implementar tal funcionalidade não é necessário utilizar a tecnologia AJAX pois, por intermédio de funções *JavaScript* obtêm-se o mesmo recurso sem precisar que nenhuma requisição seja realizada ao servidor.

ü *Select Value Text Field*:

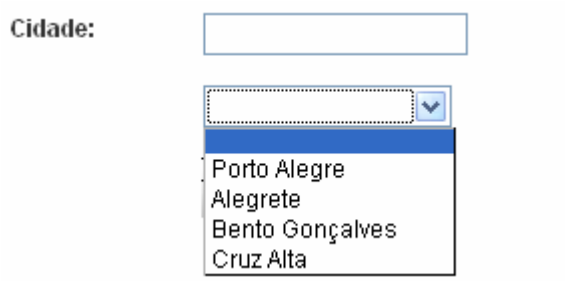

Ilustração 52 - Exemplo *Blueprints* componente *Select Value Text Field*

Este componente é similar ao anterior apresentado(*Autocomplete Text field*). O que o difere é o fato de que, ao digitar algumas partículas de nomes no campo **Cidade**, uma lista de nomes sugeridos é apresentada em outro campo da página (em vez do *combo* ser aberto no mesmo campo de edição como ocorre no componente usado para o campo **Nome**), facilitando ao usuário a escolha do conteúdo desejado.

6.2.5.3 Forma de Integração *Blueprints*

Diferentemente de algumas outras bibliotecas, em que para integrar a tecnologia AJAX e JSF se faz uso de dois *Servlets*, separando as requisições AJAX das demais, na biblioteca *Blueprints* é utilizado apenas o *Servlet* disponibilizado pelo JSF, conhecido como *FacesServlet*.

Para implementar a integração AJAX e JSF, essa biblioteca utilizou-se de um *PhaseListener* conhecido como *RemotingPhaseListener*, que atua logo após a primeira fase do ciclo de vida das requisições JSF. Depois da execução desse *PhaseListener*, um arquivo XML contendo os dados requeridos é enviado para a página para que esta atualize apenas as informações alteradas pelas funções *JavaScript* e nenhuma outra fase do ciclo seja executada.

A forma de integração AJAX e JSF pode ser representada pela figura a seguir:

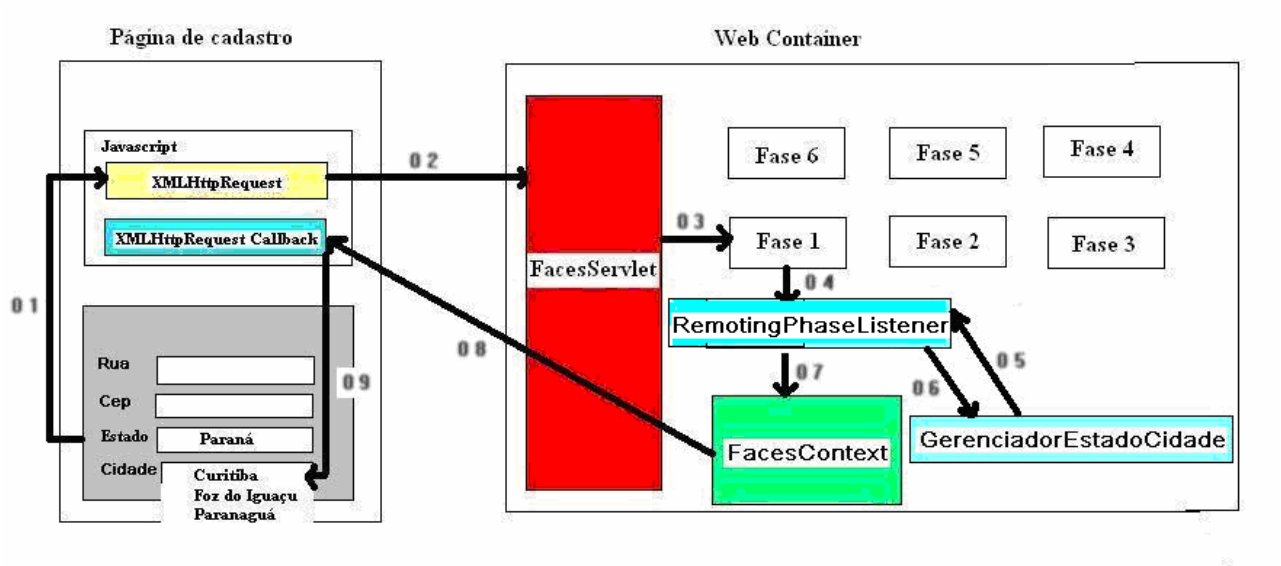

Ilustração 53 - Etapas realizadas por uma requisição utilizando a biblioteca *BluePrints*

Observando-se a "Ilustração 53 - Etapas realizadas por uma requisição utilizando a biblioteca *BluePrints*", percebe-se a similaridade desta idéia de integração AJAX e JSF com a apresentada na seção "**Erro! Fonte de referência não encontrada.**: Utilizando um *PhaseListener* após a primeira fase do ciclo de vida".

6.2.5.4 Considerações Finais *Blueprints*

Um destaque desta biblioteca é o componente ***Rich Textarea Editor***, cujas funcionalidades não foram encontradas em nenhum outro projeto pesquisado. No entanto, observa-se ainda a falta de recursos de correção ortográfica, que pode ser implementado fazendo uso da tecnologia AJAX.

Como o referido projeto encontra-se em desenvolvimento e pelo potencial das empresas nele envolvidas, espera-se que a biblioteca ***Blueprints*** receba, em pouco tempo, incremento em componentes e novas funcionalidades.

6.2.6 DWR

6.2.6.1 Introdução

O projeto DWR [DWR, 2005] possui o seguinte slogan: AJAX fácil para JAVA (Easy AJAX for JAVA).

Para a utilização desta biblioteca, o desenvolvedor deve mapear em seu aplicativo um arquivo XML, contendo todas as classes juntamente com seus métodos executados de forma assíncrona. Com recursos da biblioteca são construídas as funções *JavaScript* necessárias para que as ações sejam executadas de maneira correta.

Utilizando-se desta ferramenta não é necessária a construção de componentes personalizados, porém exige-se do desenvolvedor conhecimento em *JavaScript* para criar funções para invocar as funcionalidades criadas pela biblioteca e também para promover interações com o usuário, como por exemplo, exibir uma mensagem alertando sobre a existência de algum campo inválido.

O DWR pode ser integrado com diversos *frameworks*, por exemplo: *Spring, Struts*, JSF, entre outras.

6.2.6.2 Aplicação Exemplo

No exemplo construído utilizando-se esta biblioteca, a funcionalidade de verificação do conteúdo digitado pelo usuário no campo **E-mail** utiliza-se da tecnologia AJAX. Quando o usuário tirar o foco deste campo, se o conteúdo inserido não contiver o caractere “@” o sistema exibe uma mensagem de erro, conforme mostra a figura a seguir.

Ilustração 54 - Exemplo DWR verificando campo E-mail

O código fonte do exemplo desenvolvido com esta biblioteca pode ser visualizado no capítulo “Anexo”, na seção “10.7 Exemplo DWR”.

6.2.6.3 Forma de Integração DWR

Para promover a integração do AJAX e JSF, o DWR utiliza-se de dois *Servlets*: *FacesServlet*, disponibilizado pelo *JavaServer Faces* e *DWRServlet* do próprio DWR, além das funções *JavaScript*, criadas pela própria biblioteca, para que as requisições assíncronas possam ser invocadas.

A figura a seguir demonstra o caminho seguido por uma requisição DWR desde a ação do usuário até a atualização da página.

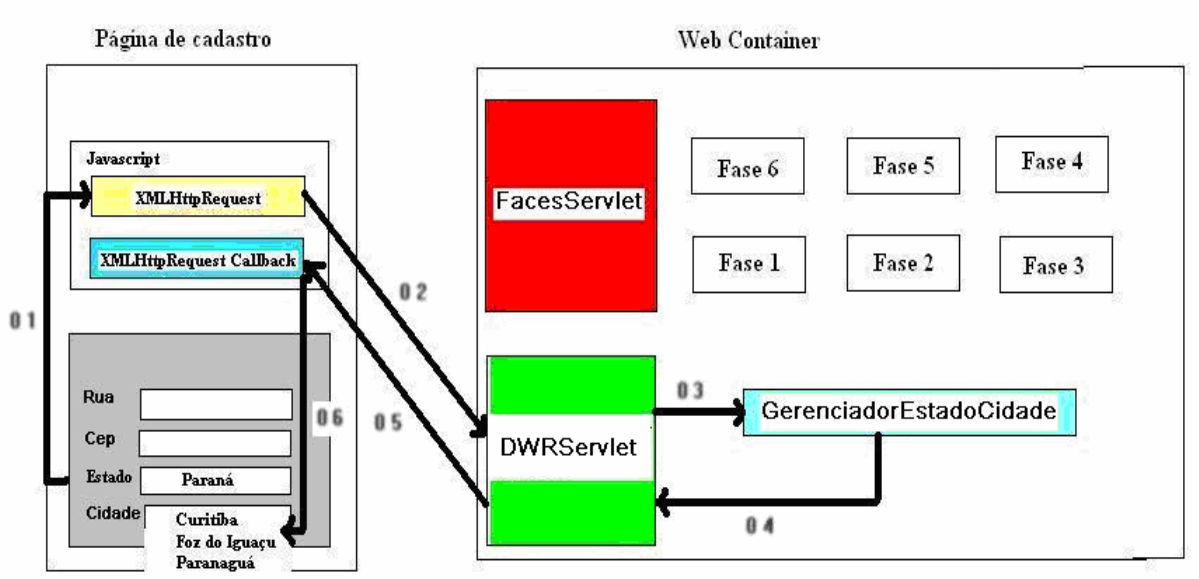

Ilustração 55 – Passos seguidos por uma requisição que utiliza a biblioteca DWR [DWR, 2005]

Concluindo, observa-se que a forma de integração do AJAX e JSF usando o DWR é similar à apresentada na seção "**Erro! Fonte de referência não encontrada.** Integrando os componentes JSF com AJAX por intermédio de funções *JavaScript*", pois em ambas não são desenvolvidos componentes personalizados (a própria biblioteca constrói funções *JavaScript* automaticamente) e são utilizados dois *Servlets*.

6.2.6.4 Considerações Finais DWR

Uma das vantagens na utilização do DWR é a sua distribuição gratuita. Mesmo assim, essa biblioteca já se encontra na versão 2.0.

Entretanto, para a utilização do DWR não são desenvolvidos componentes para auxiliar na realização de requisições assíncronas, pois estas são controladas por meio da criação de funções *JavaScript*, o que exige dos desenvolvedores conhecimento desta linguagem.

6.2.7 Mabon

6.2.7.1 Introdução

Este é um projeto *open-source* chamado de *Mabon* [MABON, 2006] liderado por John Fallows, e que está baseado nos princípios apresentados em seu próprio livro "Pro AJAX and JSF" [JACOBI&FALLOWS, 2006a], relacionados às formas de integração entre AJAX e *JavaServer Faces*.

6.2.7.2 Aplicação Exemplo

Não foi possível implementar um exemplo com esta biblioteca, pois o único componente disponibilizado até o momento, denominado *inputSuggest*, semelhante ao Google Suggest [GOOGLE, 2006], apresenta problemas em sua execução, cuja funcionalidade proposta, conforme figura a seguir, é a de auxiliar na escolha de conteúdo em preenchimento de campos.

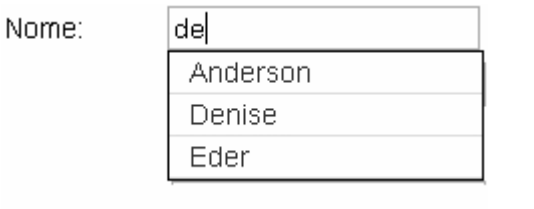

Ilustração 56 - Projeto *Mabon* componente *inputSuggest*

O exemplo pretendido tinha o seguinte objetivo: a cada letra digitada, o sistema deveria sugerir todos os nomes cadastrados contendo a partícula inserida pelo usuário.

6.2.7.3 Forma de Integração *Mabon*

As principais classes implementadas por esta biblioteca para que os componentes JSF utilizem a tecnologia AJAX de maneira correta são:

- ü *MabonLifecycle*: esta é a classe que contém todas as fases do ciclo de vida das requisições JSF que utilizam a tecnologia AJAX, as quais serão a seguir comentadas:
- *ApplyRequestValues Phase*: encontrar a classe responsável por executar o método da requisição.
- *InvokeApplication Phase*: executar o método da requisição.
- *RendererResponsePhase*: coletar informações do método executado na fase anterior e terminar o ciclo de vida das requisições JSF que utilizam tecnologia AJAX são algumas das responsabilidades desta fase.

Todos os passos de uma requisição desta biblioteca podem ser observados na figura abaixo:

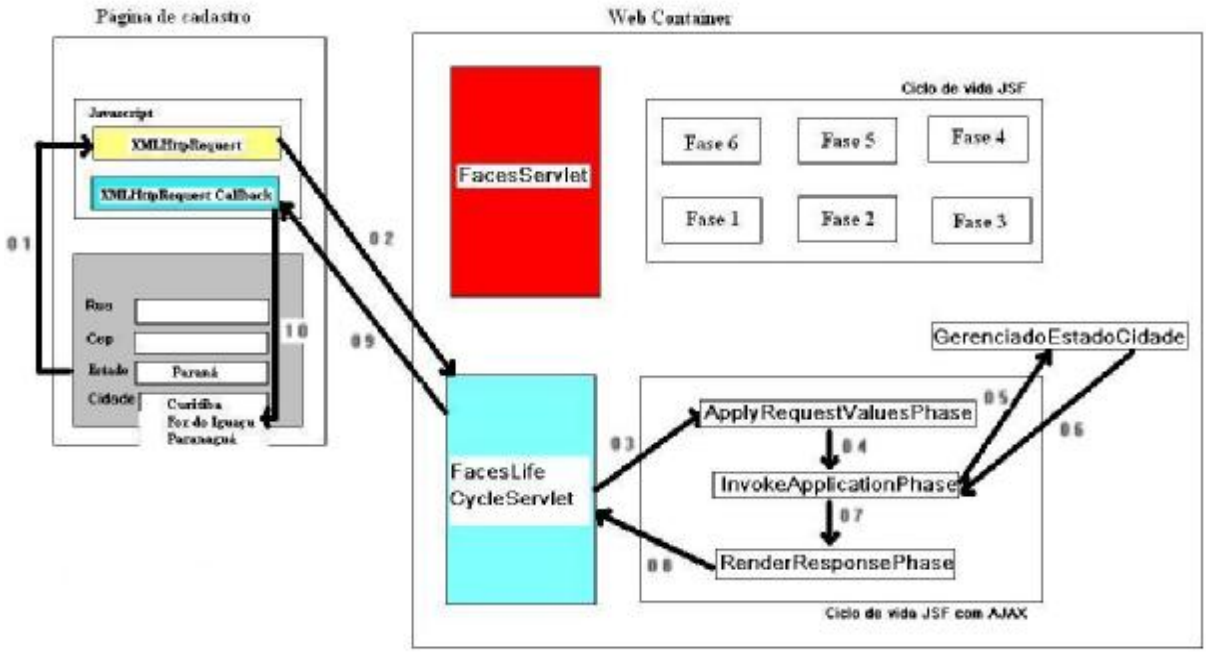

Ilustração 57 - Etapas realizadas por uma requisição da biblioteca Mabon. Fonte: [MABON,2006]

Podemos observar que esta forma de integração é semelhante à apresentada na seção "**Erro! Fonte de referência não encontrada.** Criando um *Servlet* para gerenciar as requisições AJAX e implementando um novo ciclo de vida para requisições AJAX" e, portanto, possui as mesmas características apresentadas no referido tópico.

6.2.7.4 Considerações Finais *Mabon*

Aparentemente este projeto não está mais em desenvolvimento, visto que a biblioteca *Mabon* está na versão 0.1 desde fevereiro de 2006 e apenas o componente *inputSuggest* foi concluído e, mesmo assim, o exemplo disponibilizado pelo fabricante não funciona. A fraca participação de interessados na lista de discussão dificultou a obtenção de mais informações sobre o andamento do projeto.

6.2.8 DynaFaces

6.2.8.1 Introdução

O projeto *DynaFaces* [SUN, 2006] teve sua idéia publicada em setembro de 2005, no *blog* de Jacob Hookom. Obteve muitas contribuições de alguns desenvolvedores especialistas no assunto de integração AJAX e JSF, como Ed Burns e Adam Winer, e conta atualmente com o apoio da Sun Microsystem.

No JavaOne 2006 foi apresentada a versão 0.1, com o intuito de divulgar a filosofia de integração utilizada, e obteve boa aceitação por parte dos desenvolvedores, sendo atualmente a forma mais provável de ser utilizada como padrão para a especificação JSF 2.0 a fim de contemplar os recursos do AJAX.

6.2.8.2 Aplicação Exemplo

Como este projeto é incipiente e seus componentes ainda não foram liberados para utilização, não foi possível implementar um exemplo utilizando o *DynaFaces*. Dessa forma, será comentado a seguir alguns dos componentes apresentados durante o JavaOne.

- *ServerSuggest*:

As funcionalidades do *ServerSugget*, que podem ser visualizadas na figura abaixo, permitem que o sistema sugira vários nomes cadastrados que contenham a partícula inserida pelo usuário.

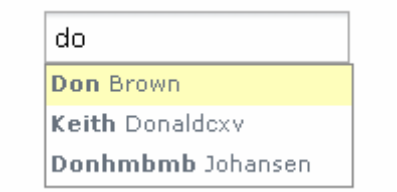

Ilustração 58 – Componente *ServerSuggest* de *DynaFaces* da apresentação JavaOne[JAVAONE, 2006]

ü *DataTable*:

Componente que permite que os registros contidos numa lista sejam exibidos em uma tabela e separados por páginas.

Como no exemplo da "Ilustração 59 – Componente DataTable de DynaFaces da apresentação JavaOne [JAVAONE,2006]", utilizando a tecnologia AJAX o usuário pode navegar entre as páginas da tabela e apenas o conteúdo alterado será atualizado, por meio de uma requisição assíncrona, sem a necessidade de renderizar novamente toda a tela do aplicativo.

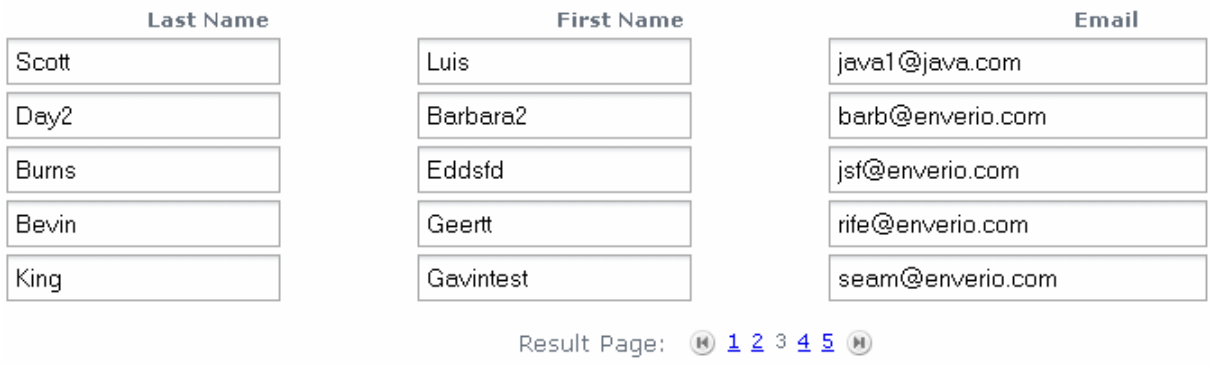

Ilustração 59 – Componente DataTable de DynaFaces da apresentação JavaOne [JAVAONE,2006]

6.2.8.3 Forma de Integração *Dynafaces*

A idéia utilizada por este projeto é similar à apresentada na seção "**Erro! Fonte de referência não encontrada.**2 Modificando a especificação JSF" e pretende modificar a especificação JSF. Com isso, todos os componentes herdariam a capacidade de utilizar a tecnologia AJAX sem nenhum esforço adicional do desenvolvedor.

6.2.8.4 Considerações Finais *Dynafaces*

Embora ainda não possua um conjunto de componentes disponíveis com a filosofia proposta por este projeto, esta idéia, como vimos anteriormente, é a mais cotada para ser aderida na especificação JSF2.0, que pretende promover a integração deste *framework* com a tecnologia AJAX.

6.3 Estudo Comparativo

Nesta seção será apresentado um comparativo entre as bibliotecas estudadas, considerando alguns quesitos, como por exemplo: custo, facilidade de uso, eficiência, documentação e forma de integração de JSF com AJAX. Os conceitos foram atribuídos pelo autor deste trabalho, baseados na análise de documentação disponível bem como no desenvolvimento dos exemplos.

Tabela 2 - Comparação das bibliotecas estudadas

	ICEfaces	Ajaxfaces	Ajax4jsf
Custo	Open-source	Pago	Código livre
Facilidade de aprendizado	Ótimo	Regular	Bom
Facilidade de uso	Ótimo	Ruim	Bom
Documentação	Ótimo	Ruim	Bom
Exemplos disponibilizados	Ótimo	Ruim	Bom
Quantidade de componentes	Bom	Ruim	Bom
Qualidade dos componentes	Ótimo	Ruim	Bom
Auxílio ao desenvolvedor	Ótimo	Ruim	Bom
Plugins para IDE	Bom	Ruim	Ótimo
Forma de integração	5.2.2.2.1 Criando um Servlet para gerenciar requisições AJAX	5.2.2.2.1 Criando um Servlet para gerenciar requisições AJAX	5.2.2.1.6: Utilizando PhaseListener em três fases do ciclo de vida

	Backbase	Blueprints	DWR
Custo	Pago	Grátis	Grátis
Facilidade de aprendizado	Bom	Ótimo	Bom
Facilidade de uso	Bom	Bom	Bom
Documentação	Ótimo	Bom	Regular
Exemplos disponibilizados	Bom	Ótimo	Regular
Quantidade de componentes	Ótimo	Bom	*
Qualidade dos componentes	Ótimo	Bom	*
Auxílio ao desenvolvedor	Ótimo	Bom	Bom
Plugins para IDE	Ótimo	Ótimo	Ruim
Forma de integração	5.2.2.1.5: Implementando um módulo de controle após a terceira fase do ciclo de vida	5.2.2.1.3: Utilizando um PhaseListener após a primeira fase do ciclo de vida	5.2.1.1 Integrando os componentes JSF com AJAX através de funções JavaScript

	Mabon	DynaFaces
Custo	Open Source	Open Source
Facilidade de aprendizado	**	**
Facilidade de uso	**	**
Documentação	**	**
Exemplos disponibilizados	**	**
Quantidade de componentes	**	**
Qualidade dos componentes	**	**
Auxílio ao desenvolvedor	**	**
Plugins para IDE	**	**
Forma de integração	5.2.2.2.2 Criando um Serlvet para gerenciar as requisições AJAX e implementando um novo ciclo de vida para requisições AJAX	5.2.1.2 Modificando a especificação JSF

* - O projeto DWR não desenvolve componentes.

** - Não foram realizadas avaliações de alguns quesitos dos projetos *Mabon* e *DynaFaces* pois no momento do desenvolvimento deste trabalho ainda se encontravam em versão inicial.

Para uma melhor compreensão foi atribuído um intervalo de notas para cada conceito, conforme a tabela abaixo:

Tabela 3 – Notas associadas aos conceitos dados nos quesitos das bibliotecas

Conceito	Nota
Ótimo	8 - 10
Bom	6 - 8
Regular	4 - 6
Ruim	0 - 4

6.4 Conclusão

Constata-se que existem inúmeras bibliotecas de componentes JSF que usufruem da tecnologia AJAX utilizando para isso as mais diversas formas de integração, cada qual com suas vantagens e desvantagens, faltando ainda um consenso da melhor forma.

Outra constatação é a de que boas bibliotecas estudadas têm seu custo de licenciamento alto, incompatível com a realidade de pequenas empresas e desenvolvedores autônomos.

Contudo, os problemas citados acima devem desaparecer na versão do *framework* JSF 2.0, pois este tem como propósito padronizar e incorporar ao *JavaServer Faces* à tecnologia AJAX. Desse modo, os novos componentes JSF poderão ter os recursos AJAX de forma nativa. Adicionalmente, como o JSF 2.0 será distribuído sem custo de licenciamento, estará disponível a todos os desenvolvedores, alternativa frente ao alto custo cobrado por boas bibliotecas.

Considerando esses fatores é possível prever que as bibliotecas estudadas neste trabalho, para sobreviverem, devem se tornar *open source* (fato já ocorrido na biblioteca *ICEfaces*), ou de distribuição gratuita ou ainda oferecerem diferenciais de funcionalidades e de recursos que justifiquem o seu preço.

No entanto, até a especificação JSF 2.0 ser publicada, dentre as bibliotecas pesquisadas neste trabalho, levando-se em conta os quesitos analisados, o projeto *ICEfaces* é o melhor para o aproveitamento dos recursos da integração AJAX e JSF para o desenvolvimento de aplicativos.

Além das vantagens já apresentadas, o projeto *ICEfaces* permite a criação de novos componentes para integrar as referidas tecnologias.

7 COMPONENTE DESENVOLVIDO

7.1 Introdução

Enquanto a especificação do JSF 2.0 não for disponibilizada, com base nos estudos realizados, considera-se que a melhor forma de integração do JSF e AJAX é a que utiliza um *Servlet* apenas para gerenciar as requisições assíncronas, como visto na seção "**Erro! Fonte de referência não encontrada.**.1 *Criando um Servlet para gerenciar requisições AJAX*", devido principalmente a facilidade de implementação e eficiência. Fazendo uso dessa idéia foi desenvolvido um componente com a principal funcionalidade de permitir a validação de campos de entrada de conteúdo dos aplicativos, interagindo com o usuário tão logo o foco seja retirado do referido campo, sem a necessidade de renderizar toda a página.

7.2 Componente desenvolvido

Foi criado um componente chamado de *validatorAjax* que se utiliza da tecnologia AJAX e permite realizar a validação de qualquer campo de um sistema, necessitando apenas que o desenvolvedor implemente o método *valide(String valor)* da classe abstrata *Validate*.

Para a criação deste componente foram desenvolvidas cinco classes:

- *AjaxValidatorRenderer*:

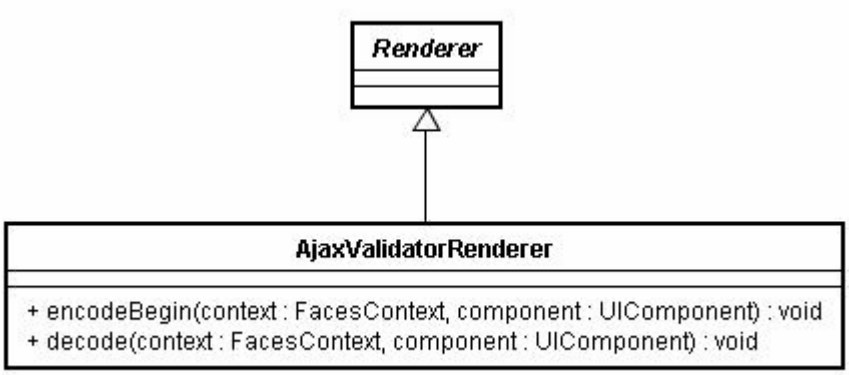

Ilustração 60 - Diagrama de classe com *AjaxValidatorRenderer*

Esta é uma subclasse da classe *Renderer* do JSF e tem como principal função definir os atributos dos campos que serão exportados para o código fonte da página.

Neste componente foram definidos os seguintes atributos: *value, size, idValidator, id, onmouseover* e *onblur* , que definem respectivamente o atributo da classe que receberá o conteúdo do *value*, o tamanho do campo, a classe que fará a validação, o identificador e os efeitos que serão ativados quando o mouse passa sobre e quando o foco é retirado do componente.

- *UIValidatorAjax*:

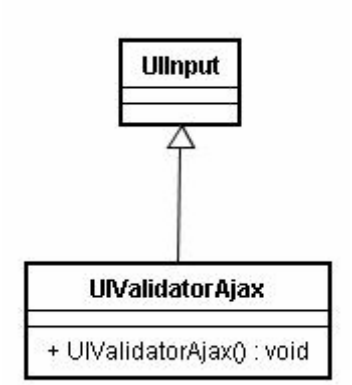

Ilustração 61 - Diagrama de classe com *UIValidatorAjax*

Esta classe é uma extensão de *UIInput*, a qual define que o renderizador deste componente será o *AjaxValidatorRenderer*.

ü *ValidatorAjaxTag*:

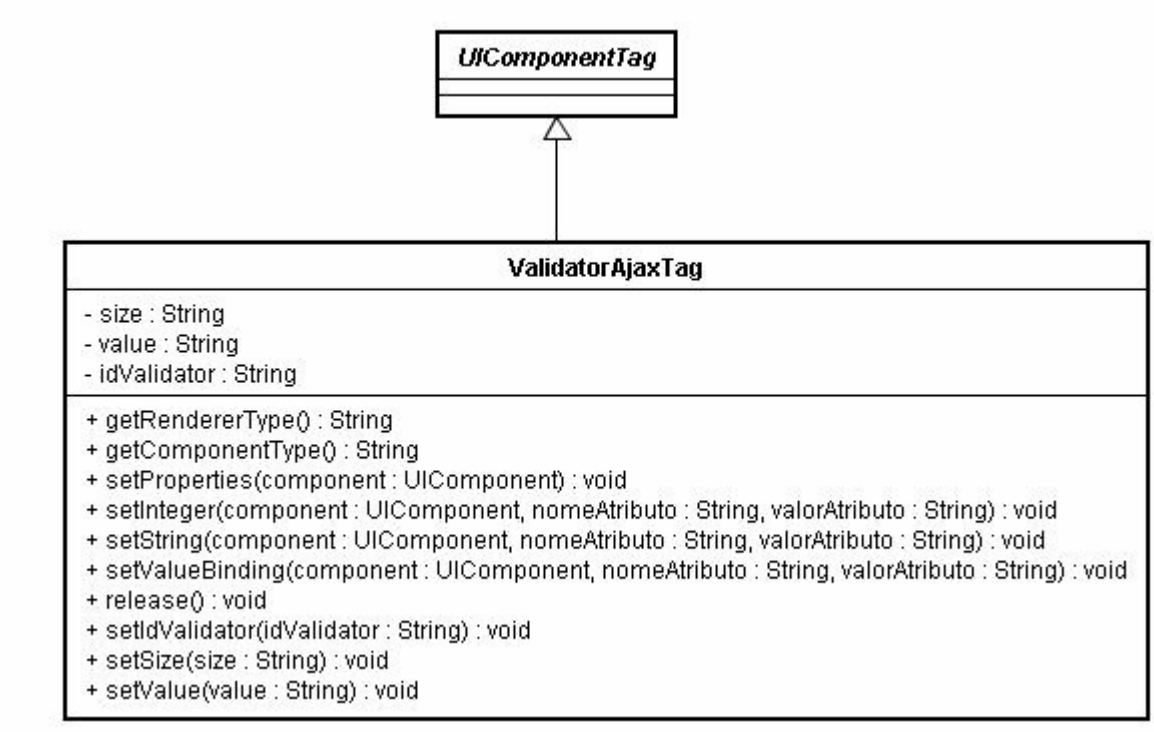

Ilustração 62 - Diagrama de classe com *ValidatorAjaxTag*

ValidatorAjaxTag é uma classe que estende a *UIComponentTag* e possui como principal objetivo definir os atributos da *tag* que poderão ser utilizados pelo desenvolvedor.

ü *GerenciadorServlet*:

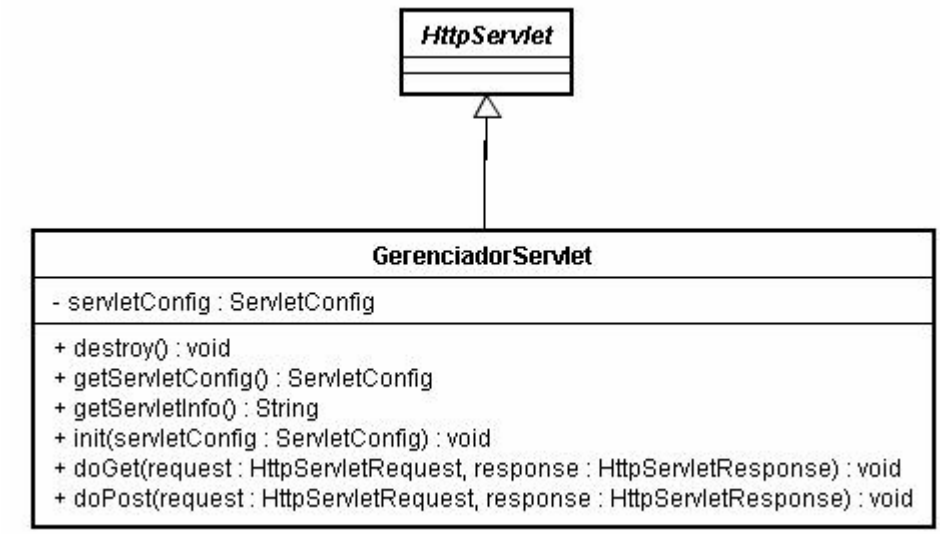

Ilustração 63 - Diagrama de classe com classe GerenciadorServlet

Este é um *Servlet* invocado pelo *validatorAjax* toda vez que o usuário tira o foco deste componente.

No *GerenciadorServlet* é instanciado um objeto da classe definida no atributo *idValidator* da *tag* para que esta instância valide o conteúdo inserido pelo usuário no *validatorAjax*. Depois da validação, um arquivo *xml* é devolvido para a página, sinalizando se o conteúdo digitado pelo cliente é valido ou não.

- *Validate*:

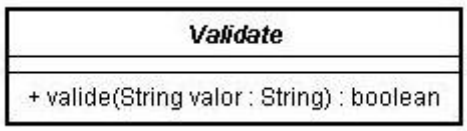

Ilustração 64 - Diagrama de classe com classe abstrata *Validate*

Esta é uma classe abstrata que deve ser estendida para cada tipo de campo que o desenvolvedor deseje validar em qualquer parte do sistema. Ao utilizar o componente *validatorAjax* será necessário apenas que o desenvolvedor sobrescreva o método *valide(String valor)*.

Como exemplos de tipos de campos podem ser citados: **CPF**, **E-mail**, **CEP** etc.

O componente *validatorAjax* faz uso de dois arquivos, a seguir descritos:

- *ajax.js*:

Neste arquivo estão definidas todas as funções J*avaScript* necessárias para funcionamento do componente.

- *validatorAjax.tld*:

O arquivo *validatorAjax.tld* juntamente com a classe *ValidatorAjaxTag* definem os atributos que o desenvolvedor poderá utilizar neste componente.

7.3 Exemplo utilizando o componente desenvolvido

A figura a seguir demonstra o exemplo desenvolvido utilizando-se do componente *validatorAjax*.

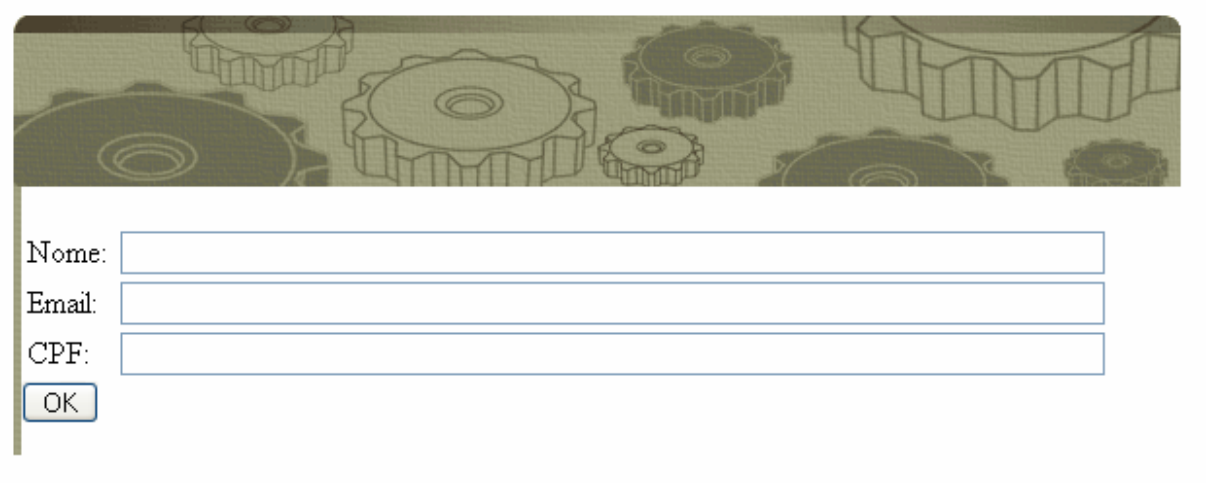

Ilustração 65 - Exemplo utilizando o componente validatorAjax

Neste exemplo foram criadas duas classes, *EmailValidator* e *CPFValidator*, que estendem a classe abstrata *Validate*, para validarem o conteúdo dos campos **E-mail** e **CPF** respectivamente.

O seguinte requisito foi definido no componente *validatorAjax*: toda vez que o usuário digitar um valor inválido, o conteúdo do campo torna-se vermelho tão logo o foco saia do referido campo. Isso pode ser observado no exemplo implementado a seguir, nos campos **E-mail** e **CPF**.

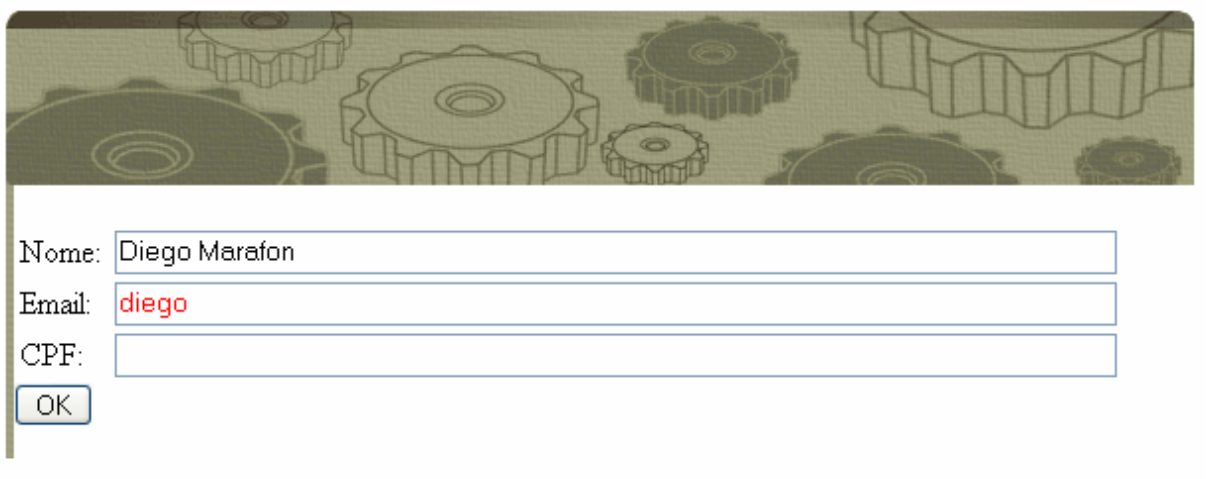

Ilustração 66 - Exemplo utilizando-se o componente *validatorAjax* validando o campo E-mail

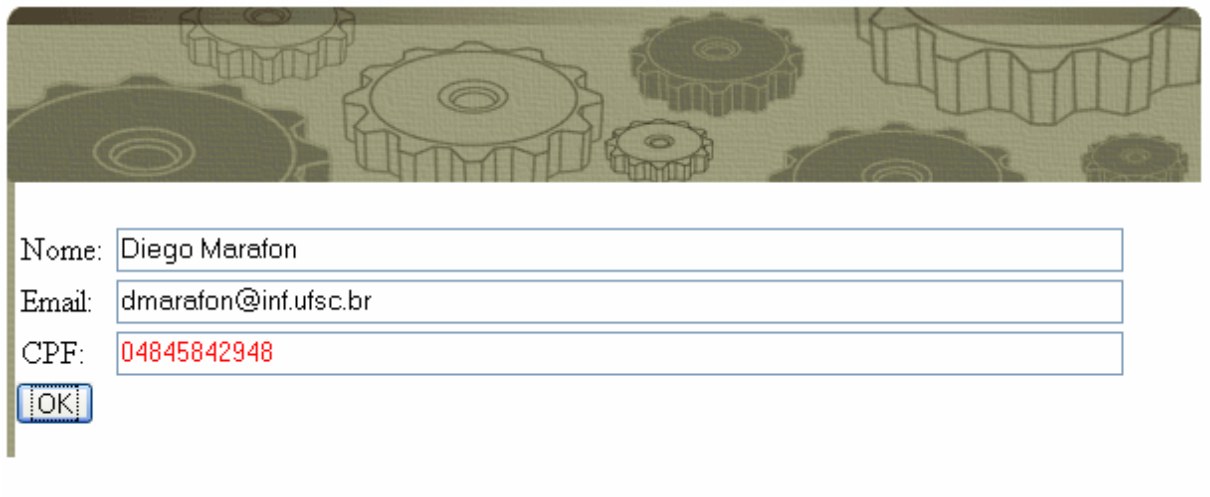

Ilustração 67 - Exemplo utilizando-se o componente *validatorAjax* validando o campo CPF

Todo o código fonte do exemplo apresentado pode ser visualizado no capítulo "Anexo", na seção "10.8 Exemplo *ValidatorAjax*".

7.4 Conclusão

A implementação do componente *validatorAjax* vem confirmar um dos objetivos do *framework* JSF que é o de tornar a criação de componentes personalizados uma tarefa viável de ser executada.

Também foi possível demonstrar a viabilidade da criação de componentes, os quais podem oferecer facilidades e produtividade aos desenvolvedores, aproveitando simultaneamente as potencialidades JSF e da tecnologia AJAX, explorando principalmente a interatividade dos aplicativos, tornando-os mais ricos em funcionalidades e confortáveis para os usuários.

8 CONCLUSÃO FINAL

Um assunto que foi objeto de estudo neste trabalho foi o *framework JavaServer Faces*, o qual atua principalmente nas camadas de Visão e Controle do modelo MVC (Modelo – Visão – Controle).

Esse *framework* surgiu nesta década, tendo como um de seus diferenciais o de possuir um modelo de programação dirigida a eventos, tentando assim tornar o desenvolvimento Web mais parecido com o *desktop*. No entanto, quando comparado com os seus concorrentes, como é o caso do *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.

Tal limitação faz com que as páginas dos aplicativos sejam totalmente reconstruídas ao sofrerem algum evento, mesmo que apenas partes dos seus dados tenham sido modificados, causando desconforto para o usuário.

Recentemente, com o surgimento do AJAX, que promete oferecer a interatividade esperada nos aplicativos Web, o foco dos estudos está dirigido para a integração do *framework* JSF com esta tecnologia, buscando fazer uso dos benefícios de cada um.

Muitos esforços foram movidos nesta direção, criando-se idéias e bibliotecas de componentes que buscam esta integração, tema deste trabalho.

Considerando-se os prós e contras de todas as formas de integração estudadas, a melhor idéia considerada é a apresentada na seção "**Erro! Fonte de referência não encontrada.** Modificando a especificação JSF" e que fez muito sucesso no último JavaOne.

A grande vantagem obtida com a utilização dessa idéia é o fato de que os componentes JSF poderão realizar requisições assíncronas, fazendo uso da tecnologia AJAX, sem nenhum esforço adicional dos desenvolvedores.

A importância dada a este assunto motivou os responsáveis pela especificação do JSF para que a versão 2.0, a qual deve ser lançada em 2008, contemple a integração deste *framework* com a tecnologia AJAX. O pensamento mais cotado a ser adotado para implementar tal integração é o apresentado na seção "**Erro! Fonte de referência não encontrada.**2 Modificando a especificação JSF" [BREAU, 2006].

Caso tais requisitos forem implementados no JSF 2.0, a construção de bibliotecas de componentes *JavaServer Faces* que disponibilizam as funcionalidades do AJAX se tornará muito mais fácil. Com isso, muitas das atuais bibliotecas existentes deverão seguir o exemplo da *ICEfaces*, que se tornou *open source*, e muitos outros projetos de componentes acessíveis em termos de custos deverão surgir, revolucionando o desenvolvimento de aplicativos para Web.

Enquanto isso não se tornar realidade, segundo este estudo desenvolvido, a melhor forma de integração existente até o momento é a comentada na seção "**Erro! Fonte de referência não encontrada.**.1 Criando um *Servlet* para gerenciar requisições AJAX", e que foi utilizada no "Capítulo 7. Componente desenvolvido – *validatorAjax"*.

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, bem como desenvolver componente que faça esta integração da melhor forma estudada, foram alcançados. Entretanto, apesar das evidências apresentadas ao longo do trabalho que indicam tendências de padrões futuros, constata-se também que não existe um consenso da melhor forma de junção do *framework* e da tecnologia em evidência. Analisando a quantidade de projetos que estão surgindo com objetivo de promover tal integração, a única certeza que resta é a de que muitas novidades surgirão nos próximos anos.

8.1 Trabalhos futuros

Considerando que os responsáveis pela especificação JSF têm como objetivo lançar o seu *framework* integrado com AJAX na versão 2.0, sugere-se como possíveis trabalhos futuros relacionados a este assunto:

Construção de novos componentes. O componente *validatorAjax* foi desenvolvido neste trabalho com o objetivo de demonstrar na prática a melhor forma atual de construção de componentes JSF que utilizam a tecnologia AJAX. Dessa forma, um dos trabalhos futuros pode ser o desenvolvimento de outros componentes que fazem uso da mesma filosofia adotada no *validatorAjax*.

Analisar a forma de integração do JSF 2.0 com AJAX. Tão logo seja disponibilizada uma versão preliminar do JSF 2.0, analisar a forma de integração adotada com AJAX, comparando-a com os métodos de integração apresentados neste trabalho.

9 REFERÊNCIAS BIBLIOGRÁFICAS

[ADAPTATIVE PATH, 2006] **Adaptative Path**. Ajax: A New Approach To Web Applications. Disponível em: <http://adaptivepath.com/publications/essays/archives/000385.php> . Acesso em: 10 jun. 2006.

[APACHE, 2006] APACHE. **ADF Faces – Apache Incubator**. Disponível em: <http://incubator.apache.org/projects/adffaces.html>. Acesso em: 26 set. 2006.

[BACKBASE, 2003] BACKBASE. **The #1 AJAX Development Framework**. Disponível em: <http://www.backbase.com>. Acesso em: 17 jul. 2006.

[BASLER, 2006a] BASLER, Mark. **Using PhaseListener Approach for JavaServer Faces Technology with AJAX**. Disponível em: <https://blueprints.dev.java.net/bpcatalog/ee5/ajax/phaselistener.html> .Acesso em: 05 out. 2006.

[BASLER, 2006b] BASLER, Mark. **Using a Servlet with JavaServer Faces Technology and AJAX**. Disponível em: <https://blueprints.dev.java.net/bpcatalog/ee5/ajax/servletControllerwithJSF.html>. Acesso em: 05 out. 2006.

[BREAU, 2006] BREAU, Philip. **JSF1.2 and JSF2.0** [mensagem pessoal]. Mensagem recebida por < dmarafon@gmail.com > em 22 nov. 2006.

[BURNS&HOOKOM&WINER, 2006] BURNS, Ed; HOOKOM, Jacob; WINER, Adam. **Evolving JavaServer Faces Technology: AJAX Done Right**. Disponível em: <http://weblogs.java.net/blog/edburns/20060519-ajax-done-right-00.html>. Acesso em: 11 out. 2006.

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

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

[CYBERXP.NET, 2005] CYBERXP.NET. **AjaxFaces:General and Complete Integration Solution for JSF and AJAX**. Disponível em <http://www.ajaxfaces.com>. Acesso em: 09 jul. 2006.

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

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

[DWR, 2005] DWR. **DWR Easy AJAX for JAVA**. Disponível em: <http://getahead.ltd.uk/dwr>. Acesso em: 21 jul. 2006.

[EXADEL, 2006] EXADEL. **A4J User Guide**. Disponível em: <https://ajax4jsf.dev.java.net/nonav/documentation/ajax-documentation/developerGuide.html>. Acesso em: 08 set. 2006.

[EXADEL, 2006a] EXADEL. **RichFaces**. Disponível em: <http://exadel.com/web/portal/products/VisualComponentPlatform>. Acesso em: 27 out. 2006.

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

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

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

[GOOGLE, 2006] GOOGLE. **Google Suggest**. Disponível em: <http://www.google.com.br/webhp?complete=1&hl=em>. Acesso em: 15 jun. 2006.

[HOOKOM, 2006] 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 nov. 2006.

[ICESOFT, 2006] ICESOFT. **ICEfaces the rich web application**. Disponível em: <http://www.icesoft.com>. Acesso em: 1º jul. 2006.

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

[JACOBI&FALLOWS, 2006b] 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.

[JAVAONE, 2006] JAVAONE. **JavaOne. Suns 2006 Worldwide Java Developer**. Disponível em: <http://sunapp1.whardy.com:8090/jsf-j12/home.jsf>. Acesso em: 1º nov. 2006.

[MABON, 2006] MABON. MAnaged Bean Object Notation. Disponível em: <https://mabon.dev.java.net>. Acesso em: 10 ago. 2006.

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

[MURRAY&BALL, 2006] MURRAY, Gregory; Ball Jeniffer. **Including Ajax Functionality in a Custom JavaServer Faces Component**. Disponível em: <http://java.sun.com/javaee/javaserverfaces/ajax/tutorial.jsp>. Acesso em: 09 out. 2006.

[MURRAY&NORBYE&BURNS, 2005] MURRAY, G.;NORBYE, T.;BURNS, E. **Using JavaServer Faces Technology with AJAX**. Disponível em: <https://bpcatalog.dev.java.net/nonav/ajax/jsf-ajax>. Acesso em: 12 nov. 2005.

[ORACLE, 2004] ORACLE. **ADF Faces Oracle ADF's JSF Components**. Disponível em: <http://www.oracle.com/technology/products/jdev/htdocs/partners/addins/exchange/jsf/index.html>. Acesso em: 26 set. 2006.

[ORT&BASLER, 2006] ORT, Ed; BASLER, Mark. Disponível em: <http://java.sun.com/developer/technicalArticles/J2EE/AJAX/DesignStrategies>. Acesso em: 12 out. 2006.

[SCHALK&BURNS&HOLMES, 2006] SCHALK, Chris; BURNS, Ed; HOLMES, James. **JavaServer Faces: The complete reference**. 1.ed. McGraw-Hill Osborne. 2006. p. 287-320.

[SUN DEVELOPER NETWORK, 2005]. SUN. **Java BluePrints: Guidelines, patterns, and code for end-to-end applications.** Disponível em: <http://java.sun.com/reference/blueprints>. Acesso em: 13 jul. 2006.

[SUN, 2005] SUN. **Sun Microsystems**. Disponível em <http://www.sun.com>. Acesso em: 11 dez. 2005.

[SUN, 2005a] SUN. **Gmail**. Disponível em: <http://www.gmail.com>. Acesso em: 12 fev. 2006.

[SUN, 2005b] SUN. **JavaServer Faces Technology**. Disponível em: <http://java.sun.com/javaee/javaserverfaces> Acesso em: 1º nov. 2005.

[SUN, 2005c] SUN. **Java Enterprise Edition 5.0**. Disponível em: <http://java.sun.com/javaee/technologies/javaee5.jsp>. Acesso em: 1º nov. 2005.

[SUN, 2006] SUN. **DynaFaces**. Disponível em <https://jsf-extensions.dev.java.net/nonav/mvn/reference-ajax.html>. Acesso em: 20 out. 2006.

[SUN, 2006a] SUN. **Asynchronous JavaScript and XML**. Disponível em: <http://java.sun.com/developer/technicalArticles/J2EE/AJAX>. Acesso em: 15 jun. 2006.

[SUN, 2006b] SUN. **Google Maps**. Disponível em: <http://maps.google.com/>. Acesso em: 12 jun. 2006.

[THE WEB DESIGNER 5 HTTP PRIMER, 2006] **How does HTTP works**. Disponível em: <http://www.dmc.dit.ie/maim2002/mairead/practice/projects/MP4/What/what_2.html>. Acesso em: 23 jul. 2006.

[W3C, 2006] W3C. **World Wide Web Consortium.** Disponível em: <http://www.w3.org>. Acesso em: 15 jun. 2006.

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

[WIKIPEDIA, 2006] **WIKIPEDIA.** Wikipédia. Disponível em: <http://pt.wikipedia.org/wiki/J2EE>. Acesso em: 02 dez. 2006.

10 Anexos

10.1 Exemplo JSF

Classe Pessoa

```
package br.com.util;
import java.util.Date;

public class Pessoa {

            private Integer cdPessoa;
            private String nmPessoa;
            private String senha;
            private Date dtNascimento;
            private String nuIdenSujeito;
            private String email;
            private String nmCidade;
            private String nmEstado;

            public Integer getCdPessoa() {
                  return cdPessoa;
            }
            public void setCdPessoa(Integer cdPessoa) {
                  this.cdPessoa = cdPessoa;
            }
            public Date getDtNascimento() {
                  return dtNascimento;
            }
            public void setDtNascimento(Date dtNascimento) {
                  this.dtNascimento = dtNascimento;
            }
            public String getEmail() {
                  return email;
            }
            public void setEmail(String email) {
                  this.email = email;
            }
            public String getNmCidade() {
                  return nmCidade;
            }
            public void setNmCidade(String nmCidade) {
                  this.nmCidade = nmCidade;
            }
            public String getNmEstado() {
                  return nmEstado;
            }
            public void setNmEstado(String nmEstado) {
                  this.nmEstado = nmEstado;
            }
            public String getNmPessoa() {
                  return nmPessoa;
            }
            public void setNmPessoa(String nmPessoa) {
```

```
		this.nmPessoa = nmPessoa;
	}
	public String getNuIdenSujeito() {
		return nuIdenSujeito;
	}
	public void setNuIdenSujeito(String nuIdenSujeito) {
		this.nuIdenSujeito = nuIdenSujeito;
	}
	public String getSenha() {
		return senha;
	}
	public void setSenha(String senha) {
		this.senha = senha;
	}
}
```

Classe PessoaForm

```
package br.com.util;

import javax.faces.context.FacesContext;
import javax.faces.event.ValueChangeEvent;
import javax.faces.model.SelectItem;

public class PessoaForm{

	private Pessoa pessoa;
	private SelectItem[] listaEstados;
	private SelectItem[] listaCidades;

	public PessoaForm(){
		this.pessoa = new Pessoa();
		populeListaEstados();
		this.listaCidades = new SelectItem[]{
			new SelectItem("","")
		};

	}

	public Pessoa getPessoa() {
		return pessoa;
	}

	public void setPessoa(Pessoa pessoa) {
		this.pessoa = pessoa;
	}

	private void populeListaEstados(){
		listaEstados = new SelectItem[]{
			new SelectItem("RS","Rio Grande do Sul"),
			new SelectItem("SC","Santa Catarina"),
			new SelectItem("PR","Paraná"),
			new SelectItem("SP","São Paulo"),
			new SelectItem("RJ","Rio de Janeiro"),
			new SelectItem("ES","Espírito Santo"),
			new SelectItem("MG","Minas Gerais"),
		};

	}

	public SelectItem[] getListaEstados() {
```

```
		return listaEstados;
	}
	public void setListaEstados(SelectItem[] listaEstados) {
		this.listaEstados = listaEstados;
	}

	public void populeListaCidades(String sgEstado){

		System.out.println("sgEstado = "+sgEstado);

		if("RS".equalsIgnoreCase(sgEstado)){
			listaCidades = new SelectItem[]{
					new SelectItem("Porto Alegre","Porto Alegre"),
					new SelectItem("Alegrete","Alegrete"),
					new SelectItem("Bagé","Bagé"),
					new SelectItem("Caxias","Caxias"),
					new SelectItem("Passo Fundo","Passo Fundo"),
					new SelectItem("Pelotas","Pelotas")

				};
		}else if("SC".equalsIgnoreCase(sgEstado)){
			listaCidades = new SelectItem[]{
					new
SelectItem("Florianópolis","Florianópolis"),

					new SelectItem("Criciúma","Criciúma"),
					new SelectItem("Itajaí","Itajaí"),
					new SelectItem("Joinvile","Joinvile"),
					new SelectItem("São José","São José"),

					new SelectItem("Xavantina","Xavantina")
				};
		}else if("SC".equalsIgnoreCase(sgEstado)){
			listaCidades = new SelectItem[]{
					new SelectItem("Curitiba","Curitiba"),
					new SelectItem("Apucarana","Apucarana"),
					new SelectItem("Irati","Irati"),
					new SelectItem("Paranaguá","Paranaguá")

				};
		}else if("SP".equalsIgnoreCase(sgEstado)){
			listaCidades = new SelectItem[]{
					new SelectItem("São Paulo","São Paulo"),
					new SelectItem("Campinas","Campinas"),
					new SelectItem("Itú","Itú"),

					new SelectItem("Santos","Santos"),

					new SelectItem("Santo André","Santo André"),

					new SelectItem("São Bernardo","São Bernardo"),
					new SelectItem("São Caetano","São Caetano")
				};
		}else if("RJ".equalsIgnoreCase(sgEstado)){
			listaCidades = new SelectItem[]{
					new SelectItem("Rio de Janeiro","Rio de
Janeiro"),
					new SelectItem("Agulhas Negras","Agulhas
Negras"),
					new SelectItem("Bangú","Bangú"),
					new SelectItem("Cabo Frio","Cabo Frio"),
```

```
                              new SelectItem("Granja","Granja"),
                              new SelectItem("Macaé","Macaé")

                        };
            }else if("ES".equalsIgnoreCase(sgEstado)){
                  listaCidades = new SelectItem[]{
                              new SelectItem("Vitória","Vitória"),
                              new SelectItem("Argolas","Argolas"),
                              new SelectItem("Goiabeiras","Goiabeiras"),
                              new SelectItem("Guaraná","Guaraná"),

                        };
            }else if("MG".equalsIgnoreCase(sgEstado)){
                  listaCidades = new SelectItem[]{
                              new SelectItem("Belo Horizonte","Belo
Horizonte"),
                              new SelectItem("Alfenas","Alfenas"),
                              new SelectItem("Canastra","Canastra"),
                              new SelectItem("Esmeraldas","Esmeraldas"),
                              new SelectItem("Diamantina","Diamantina"),
                              new SelectItem("Mamonas","Mamonas")
                                                      };
            }
      }

      public SelectItem[] getListaCidades() {
            return listaCidades;
      }

      public void setListaCidades(SelectItem[] listaCidades) {
            this.listaCidades = listaCidades;
      }

      public void estadoChanged(ValueChangeEvent event){
            System.out.println("entrou no estado changed");
            FacesContext context = FacesContext.getCurrentInstance();
            String sgEstado = (String)event.getNewValue();


            this.populeListaCidades(sgEstado);

      }
}
```

Classe CPFValidator

```
package br.com.util.validator;

import java.util.Vector;

import javax.faces.application.FacesMessage;
import javax.faces.component.UIComponent;
import javax.faces.context.FacesContext;
import javax.faces.validator.Validator;
import javax.faces.validator.ValidatorException;

public class CPFValidator implements Validator{

      private final int tamanhoCpf = 11;
```

```
      public void validate(FacesContext context, UIComponent component,
Object value) throws ValidatorException {
            if(value == null){
                  return;
            }

            if(!confirmaCPF(value.toString())){
                  FacesMessage msg = new
FacesMessage(FacesMessage.SEVERITY_ERROR, "ERROR1", "CPF inválido");
                  throw new ValidatorException(msg);
            }

      }

      /* Cálculo do CPF foi retirado da página
http://www2.fundao.pro.br/articles.asp?cod=23 */

      private boolean confirmaCPF (String strCpf ) {
            int     d1, d2;
            int     digito1, digito2, resto;
            int     digitoCPF;
            String  nDigResult;

            d1 = d2 = 0;
            digito1 = digito2 = resto = 0;

            for (int nCount = 1; nCount < strCpf.length() -1; nCount++){

               digitoCPF = Integer.valueOf (strCpf.substring(nCount -1,
nCount)).intValue();
               d1 = d1 + ( 11 - nCount ) * digitoCPF;
               d2 = d2 + ( 12 - nCount ) * digitoCPF;
            }

            resto = (d1 % 11);

            if (resto < 2)
               digito1 = 0;
            else
               digito1 = 11 - resto;

            d2 += 2 * digito1;
            resto = (d2 % 11);

            if (resto < 2)
               digito2 = 0;
            else
               digito2 = 11 - resto;

            String nDigVerific = strCpf.substring (strCpf.length()-2,
strCpf.length());

            nDigResult = String.valueOf(digito1) + String.valueOf(digito2);

            return nDigVerific.equals(nDigResult);
         }
```

```
}
```

Arquivo faces-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE faces-config PUBLIC "-//Sun Microsystems, Inc.//DTD JavaServer
Faces Config 1.1//EN" "http://java.sun.com/dtd/web-facesconfig_1_1.dtd">

<faces-config>
      <navigation-rule>
            <from-view-id>/index.jsp</from-view-id>
            <navigation-case>
                  <from-outcome>login</from-outcome>
                  <to-view-id>/result.jsp</to-view-id>
            </navigation-case>
      </navigation-rule>
      <managed-bean>
            <managed-bean-name>pessoa</managed-bean-name>
            <managed-bean-class>br.com.util.Pessoa</managed-bean-class>
            <managed-bean-scope>session</managed-bean-scope>
      </managed-bean>


      <managed-bean>

            <managed-bean-name>pessoaForm</managed-bean-name>
            <managed-bean-class>br.com.util.PessoaForm</managed-bean-class>
            <managed-bean-scope>session</managed-bean-scope>

      </managed-bean>

      <validator>
            <validator-id>br.com.util.validator.CPFValidator</validator-id>
            <validator-class>br.com.util.validator.CPFValidator</validator-
class>
      </validator>

</faces-config>
```

Arquivo web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="2.4"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
  <context-param>
    <param-name>javax.faces.CONFIG_FILES</param-name>
    <param-value>/WEB-INF/faces-config.xml</param-value>
  </context-param>
  <servlet>
    <servlet-name>Faces Servlet</servlet-name>
    <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
    <load-on-startup>0</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>*.faces</url-pattern>
  </servlet-mapping>
  <welcome-file-list>
```

```
  <welcome-file>start.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

Arquivo index.jsp

```
<%@ include file="./layout/cabecalhofamilia.jsp" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>


<html>
  <head>
    <title>Index</title>
  </head>
  <body>
  <f:view>
      <h:form>
      <table width="100%" border="0" cellpadding="0" cellspacing="0">
      <tr>
            <td width="50">
            Nome:
            </td>
            <td width="70">
                  <h:inputText value="#{pessoaForm.pessoa.nmPessoa}"/>
            </td>
            <td width="*"> </td>
      </tr>
      <tr>
            <td>
            Senha:
            </td>
            <td>
            <h:inputSecret id="senha" value="#{pessoaForm.pessoa.senha}"
maxlength="15" />
            </td>
            <td></td>
      </tr>
      <tr>
            <td>
            CPF:
            </td>
            <td>
            <h:inputText id="cpf"
value="#{pessoaForm.pessoa.nuIdenSujeito}" maxlength="11">
            <f:validator validatorId="br.com.util.validator.CPFValidator"/>
            </h:inputText>
            </td>
            <td><h:message for="cpf" style="color: red; text-decoration:
overline;text-align: left"/>
            <td>
      </tr>
      <tr>
            <td>
            Email:
            </td>
            <td>
                  <h:inputText value="#{pessoaForm.pessoa.email}"/>
            </td>
            <td width="100"> </td>
```

```
      </tr>
      <tr>
            <td>
            Estado:
            </td>
            <td>
            <h:selectOneListbox style="width: 100%" size="1"
value="#{pessoaForm.pessoa.nmEstado}" onchange="submit()"
valueChangeListener="#{pessoaForm.estadoChanged}">
            <f:selectItems value="#{pessoaForm.listaEstados}"/>
                                    </h:selectOneListbox>
            </td>
            <td width="100"> </td>
      </tr>
      <tr>
            <td>
            Cidade:
            </td>
            <td>
            <h:selectOneListbox style="width: 100%" size="1"
value="#{pessoaForm.pessoa.nmCidade}">
            <f:selectItems value="#{pessoaForm.listaCidades}"/>
            </h:selectOneListbox>
            </td>
            <td width="100"> </td>
      </tr>
</table>
      <h:commandButton value="OK" action="login"/>
</h:form>
</f:view>
</body>
</html>
<%@ include file="./layout/rodape.htm" %>
```

Arquivo result.jsp

```
<%@ include file="./layout/cabecalhofamilia.jsp" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>


<html>
  <head>
    <title>Resultado</title>
  </head>
  <body>
  <f:view>
  <h:form>
      Cadastro efetuado com sucesso!<br>

      Nome: <h:outputText value="#{pessoaForm.pessoa.nmPessoa}"/><br>
      CPF: <h:outputText value="#{pessoaForm.pessoa.nuIdenSujeito}"/><br>
      Nascimento: <h:outputText
value="#{pessoaForm.pessoa.dtNascimento}"/><br>
      Email: <h:outputText value="#{pessoaForm.pessoa.email}"/><br>
      Cidade: <h:outputText value="#{pessoaForm.pessoa.nmCidade}"/><br>
      Estado: <h:outputText value="#{pessoaForm.pessoa.nmEstado}"/><br>

  </h:form>
</f:view>
```

```
</body>
</html>
<%@ include file="./layout/rodape.htm" %>
```

Arquivo Start.jsp

```
<%@ include file="./layout/cabecalhofamilia.jsp" %>
<html>
  <head>
    <title>My JSP 'start.jsp' starting page</title>
  </head>
  <body>
            <jsp:forward page="index.faces"/>
  </body>
</html>
```

10.2 Exemplo JSF + AJAX

Classe GerenciadorPessoas

```
package br.com.util;
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class GerenciadorPessoas extends HttpServlet{

      private ServletConfig servletConfig = null;
       public void destroy() {
        servletConfig = null;
       }

       public ServletConfig getServletConfig() {
         return (this.servletConfig);
       }

       public String getServletInfo() {
        return (this.getClass().getName());
       }

       public void init(ServletConfig servletConfig) throws
ServletException {
             this.servletConfig = servletConfig;
       }

       public void doGet(HttpServletRequest request,HttpServletResponse
response)throws java.io.IOException,ServletException {

             java.io.PrintWriter out=response.getWriter();
             String strSenha = (String)request.getParameter("senha");
             String strCPF = (String)request.getParameter("cpf");
             if((strSenha != null)&&!(strSenha.equals(""))){

                   boolean senhaValida = false;
                   int tamanhoSenha = strSenha.length();
                   int tamanhoSenhaIdeal = 5;
                   if(tamanhoSenha >= tamanhoSenhaIdeal){
                         senhaValida = true;
                   }
                   out.print("senha|"+senhaValida);
             }else if((strCPF != null)&&!(strCPF.equals(""))){
                   boolean cpfValido = this.confirmaCPF(strCPF);
                   out.print("cpf|"+cpfValido);
             }
             out.flush();
      }

       public void doPost(HttpServletRequest request,HttpServletResponse
response) throws java.io.IOException, ServletException {
             doGet(request, response);
       }
```

```
       private boolean confirmaCPF (String strCpf ) {
            int     d1, d2;
            int     digito1, digito2, resto;
            int     digitoCPF;
            String  nDigResult;
            d1 = d2 = 0;
            digito1 = digito2 = resto = 0;

            for (int nCount = 1; nCount < strCpf.length() -1; nCount++){

               digitoCPF = Integer.valueOf (strCpf.substring(nCount -1,
nCount)).intValue();
               d1 = d1 + ( 11 - nCount ) * digitoCPF;
               d2 = d2 + ( 12 - nCount ) * digitoCPF;
            }
            resto = (d1 % 11);

            if (resto < 2)
               digito1 = 0;
            else
               digito1 = 11 - resto;

            d2 += 2 * digito1;
            resto = (d2 % 11);

            if (resto < 2)
               digito2 = 0;
            else
               digito2 = 11 - resto;
            String nDigVerific = strCpf.substring (strCpf.length()-2,
strCpf.length());
            nDigResult = String.valueOf(digito1) + String.valueOf(digito2);
            return nDigVerific.equals(nDigResult);
         }
}
```

Classe Pessoa

```
package br.com.util;
public class Pessoa {
      private Integer cdPessoa;
      private String nmPessoa;
      private String nuIdentPessoa;
      private String senha;
      private String email;
      private String nmEstado;
      private String nmCidade;

      public String getNuIdentPessoa() {
            return nuIdentPessoa;
      }
      public void setNuIdentPessoa(String nuIdentPessoa) {
            this.nuIdentPessoa = nuIdentPessoa;
      }
      public Integer getCdPessoa() {
            return cdPessoa;
      }
      public void setCdPessoa(Integer cdPessoa) {
            this.cdPessoa = cdPessoa;
      }
```

```
	public String getNmPessoa() {
		return nmPessoa;
	}
	public void setNmPessoa(String nmPessoa) {
		this.nmPessoa = nmPessoa;
	}
	public String getEmail() {
		return email;
	}
	public void setEmail(String email) {
		this.email = email;
	}
	public String getNmCidade() {
		return nmCidade;
	}
	public void setNmCidade(String nmCidade) {
		this.nmCidade = nmCidade;
	}
	public String getNmEstado() {
		return nmEstado;
	}
	public void setNmEstado(String nmEstado) {
		this.nmEstado = nmEstado;
	}
	public String getSenha() {
		return senha;
	}
	public void setSenha(String senha) {
		this.senha = senha;
	}
}
```

Classe PessoaForm

```
package br.com.util;

import javax.faces.context.FacesContext;
import javax.faces.event.ValueChangeEvent;
import javax.faces.model.SelectItem;

public class PessoaForm{
	private Pessoa pessoa;
	private SelectItem[] listaEstados;
	private SelectItem[] listaCidades;

	public PessoaForm(){
		this.pessoa = new Pessoa();
		populeListaEstados();
		this.listaCidades = new SelectItem[]{
			new SelectItem("","")
		};
	}

	public Pessoa getPessoa() {
		return pessoa;
	}

	public void setPessoa(Pessoa pessoa) {
		this.pessoa = pessoa;
	}
```

```
	private void populeListaEstados(){
		listaEstados = new SelectItem[]{
			new SelectItem("RS","Rio Grande do Sul"),
			new SelectItem("SC","Santa Catarina"),
			new SelectItem("PR","Paraná"),
			new SelectItem("SP","São Paulo"),
			new SelectItem("RJ","Rio de Janeiro"),
			new SelectItem("ES","Espírito Santo"),
			new SelectItem("MG","Minas Gerais"),
		};

	}

	public SelectItem[] getListaEstados() {
		return listaEstados;
	}
	public void setListaEstados(SelectItem[] listaEstados) {
		this.listaEstados = listaEstados;
	}

	public void populeListaCidades(String sgEstado){

		if("RS".equalsIgnoreCase(sgEstado)){
			listaCidades = new SelectItem[]{
					new SelectItem("Porto Alegre","Porto Alegre"),
					new SelectItem("Alegrete","Alegrete"),
					new SelectItem("Bagé","Bagé"),
					new SelectItem("Caxias","Caxias"),
					new SelectItem("Passo Fundo","Passo Fundo"),
					new SelectItem("Pelotas","Pelotas")

				};
		}else if("SC".equalsIgnoreCase(sgEstado)){
			listaCidades = new SelectItem[]{
					new
SelectItem("Florianópolis","Florianópolis"),

					new SelectItem("Criciúma","Criciúma"),
					new SelectItem("Itajaí","Itajaí"),
					new SelectItem("Joinvile","Joinvile"),
					new SelectItem("São José","São José"),

					new SelectItem("Xavantina","Xavantina")
				};
		}else if("SC".equalsIgnoreCase(sgEstado)){
			listaCidades = new SelectItem[]{
					new SelectItem("Curitiba","Curitiba"),
					new SelectItem("Apucarana","Apucarana"),
					new SelectItem("Irati","Irati"),
					new SelectItem("Paranaguá","Paranaguá")

				};
		}else if("SP".equalsIgnoreCase(sgEstado)){
			listaCidades = new SelectItem[]{
					new SelectItem("São Paulo","São Paulo"),
					new SelectItem("Campinas","Campinas"),
					new SelectItem("Itú","Itú"),

					new SelectItem("Santos","Santos"),

					new SelectItem("Santo André","Santo André"),
```

```
                              new SelectItem("São Bernardo","São Bernardo"),
                              new SelectItem("São Caetano","São Caetano")
                        };
            }else if("RJ".equalsIgnoreCase(sgEstado)){
                  listaCidades = new SelectItem[]{
                              new SelectItem("Rio de Janeiro","Rio de
Janeiro"),
                              new SelectItem("Agulhas Negras","Agulhas
Negras"),
                              new SelectItem("Bangú","Bangú"),
                              new SelectItem("Cabo Frio","Cabo Frio"),
                              new SelectItem("Granja","Granja"),
                              new SelectItem("Macaé","Macaé")

                        };
            }else if("ES".equalsIgnoreCase(sgEstado)){
                  listaCidades = new SelectItem[]{
                              new SelectItem("Vitória","Vitória"),
                              new SelectItem("Argolas","Argolas"),
                              new SelectItem("Goiabeiras","Goiabeiras"),
                              new SelectItem("Guaraná","Guaraná"),

                        };
            }else if("MG".equalsIgnoreCase(sgEstado)){
                  listaCidades = new SelectItem[]{
                              new SelectItem("Belo Horizonte","Belo
Horizonte"),
                              new SelectItem("Alfenas","Alfenas"),
                              new SelectItem("Canastra","Canastra"),
                              new SelectItem("Esmeraldas","Esmeraldas"),
                              new SelectItem("Diamantina","Diamantina"),
                              new SelectItem("Mamonas","Mamonas")
                                                      };
            }
      }

      public SelectItem[] getListaCidades() {
            return listaCidades;
      }

      public void setListaCidades(SelectItem[] listaCidades) {
            this.listaCidades = listaCidades;
      }

      public void estadoChanged(ValueChangeEvent event){
            System.out.println("entrou no estado changed");
            FacesContext context = FacesContext.getCurrentInstance();
            String sgEstado = (String)event.getNewValue();
            this.populeListaCidades(sgEstado);
      }
}
```

Arquivo faces-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE faces-config PUBLIC "-//Sun Microsystems, Inc.//DTD JavaServer
Faces Config 1.1//EN" "http://java.sun.com/dtd/web-facesconfig_1_1.dtd">
<faces-config>
      <navigation-rule>
            <from-view-id>/index.jsp</from-view-id>
            <navigation-case>
```

```
                <from-outcome>login</from-outcome>
                <to-view-id>/result.jsp</to-view-id>
            </navigation-case>
        </navigation-rule>
        <managed-bean>
            <managed-bean-name>pessoa</managed-bean-name>
            <managed-bean-class>br.com.util.Pessoa</managed-bean-class>
            <managed-bean-scope>session</managed-bean-scope>
        </managed-bean>
        <managed-bean>
            <managed-bean-name>pessoaForm</managed-bean-name>
            <managed-bean-class>br.com.util.PessoaForm</managed-bean-class>
            <managed-bean-scope>session</managed-bean-scope>
        </managed-bean>
</faces-config>
```

Arquivo web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="2.4"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
  <context-param>
    <param-name>javax.faces.CONFIG_FILES</param-name>
    <param-value>/WEB-INF/faces-config.xml</param-value>
  </context-param>
  <servlet>
    <servlet-name>Faces Servlet</servlet-name>
    <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
    <load-on-startup>0</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>*.faces</url-pattern>
  </servlet-mapping>
  <welcome-file-list>
      <welcome-file>start.jsp</welcome-file>
  </welcome-file-list>
  <servlet>
      <servlet-name>GerenciadorPessoas</servlet-name>
      <servlet-class>br.com.util.GerenciadorPessoas</servlet-class>
  </servlet>
  <servlet-mapping>
      <servlet-name>GerenciadorPessoas</servlet-name>
      <url-pattern>/GerenciadorPessoas</url-pattern>
  </servlet-mapping>
</web-app>
```

Arquivo start.jsp

```
<html>
  <head>
  </head>
  <body>
            <jsp:forward page="index.faces"/>
  </body>
</html>
```

Arquivo index.jsp

```
<%@ include file="./layout/cabecalhofamilia.jsp" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>

<html>
  <head>
      <TITLE>index</TITLE>
  </head>
  <body>
  <f:view>
  <h:form>
  <table width="100%" border="0">
  <tr>
      <td width="80">Codigo:</td>
      <td width="*"><h:inputText value="#{pessoaForm.pessoa.cdPessoa}"
id="cdpessoa" onchange="getCustomerInfo(this)"/></td>
      <td width="350"> </td>
  </tr>
  <tr>
      <td>Nome:</td>
      <td><h:inputText value="#{pessoaForm.pessoa.nmPessoa}" id="nmpessoa"
onchange="getCustomerInfo(this)"/></td>
      <td></td>
  </tr>
  <tr>
      <td>Senha:</td>
      <td><h:inputText value="#{pessoaForm.pessoa.senha}" id="senha"
onchange="getCustomerInfo(this)"/></td>
      <td>
       <div id="senhafraca" style="display: none;" align="left">
            <LABEL style="color: red;">A senha deve conter letras e número.
</LABEL>
      </div>
      </td>
  </tr>
  <tr>
      <td>CPF:</td>
      <td><h:inputText value="#{pessoaForm.pessoa.nuIdentPessoa}"
maxlength="11" id="nuIdentPessoa" onchange="getCustomerInfo(this)"/></td>
      <td>
      <div id="cpfinvalido" style="display: none;" align="left">
      <LABEL style="color: red;">CPF inválido.</LABEL>
      </div>
      </td>
</tr>
</table>
<h:commandButton value="OK" action="login"/>
</h:form>
</f:view>
</body>
</html>
<%@ include file="./layout/rodape.htm" %>
<script language="JavaScript" type="text/JavaScript">
   var request = false;
   try {
     request = new XMLHttpRequest();
   } catch (trymicrosoft) {
     try {
       request = new ActiveXObject("Msxml2.XMLHTTP");
     } catch (othermicrosoft) {
```

```
        try {
          request = new ActiveXObject("Microsoft.XMLHTTP");
        } catch (failed) {
          request = false;
        }
      }
    }
    if (!request)
      alert("Error initializing XMLHttpRequest!");

    function getCustomerInfo( idObj) {

        var senha = document.getElementById("_id0:senha").value;
        var cpf = document.getElementById("_id0:nuIdentPessoa").value;
        var url =
"/cadastroAjax/GerenciadorPessoas?senha="+senha+"&cpf="+cpf;

      request.open("GET", url, true);
      request.onreadystatechange = updatePage;
      request.send(null);
    }

    function updatePage() {
      if (request.readyState == 4) {
        if (request.status == 200) {
          var response = request.responseText.split("|");

          divSenha = document.getElementById('senhafraca');
              divCPF = document.getElementById('cpfinvalido');

              if(response[0] == 'senha'){
                    if(response[1] == 'false'){
                          divSenha.style.display = 'inline';
                    }else{
                          divSenha.style.display = 'none';
                    }
              }else if(response[0] == 'cpf'){
                   if(response[1] == 'false'){
                          divCPF.style.display = 'inline';
                    }else{
                          divCPF.style.display = 'none';

                    }
              }

        } else if (request.status == 404) {
          alert ("Requested URL is not found.");
        } else if (request.status == 403) {
          alert("Access denied.");
        } else
          alert("status is " + request.status);
      }
    }
</script>
```

Arquivo result.jsp

```
<%@ include file="./layout/cabecalhofamilia.jsp" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
```

```
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>


<html>
  <head>
    <title>Resultado</title>
  </head>
  <body>
  <f:view>
  <h:form>
      Cadastro efetuado com sucesso!<br>

      Nome: <h:outputText value="#{pessoaForm.pessoa.nmPessoa}"/><br>
      CPF: <h:outputText value="#{pessoaForm.pessoa.nuIdenSujeito}"/><br>
Nascimento: <h:outputText
value="#{pessoaForm.pessoa.dtNascimento}"/><br>
      Email: <h:outputText value="#{pessoaForm.pessoa.email}"/><br>
      Cidade: <h:outputText value="#{pessoaForm.pessoa.nmCidade}"/><br>
      Estado: <h:outputText value="#{pessoaForm.pessoa.nmEstado}"/><br>

  </h:form>
</f:view>
</body>
</html>
<%@ include file="./layout/rodape.htm" %>
```

10.3 Exemplo ICEfaces

Classe Arquivo

```
package br.com.icefaces.util;

import com.icesoft.faces.webapp.xmlhttp.PersistentFacesState;

public class Arquivo {

	private String status;
	private String arquivoLocal;

	public Arquivo(){}

	public String getArquivoLocal() {
		return arquivoLocal;
	}
	public void setArquivoLocal(String arquivoLocal) {
		this.arquivoLocal = arquivoLocal;
	}
	public String getStatus() {
		return status;
	}
	public void setStatus(String status) {
		this.status = status;
	}

}
```

Classe Autocomplete

```
package br.com.icefaces.util;

import java.util.ArrayList;
import java.util.List;

import javax.faces.event.ValueChangeEvent;
import javax.faces.model.SelectItem;

import com.icesoft.faces.component.selectinputtext.SelectInputText;

public class Autocomplete {

	private List<SelectItem> listaNomes;
	private String nomeAtual;
	private List<SelectItem> listaNomesFixo;

	public Autocomplete(){
		listaNomes = new ArrayList<SelectItem>();
		listaNomesFixo = new ArrayList<SelectItem>();
		nomeAtual = "";
		this.populeListaNomes();
		listaNomesFixo.addAll(listaNomes);
	}

	private void populeListaNomes(){
		listaNomes.add(new SelectItem("Anderson"));
```

```
            listaNomes.add(new SelectItem("Andrey"));
            listaNomes.add(new SelectItem("Augusto"));
            listaNomes.add(new SelectItem("Bruno"));
            listaNomes.add(new SelectItem("Caio"));
            listaNomes.add(new SelectItem("Denise"));
            listaNomes.add(new SelectItem("Diego"));
            listaNomes.add(new SelectItem("Douglas"));
            listaNomes.add(new SelectItem("Eder"));
            listaNomes.add(new SelectItem("Edson"));
            listaNomes.add(new SelectItem("Eduardo"));
            listaNomes.add(new SelectItem("Felipe"));
            listaNomes.add(new SelectItem("Fernando"));
            listaNomes.add(new SelectItem("Gabriel"));
            listaNomes.add(new SelectItem("Gilberto"));
            listaNomes.add(new SelectItem("Guilherme"));
            listaNomes.add(new SelectItem("Jeronimo"));
            listaNomes.add(new SelectItem("Joao"));
            listaNomes.add(new SelectItem("Leonardo"));
            listaNomes.add(new SelectItem("Luiz"));
            listaNomes.add(new SelectItem("Marcelo"));
            listaNomes.add(new SelectItem("Martín"));
            listaNomes.add(new SelectItem("Maximiliano"));
            listaNomes.add(new SelectItem("Paulo"));
            listaNomes.add(new SelectItem("Rafael"));
            listaNomes.add(new SelectItem("Sabrina"));
            listaNomes.add(new SelectItem("Tiago"));
            listaNomes.add(new SelectItem("Thiago"));
            listaNomes.add(new SelectItem("Tie"));
            listaNomes.add(new SelectItem("Victor"));
      }

      public List getListaNomes() {
            if(listaNomes == null){
                  listaNomes = new ArrayList();
            }
            return listaNomes;
      }

      public void setListaNomes(List listaNomes) {
            this.listaNomes = listaNomes;
      }

      public void updateList(ValueChangeEvent event){
            listaNomes = new ArrayList();
            if(event.getComponent() instanceof SelectInputText ){

                  SelectInputText autocomplete =
(SelectInputText)event.getComponent();
                  String valorDigitado =
(String)autocomplete.getValue().toString();

                  int tamanhoListaNomes = listaNomesFixo.size();
                  String valorAtual;
                  for(int i=0; i<tamanhoListaNomes;i++){
                        valorAtual =
((String)((SelectItem)listaNomesFixo.get(i)).getValue());

      if(valorAtual.toUpperCase().contains(valorDigitado.toUpperCase())){

      listaNomes.add((SelectItem)listaNomesFixo.get(i));
                  }
```

```
                }
            }
        }

        public String getNomeAtual() {
            return nomeAtual;
        }

        public void setNomeAtual(String nomeAtual) {
            this.nomeAtual = nomeAtual;
        }

        public List getListaNomesFixo() {
            return listaNomesFixo;
        }

        public void setListaNomesFixo(List listaNomesFixo) {
            this.listaNomesFixo = listaNomesFixo;
        }

}
```

Classe Cidade

```
package br.com.icefaces.util;
public class Cidade {
        private Integer cdCidade;
        private String nmCidade;

        public Integer getCdCidade() {
            return cdCidade;
        }
        public void setCdCidade(Integer cdCidade) {
            this.cdCidade = cdCidade;
        }
        public String getNmCidade() {
            return nmCidade;
        }
        public void setNmCidade(String nmCidade) {
            this.nmCidade = nmCidade;
        }

}
```

Classe Endereço

```
package br.com.icefaces.util;
public class Endereco {

        private String nmRua;
        private Integer nuResidencia;
        private String deComplemento;
        private String nmBairro;
        private Cidade cidade;
        private Estado estado;

        public Endereco(){
            this.cidade = new Cidade();
            this.estado = new Estado();
```

```
	}

	public Cidade getCidade() {
		return cidade;
	}
	public void setCidade(Cidade cidade) {
		this.cidade = cidade;
	}
	public String getDeComplemento() {
		return deComplemento;
	}
	public void setDeComplemento(String deComplemento) {
		this.deComplemento = deComplemento;
	}
	public String getNmBairro() {
		return nmBairro;
	}
	public void setNmBairro(String nmBairro) {
		this.nmBairro = nmBairro;
	}
	public String getNmRua() {
		return nmRua;
	}
	public void setNmRua(String nmRua) {
		this.nmRua = nmRua;
	}
	public Integer getNuResidencia() {
		return nuResidencia;
	}
	public void setNuResidencia(Integer nuResidencia) {
		this.nuResidencia = nuResidencia;
	}
	public Estado getEstado() {
		return estado;
	}
	public void setEstado(Estado estado) {
		this.estado = estado;
	}
}
```

Classe Estado

```
package br.com.icefaces.util;
public class Estado {
	private Integer cdEstado;
	private String deEstado;
	public Integer getCdEstado() {
		return cdEstado;
	}
	public void setCdEstado(Integer cdEstado) {
		this.cdEstado = cdEstado;
	}
	public String getDeEstado() {
		return deEstado;
	}
	public void setDeEstado(String deEstado) {
		this.deEstado = deEstado;
	}
}
```

Classe GerenciadorEstadoCidade

```
package br.com.icefaces.util;
import java.util.ArrayList;
import java.util.List;

import javax.faces.event.ValueChangeEvent;
import javax.faces.model.SelectItem;
import com.icesoft.faces.component.ext.HtmlSelectOneMenu;
import com.icesoft.faces.component.selectinputtext.SelectInputText;

public class GerenciadorEstadoCidade {
      private List<SelectItem> listaEstados;
      private List<SelectItem> listaCidades;

      public GerenciadorEstadoCidade(){
            listaEstados = new ArrayList<SelectItem>();
            listaCidades = new ArrayList<SelectItem>();
            this.populeListaEstados();
      }
      private void populeListaEstados(){

            listaEstados.add(new SelectItem("Paraná"));
            listaEstados.add(new SelectItem("Santa Catarina"));
            listaEstados.add(new SelectItem("Rio Grande do Sul"));

      }

      public void updateList(ValueChangeEvent event){
            if(event.getComponent() instanceof HtmlSelectOneMenu ){
                  listaCidades.clear();
                  HtmlSelectOneMenu selectOneMenu =
(HtmlSelectOneMenu)event.getComponent();
                  String valorSelecionado =
(String)selectOneMenu.getValue();
                  if("Santa Catarina".equalsIgnoreCase(valorSelecionado)){
                        System.out.println("Selecionado Santa Catarina");
                        listaCidades.add(new SelectItem("Florianópolis"));

                        listaCidades.add(new SelectItem("Blumenau"));
                        listaCidades.add(new SelectItem("Chapecó"));
                        listaCidades.add(new SelectItem("Joinvile"));
                        listaCidades.add(new SelectItem("Xavantina"));

                  }else if("Paraná".equalsIgnoreCase(valorSelecionado)){
                        System.out.println("Selecionado Paraná");
                        listaCidades.add(new SelectItem("Curitiba"));
                        listaCidades.add(new SelectItem("Londrina"));
                        listaCidades.add(new SelectItem("Maringá"));
                        listaCidades.add(new SelectItem("São José dos
Pinhais"));

                  }else if("Rio Grande do
Sul".equalsIgnoreCase(valorSelecionado)){
                        System.out.println("Selecionado Rio Grande do
Sul");
                        listaCidades.add(new SelectItem("Porto Alegre"));
                        listaCidades.add(new SelectItem("Alegrete"));
                        listaCidades.add(new SelectItem("Bento
Gonçalves"));
                        listaCidades.add(new SelectItem("Caxias"));
```

```
                        listaCidades.add(new SelectItem("Cruz Alta"));

                  }else{
                        System.out.println("Nenhum selecionado");
                  }
            }
      }

      public List<SelectItem> getListaEstados() {
            return listaEstados;
      }

      public void setListaEstados(List<SelectItem> listaEstados) {
            this.listaEstados = listaEstados;
      }

      public List<SelectItem> getListaCidades() {
            return listaCidades;
      }

      public void setListaCidades(List<SelectItem> listaCidades) {
            this.listaCidades = listaCidades;
      }

}
```

Classe GerenciadorPessoas

```
package br.com.icefaces.util;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Date;
import java.util.List;

public class GerenciadorPessoas {
      private List<Pessoa> listaPessoas;
      public GerenciadorPessoas(){
            this.listaPessoas = new ArrayList<Pessoa>();
            this.populeListaPessoas();
      }

      private void populeListaPessoas(){
            this.adicionaPessoa(new Pessoa("Anderson",new
Date("01/01/1985"),"0000000","p","0000000"));
            this.adicionaPessoa(new Pessoa("Andrey",new
Date("27/08/1987"),"0000001","p","0000001"));
            this.adicionaPessoa(new Pessoa("Augusto",new
Date("03/04/1988"),"0000002","p","0000002"));
            this.adicionaPessoa(new Pessoa("Bruno",new
Date("27/07/1983"),"0000003","p","0000003"));
            this.adicionaPessoa(new Pessoa("Caio",new
Date("12/12/1985"),"0000004","p","0000004"));
            this.adicionaPessoa(new Pessoa("Denise",new
Date("17/01/1984"),"0000005","p","0000005"));
            this.adicionaPessoa(new Pessoa("Diego",new
Date("03/06/1981"),"0000006","p","0000006"));
            this.adicionaPessoa(new Pessoa("Douglas",new
Date("01/08/1985"),"0000007","p","0000007"));
            this.adicionaPessoa(new Pessoa("Eder",new
Date("19/07/1985"),"0000008","p","0000008"));
```

```
            this.adicionaPessoa(new Pessoa("Edson",new
Date("06/09/1989"),"0000009","p","0000009"));
            this.adicionaPessoa(new Pessoa("Eduardo",new
Date("01/01/1985"),"0000010","p","00000010"));
            this.adicionaPessoa(new Pessoa("Felipe",new
Date("28/11/1985"),"0000011","p","00000011"));
            this.adicionaPessoa(new Pessoa("Fernando",new
Date("20/04/1985"),"0000012","p","00000012"));
            this.adicionaPessoa(new Pessoa("Gabriel",new
Date("31/01/1989"),"0000013","p","00000013"));
            this.adicionaPessoa(new Pessoa("Gilberto",new
Date("24/09/1985"),"0000014","p","00000014"));
            this.adicionaPessoa(new Pessoa("Guilherme",new
Date("14/01/1985"),"0000015","p","00000015"));
            this.adicionaPessoa(new Pessoa("Jeronimo",new
Date("11/12/1987"),"0000016","p","00000016"));
            this.adicionaPessoa(new Pessoa("Joao",new
Date("04/01/1988"),"0000017","p","00000017"));
            this.adicionaPessoa(new Pessoa("Leonardo",new
Date("27/11/1986"),"0000018","p","00000018"));
            this.adicionaPessoa(new Pessoa("Luiz",new
Date("02/10/1985"),"0000019","p","00000019"));
            this.adicionaPessoa(new Pessoa("Marcelo",new
Date("27/09/1985"),"0000020","p","00000020"));
            this.adicionaPessoa(new Pessoa("Martín",new
Date("05/07/1983"),"0000021","p","00000021"));
            this.adicionaPessoa(new Pessoa("Maximiliano",new
Date("27/08/1985"),"0000022","p","00000022"));
            this.adicionaPessoa(new Pessoa("Paulo",new
Date("07/10/1985"),"0000023","p","00000023"));
            this.adicionaPessoa(new Pessoa("Rafael",new
Date("12/06/1983"),"0000024","p","00000024"));
            this.adicionaPessoa(new Pessoa("Sabrina",new
Date("27/12/1960"),"0000025","p","00000025"));
            this.adicionaPessoa(new Pessoa("Tiago",new
Date("17/06/1956"),"0000026","p","00000026"));
            this.adicionaPessoa(new Pessoa("Thiago",new
Date("12/10/1953"),"0000027","p","00000027"));
            this.adicionaPessoa(new Pessoa("Tie",new
Date("25/04/1985"),"0000028","p","00000028"));
            this.adicionaPessoa(new Pessoa("Victor",new
Date("06/04/1985"),"0000029","p","00000029"));

            for(Pessoa p : this.getListaPessoas() ){
                  p.getEndereco().getCidade().setNmCidade("Florianopolis");
                  p.getEndereco().getEstado().setDeEstado("Santa
Catarina");
            }

      }

      public void adicionaPessoa(Pessoa pessoa){

      if((pessoa.getCdPessoa()!=null)&&!(pessoa.getCdPessoa().equals(""))){
//                EDITA PESSOA
                  int tamanhoLista = listaPessoas.size();
                  Pessoa pessoaAux;
                  for( int i=0; i<tamanhoLista; i++ ){
                        pessoaAux = listaPessoas.get(i);
                        if(pessoaAux.getCdPessoa() ==
pessoa.getCdPessoa()){
```

```
                        //é a pessoa para ser editada
                              this.listaPessoas.remove(i);
                              this.listaPessoas.add(pessoa);
                        }
                  }
            }else{
//                INSERI PESSOA
                  pessoa.setCdPessoa(listaPessoas.size());
                  this.listaPessoas.add(pessoa);
            }
      }

      public Pessoa getPessoa(Integer cdPessoa){
            int numeroPessoas = this.getListaPessoas().size();
            Pessoa pessoaResposta;
            for(int i=0;i<numeroPessoas;i++){
                  pessoaResposta = this.getListaPessoas().get(i);
                  if(pessoaResposta.getCdPessoa() == cdPessoa){
                        return pessoaResposta;
                  }
            }
            return new Pessoa();
      }

      public void removaPessoa(Integer cdPessoa){
            int numeroPessoas = this.listaPessoas.size();
            Pessoa pessoaAux;
            for( int i=0;i<numeroPessoas;i++ ){
                  pessoaAux = listaPessoas.get(i);
                  if( pessoaAux.getCdPessoa() == cdPessoa ){
                  //pessoa a ser deletada da lista
                        this.getListaPessoas().remove(i);
                        i=numeroPessoas;
                  }
            }
      }

    public List<Pessoa> getListPessoaOrdenada(){
      Collections.sort(this.getListaPessoas());
      return this.getListaPessoas();
    }

      public List<Pessoa> getListaPessoas() {
            return listaPessoas;
      }

      public void setListaPessoas(List<Pessoa> listaPessoas) {
            this.listaPessoas = listaPessoas;
      }
}
```

Classe Pessoa

```
package br.com.icefaces.util;
import java.text.SimpleDateFormat;
import java.util.Collection;
import java.util.Date;
import java.util.List;
public class Pessoa implements Comparable{
      private Integer cdPessoa;
```

```
	//dados pessoais
	private String nome;
	private String sobrenome;
	private Date dtNascimento;
	private String nuCPF;
	private String emailPessoal;
	private String nuFonePessoal;
	private Endereco endereco;
	//dados profissionais
	private String emailProfissional;
	private String nuFoneProfissional;
	private String nmEmpresa;
	private Arquivo arquivo;

	public Pessoa(){
		this.arquivo = new Arquivo();
		this.endereco = new Endereco();
	}

	public Pessoa(String nome, Date dtNasc, String nuCPF, String email,
String fone){
		this.nome = nome;
		this.dtNascimento = dtNasc;
		this.nuCPF = nuCPF;
		this.emailPessoal = email;
		this.nuFonePessoal = fone;
		this.arquivo = new Arquivo();
		this.endereco = new Endereco();
	}
	public Date getDtNascimento() {
		return dtNascimento;
	}
	public void setDtNascimento(Date dtNascimento) {
		this.dtNascimento = dtNascimento;
	}
	public String getNome() {
		return nome;
	}
	public void setNome(String nome) {
		this.nome = nome;
	}
	public String getNuCPF() {
		return nuCPF;
	}
	public void setNuCPF(String nuCPF) {
		this.nuCPF = nuCPF;
	}
	public String getSobrenome() {
		return sobrenome;
	}
	public void setSobrenome(String sobrenome) {
		this.sobrenome = sobrenome;
	}
	public String getEmailPessoal() {
		return emailPessoal;
	}
	public void setEmailPessoal(String emailPessoal) {
		this.emailPessoal = emailPessoal;
	}
	public String getEmailProfissional() {
		return emailProfissional;
```

```
	}
	public void setEmailProfissional(String emailProfissional) {
		this.emailProfissional = emailProfissional;
	}
	public String getNuFonePessoal() {
		return nuFonePessoal;
	}
	public void setNuFonePessoal(String nuFonePessoal) {
		this.nuFonePessoal = nuFonePessoal;
	}
	public String getNuFoneProfissional() {
		return nuFoneProfissional;
	}
	public void setNuFoneProfissional(String nuFoneProfissional) {
		this.nuFoneProfissional = nuFoneProfissional;
	}
	public String getNmEmpresa() {
		return nmEmpresa;
	}
	public void setNmEmpresa(String nmEmpresa) {
		this.nmEmpresa = nmEmpresa;
	}
	public Endereco getEndereco() {
		return endereco;
	}
	public void setEndereco(Endereco endereco) {
		this.endereco = endereco;
	}
	public Arquivo getArquivo() {
		return arquivo;
	}
	public void setArquivo(Arquivo arquivo) {
		this.arquivo = arquivo;
	}
	public String getDataFormatada(){
		SimpleDateFormat dateFormat = new
SimpleDateFormat("dd/MM/yyyy");
		String strData = dateFormat.format(this.getDtNascimento());

		return strData;
	}
	public int compareTo( Object obj ) {
		Pessoa outraPessoa = (Pessoa)obj;
		return this.getNome().compareTo(outraPessoa.getNome());

	}
	public Integer getCdPessoa() {
		return cdPessoa;
	}
	public void setCdPessoa(Integer cdPessoa) {
		this.cdPessoa = cdPessoa;
	}
}
```

Classe PessoaForm

```
package br.com.icefaces.util;
import java.util.EventObject;

import javax.faces.component.UIParameter;
import javax.faces.event.ActionEvent;
```

```
import com.icesoft.faces.component.inputfile.InputFile;
import com.icesoft.faces.webapp.xmlhttp.PersistentFacesState;
import com.icesoft.faces.webapp.xmlhttp.RenderingException;

public class PessoaForm {
      private Pessoa pessoa;
      private GerenciadorPessoas gerenciadorPessoas;
      private GerenciadorEstadoCidade gerenciadoEstadocidade;
      //atributos necessarios para indicar quantos porcento do
      //arquivo já está no servidor...
      private int percent;
      private PersistentFacesState state;
      //atributos que definem quais campos serão mostrados na grid paginada
      private boolean renderNome;
      private boolean renderCPF;
      private boolean renderNasc;
      private boolean renderEmail;
      private boolean renderFone;
      private boolean renderEstado;
      private boolean renderCidade;
      public PessoaForm(){
                  this.pessoa = new Pessoa();
                  this.gerenciadorPessoas = new GerenciadorPessoas();
                  this.gerenciadoEstadocidade = new
GerenciadorEstadoCidade();
                  state = PersistentFacesState.getInstance();
      }
      public void action(ActionEvent event){
             InputFile inputFile =(InputFile) event.getSource();
         //file has been saved
           if (inputFile.getStatus() == InputFile.SAVED) {
                this.getPessoa().getArquivo().setStatus( " Arquivo salvo "
);
           }else
           //invalid file, happens when clicking on upload without
selecting a file, or a file with no contents.
           if (inputFile.getStatus() == InputFile.INVALID) {
             this.getPessoa().getArquivo().setStatus( " Arquivo inválido "
);
           }else
           //file size exceeded the limit
           if (inputFile.getStatus() == InputFile.SIZE_LIMIT_EXCEEDED) {
             this.getPessoa().getArquivo().setStatus( " Arquivo acima do
limite do tamanho " );
           }
           this.pessoa.getArquivo().setArquivoLocal(
inputFile.getFilename() );
           this.percent = 0;
      }

//     Required to bind the InputFile component to its OutputProgress
component.
    public void progress (EventObject event) {
        InputFile inputFileComponent = (InputFile)event.getSource();
        percent = inputFileComponent.getFileInfo().getPercent();
        try {
            if (state != null) {
                state.render();
            }
        } catch (RenderingException ee) {
            System.out.println(ee.getMessage());
```

```
        }
    }
    public String add(){
      gerenciadorPessoas.adicionaPessoa(this.getPessoa());
      return "sucesso";
    }
    public void editarPessoa(ActionEvent event){
      UIParameter paramCdPessoa =
(UIParameter)event.getComponent().findComponent("cdPessoaedit");
      Integer cdPessoa = Integer.parseInt(
paramCdPessoa.getValue().toString() );

      this.setPessoa( gerenciadorPessoas.getPessoa( cdPessoa ) );
    }
    public void excluir(ActionEvent event){
      UIParameter paramCdPessoa =
(UIParameter)event.getComponent().findComponent("cdPessoadel");
      Integer cdPessoa = Integer.parseInt(
paramCdPessoa.getValue().toString() );
      this.gerenciadorPessoas.removaPessoa(cdPessoa);
    }
    public String reiniciaCadastro(){
      this.pessoa = new Pessoa();
      //this.gerenciadoEstadocidade = new GerenciadorEstadoCidade();
      return "novo";
    }
    public Pessoa getPessoa() {
            return pessoa;
      }
      public void setPessoa(Pessoa pessoa) {
            this.pessoa = pessoa;
      }
      public int getPercent() {
            return percent;
      }
      public void setPercent(int percent) {
            this.percent = percent;
      }
      public PersistentFacesState getState() {
            return state;
      }
      public void setState(PersistentFacesState state) {
            this.state = state;
      }
      public GerenciadorPessoas getGerenciadorPessoas() {
            return gerenciadorPessoas;
      }
      public void setGerenciadorPessoas(GerenciadorPessoas
gerenciadorPessoas) {
            this.gerenciadorPessoas = gerenciadorPessoas;
      }
      public GerenciadorEstadoCidade getGerenciadoEstadocidade() {
            return gerenciadoEstadocidade;
      }
      public void setGerenciadoEstadocidade(
                  GerenciadorEstadoCidade gerenciadoEstadocidade) {
            this.gerenciadoEstadocidade = gerenciadoEstadocidade;
      }
      public boolean isRenderCidade() {
            return renderCidade;
      }
```

```
	public void setRenderCidade(boolean renderCidade) {
		this.renderCidade = renderCidade;
	}
	public boolean isRenderCPF() {
		return renderCPF;
	}
	public void setRenderCPF(boolean renderCPF) {
		this.renderCPF = renderCPF;
	}
	public boolean isRenderEmail() {
		return renderEmail;
	}
	public void setRenderEmail(boolean renderEmail) {
		this.renderEmail = renderEmail;
	}
	public boolean isRenderEstado() {
		return renderEstado;
	}
	public void setRenderEstado(boolean renderEstado) {
		this.renderEstado = renderEstado;
	}
	public boolean isRenderFone() {
		return renderFone;
	}
	public void setRenderFone(boolean renderFone) {
		this.renderFone = renderFone;
	}
	public boolean isRenderNasc() {
		return renderNasc;
	}
	public void setRenderNasc(boolean renderNasc) {
		this.renderNasc = renderNasc;
	}
	public boolean isRenderNome() {
		return renderNome;
	}
	public void setRenderNome(boolean renderNome) {
		this.renderNome = renderNome;
	}
}
```

Classe CPFValidator

```
package br.com.util.validator;
import java.util.Vector;
import javax.faces.application.FacesMessage;
import javax.faces.component.UIComponent;
import javax.faces.context.FacesContext;
import javax.faces.validator.Validator;
import javax.faces.validator.ValidatorException;

public class CPFValidator implements Validator{
	private final int tamanhoCpf = 11;
	public void validate(FacesContext context, UIComponent component,
Object value) throws ValidatorException {
		if(value == null){
			return;
		}

		if(!confirmaCPF(value.toString())){
```

```
                  FacesMessage msg = new
FacesMessage(FacesMessage.SEVERITY_ERROR, "ERROR1", "CPF inválido");
                  throw new ValidatorException(msg);
            }
      }
      /* Cálculo do CPF foi retirado da página
http://www2.fundao.pro.br/articles.asp?cod=23 */
      private boolean confirmaCPF (String strCpf ) {
            int     d1, d2;
            int     digito1, digito2, resto;
            int     digitoCPF;
            String  nDigResult;
            d1 = d2 = 0;
            digito1 = digito2 = resto = 0;
            for (int nCount = 1; nCount < strCpf.length() -1; nCount++){

               digitoCPF = Integer.valueOf (strCpf.substring(nCount -1,
nCount)).intValue();
               d1 = d1 + ( 11 - nCount ) * digitoCPF;
               d2 = d2 + ( 12 - nCount ) * digitoCPF;
            }
            resto = (d1 % 11);
            if (resto < 2)
               digito1 = 0;
            else
               digito1 = 11 - resto;
            d2 += 2 * digito1;
            resto = (d2 % 11);
            if (resto < 2)
               digito2 = 0;
            else
               digito2 = 11 - resto;
            String nDigVerific = strCpf.substring (strCpf.length()-2,
strCpf.length());
            nDigResult = String.valueOf(digito1) + String.valueOf(digito2);
            return nDigVerific.equals(nDigResult);
         }
}
```

Classe TabsetBean

```
package com.icesoft.icefaces.tutorial.component.tabset.basic;
import java.util.List;
import java.util.ArrayList;
public class TabsetBean {
    private static final int NUMBER_OF_TABS = 3
    private List tabs = new ArrayList(NUMBER_OF_TABS);
    public TabsetBean() {
        // Loop and add default tabs with generic labels and content
        Tab toAdd;
        for (int i = 0; i < NUMBER_OF_TABS; i++) {
            toAdd = new Tab("Label " + (i+1),
                            "Content " + (i+1));
            tabs.add(toAdd);
        }
    }
    public List getTabs() {
        return tabs;
    }
    public void setTabs(List tabs) {
```

```
        this.tabs = tabs;
    }
}
```

Classe Tab

```
package com.icesoft.icefaces.tutorial.component.tabset.basic;
public class Tab
{
    private String label;
    private String content;
    public Tab() {
        label = "Default Label";
        content = "Default Content";
    }
    public Tab(String label) {
        this.label = label;
        content = "Default Content";
    }
    public Tab(String label, String content) {
        this.label = label;
        this.content = content;
    }
    public String getLabel() {
        return label;
    }
    public void setLabel(String label) {
        this.label = label;
    }
    public String getContent() {
        return content;
    }
    public void setContent(String content) {
        this.content = content;
    }
}
```

Arquivo web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application
2.3//EN" "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
  <display-name>ICEfaces Tutorial: Basic Tabset</display-name>
  <description>Component tutorial for the basics of tabset.</description>
  <context-param>
    <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
    <param-value>server</param-value>
  </context-param>
  <context-param>
    <param-name>javax.faces.application.CONFIG_FILES</param-name>
    <param-value>/WEB-INF/faces-config.xml</param-value>
  </context-param>
  <context-param>
    <param-name>com.sun.faces.validateXml</param-name>
    <param-value>true</param-value>
  </context-param>
  <context-param>
```

```
    <param-name>javax.faces.CONFIG_FILES</param-name>
    <param-value>/WEB-INF/faces-config.xml</param-value>
  </context-param>
  <!-- Faces Servlet -->
  <servlet>
    <servlet-name>Faces Servlet</servlet-name>
    <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet>
    <servlet-name>Persistent Faces Servlet</servlet-name>
    <servlet-
class>com.icesoft.faces.webapp.xmlhttp.PersistentFacesServlet</servlet-
class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet>
    <servlet-name>Blocking Servlet</servlet-name>
    <servlet-
class>com.icesoft.faces.webapp.xmlhttp.BlockingServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet>
    <servlet-name>Faces Servlet_tmp</servlet-name>
    <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
    <load-on-startup>0</load-on-startup>
  </servlet>
  <servlet>
        <servlet-name>uploadServlet</servlet-name>
        <servlet-
class>com.icesoft.faces.component.inputfile.FileUploadServlet</servlet-
class>
        <load-on-startup> 1 </load-on-startup>
  </servlet>
  <!-- Faces Servlet Mapping -->
  <servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>*.faces</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>*.jspx</url-pattern>
  </servlet-mapping>
  <!-- Persistent Faces Servlet Mapping -->
  <servlet-mapping>
    <servlet-name>Persistent Faces Servlet</servlet-name>
    <url-pattern>*.iface</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>Persistent Faces Servlet</servlet-name>
    <url-pattern>/xmlhttp/*</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>Blocking Servlet</servlet-name>
    <url-pattern>/block/*</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
        <servlet-name>uploadServlet</servlet-name>
        <url-pattern>/uploadHtml</url-pattern>
    </servlet-mapping>
  <session-config>
```

```
    <session-timeout>100</session-timeout>
  </session-config>
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

Arquivo faces-config.xml

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE faces-config PUBLIC
        "-//Sun Microsystems, Inc.//DTD JavaServer Faces Config 1.1//EN"
        "http://java.sun.com/dtd/web-facesconfig_1_1.dtd">
<faces-config>
    <application>
        <locale-config>
            <default-locale>en</default-locale>
        </locale-config>
    </application>
      <navigation-rule>
            <from-view-id>/cadastroPessoas.iface</from-view-id>
            <navigation-case>
                  <from-outcome>sucesso</from-outcome>
                  <to-view-id>/gridPessoas.iface</to-view-id>
            </navigation-case>
      </navigation-rule>
      <navigation-rule>
            <from-view-id>/gridPessoas.iface</from-view-id>
            <navigation-case>
                  <from-outcome>novo</from-outcome>
                  <to-view-id>/cadastroPessoas.iface</to-view-id>
            </navigation-case>
      </navigation-rule>
      <navigation-rule>
            <from-view-id>/gridPessoas.iface</from-view-id>
            <navigation-case>
                  <from-outcome>deleta</from-outcome>
                  <to-view-id>/gridPessoas.iface</to-view-id>

            </navigation-case>
      </navigation-rule>
      <navigation-rule>
            <from-view-id>/gridPessoas.iface</from-view-id>
            <navigation-case>
                  <from-outcome>edita</from-outcome>
                  <to-view-id>/cadastroPessoas.iface</to-view-id>
            </navigation-case>
      </navigation-rule>
    <!-- Basic Tabset example bean-->
    <managed-bean>
        <description>
            Backing bean for tabset example.
        </description>
        <managed-bean-name>tabset</managed-bean-name>
        <managed-bean-class>
            com.icesoft.icefaces.tutorial.component.tabset.basic.TabsetBean
        </managed-bean-class>
        <managed-bean-scope>session</managed-bean-scope>
    </managed-bean>
```

```
    <managed-bean>
            <managed-bean-name>pessoaForm</managed-bean-name>
            <managed-bean-class>br.com.icefaces.util.PessoaForm</managed-
bean-class>
            <managed-bean-scope>session</managed-bean-scope>
      </managed-bean>
      <managed-bean>
            <managed-bean-name>autoComplete</managed-bean-name>
            <managed-bean-class>br.com.icefaces.util.Autocomplete</managed-
bean-class>
            <managed-bean-scope>session</managed-bean-scope>
      </managed-bean>
      <managed-bean>
            <managed-bean-name>gerenciadoEstadocidade</managed-bean-name>
            <managed-bean-
class>br.com.icefaces.util.GerenciadorEstadoCidade</managed-bean-class>
            <managed-bean-scope>session</managed-bean-scope>
      </managed-bean>
    <managed-bean>
            <managed-bean-name>pessoa</managed-bean-name>
            <managed-bean-class>br.com.icefaces.util.Pessoa</managed-bean-
class>
            <managed-bean-scope>session</managed-bean-scope>
      </managed-bean>
      <managed-bean>
            <managed-bean-name>arquivo</managed-bean-name>
            <managed-bean-class>br.com.icefaces.util.Arquivo</managed-bean-
class>
            <managed-bean-scope>session</managed-bean-scope>
      </managed-bean>
      <managed-bean>
            <managed-bean-name>cidade</managed-bean-name>
            <managed-bean-class>br.com.icefaces.util.Cidade</managed-bean-
class>
            <managed-bean-scope>session</managed-bean-scope>
      </managed-bean>
      <managed-bean>
            <managed-bean-name>endereco</managed-bean-name>
            <managed-bean-class>br.com.icefaces.util.Endereco</managed-
bean-class>
            <managed-bean-scope>session</managed-bean-scope>
      </managed-bean>
      <managed-bean>
            <managed-bean-name>gerenciadorPessoas</managed-bean-name>
            <managed-bean-
class>br.com.icefaces.util.GerenciadorPessoas</managed-bean-class>
            <managed-bean-scope>session</managed-bean-scope>
      </managed-bean>
      <managed-bean>
            <managed-bean-name>estado</managed-bean-name>
            <managed-bean-class>br.com.icefaces.util.Estado</managed-bean-
class>
            <managed-bean-scope>session</managed-bean-scope>
      </managed-bean>
      <validator>
            <validator-id>br.com.util.validator.CPFValidator</validator-id>
            <validator-class>br.com.util.validator.CPFValidator</validator-
class>
      </validator>
</faces-config>
```

Arquivo index.jsp

```
<html>
    <head>
        <title>Tabset Component Tutorial</title>
    </head>
    <body>
        <jsp:forward page="cadastroPessoas.iface" />
    </body>
</html>
```

Arquivo cadastroPessoas.jspx

```
<f:view xmlns:h="http://java.sun.com/jsf/html"
    xmlns:f="http://java.sun.com/jsf/core"
    xmlns:ice="http://www.icesoft.com/icefaces/component">
    <ice:outputDeclaration doctypeRoot="HTML"
        doctypePublic="-//W3C//DTD HTML 4.01 Transitional//EN"
        doctypeSystem="http://www.w3.org/TR/html4/loose.dtd" />
    <html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=iso-
8859-1"></meta>
        <title>Cadastro de pessoas</title>
        <link rel="stylesheet" type="text/css"
href="./xmlhttp/css/xp/xp.css" />
    </head>
    <body>
    <ice:form style="width: 60%;">

    <!--CONEXAO -->
    <ice:outputConnectionStatus />

    <ice:panelTabSet>
    <ice:panelTab label="Dados Gerais">
    <table border="0">
    <tr>
    <td width="100">
            <ice:outputText value="Nome: "/>
    </td>
    <td width="120">
      <ice:selectInputText rows="15" partialSubmit="true"
valueChangeListener="#{autoComplete.updateList}"
value="#{pessoaForm.pessoa.nome}" id="nome">
          <f:selectItems value="#{autoComplete.listaNomes}" />
      </ice:selectInputText>
     </td>
      <td width="200"><h:message for="nome" style="color: red; text-
decoration: overline;text-align: left"/></td>
     </tr>
     <tr>
     <td width="100">
     <ice:outputText value="CPF: "/>
      </td>
      <td width="120">
      <ice:inputText id="cpf" value="#{pessoaForm.pessoa.nuCPF}"
maxlength="11" partialSubmit="true">
      <f:validator validatorId="br.com.util.validator.CPFValidator" />
```

```
      </ice:inputText>
      </td>
      <td width="200"><h:message for="cpf" style="color: red;"/></td>
      </tr>
<tr>
      <td>
      <ice:outputText value="Nascimento: "/>
      </td>
      <td>
      <ice:selectInputDate value="#{pessoaForm.pessoa.dtNascimento}"
partialSubmit="true" renderAsPopup="true"/>
      </td>
      <td></td>
      </tr>
      <tr>
      <td>
      <ice:outputText value="Email pessoal: "/>
      </td>
      <td>
      <ice:inputText value="#{pessoaForm.pessoa.emailPessoal}"/>
      </td>
      <td></td>
      </tr>
      <tr>
      <td>
      <ice:outputText value="Fone residencial: "/>
      </td>
      <td>
      <ice:inputText value="#{pessoaForm.pessoa.nuFonePessoal}"/>
      </td>
      <td></td>
      </tr>
      <tr>
      <td>
      <ice:outputText value="Arquivo: "/>
      </td>
      <td>
<ice:inputFile id="inputFileComponent1"
progressListener="#{pessoaForm.progress}"
actionListener="#{pessoaForm.action}"/>
      <ice:outputProgress label="Status do arquivo"
value="#{pessoaForm.percent}"/>
      <ice:outputText value="#{pessoaForm.pessoa.arquivo.status}" />
      <ice:outputText value="#{pessoaForm.pessoa.arquivo.arquivoLocal}" />

      </td>
      <td></td>
      </tr>

      </table>
      </ice:panelTab>

      <ice:panelTab label="Contato">

      <table width="100%">
      <tr>
      <td width="100">
      <ice:outputText value="Estado:"/>
      </td>
      <td width="*">
```

```
      <ice:selectOneMenu id="titleEstado" style="width:450" tabindex="10"
partialSubmit="true" value="#{pessoaForm.pessoa.endereco.estado.deEstado}"
valueChangeListener="#{gerenciadoEstadocidade.updateList}">
      <f:selectItems value="#{gerenciadoEstadocidade.listaEstados}" />
      </ice:selectOneMenu>
      </td>
      </tr>
      <tr>
      <td width="100"><ice:outputText value="Cidade:"/></td>
      <td width="*">
      <ice:selectOneMenu id="titleCidade" style="width:450"  tabindex="10"
partialSubmit="true" value="#{pessoaForm.pessoa.endereco.cidade.nmCidade}"
size="450">
<f:selectItems value="#{gerenciadoEstadocidade.listaCidades}" />
      </ice:selectOneMenu>
      </td>
      </tr>
      </table>
      </ice:panelTab>
      </ice:panelTabSet>
      <!-- BOTAO -->
      <ice:commandButton  style="font-weight: bold; font-size: 16px"
                                        action="#{pessoaForm.add}"
                                        value="Cadastrar"/>
      </ice:form>
    </body>
    </html>
</f:view>
```

Arquivo gridPessoas.jspx

```
<f:view xmlns:h="http://java.sun.com/jsf/html"
    xmlns:f="http://java.sun.com/jsf/core"
    xmlns:ice="http://www.icesoft.com/icefaces/component">
    <ice:outputDeclaration doctypeRoot="HTML"
        doctypePublic="-//W3C//DTD HTML 4.01 Transitional//EN"
        doctypeSystem="http://www.w3.org/TR/html4/loose.dtd" />

    <html>
    <head>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-
1"> </meta>
    <title>Cadastro de pessoas</title>
    <link rel="stylesheet" type="text/css" href="./xmlhttp/css/xp/xp.css"
/>
    </head>
    <body>
    <ice:form style="width: 45%;">
    <!--CONEXAO -->
    <ice:outputConnectionStatus />
    <br/>
    <table>
    <tr>
    <td width="100">
    <ice:selectBooleanCheckbox value="#{pessoaForm.renderCPF}"
partialSubmit="true" />
    </td>
      <td>
```

```
      <ice:outputText value="CPF"/>
      </td>
      </tr>
      <tr>
      <td width="100">
      <ice:selectBooleanCheckbox value="#{pessoaForm.renderNasc}"
partialSubmit="true" />
      </td>
      <td>
      <ice:outputText value="Data nascimento"/>
      </td>
      </tr>
      <tr>
      <td width="100">
      <ice:selectBooleanCheckbox value="#{pessoaForm.renderEmail}"
partialSubmit="true" checked="true"/>
      </td>
      <td>
      <ice:outputText value="Email"/>
      </td>
      </tr>
      <tr>
      <td width="100">
      <ice:selectBooleanCheckbox value="#{pessoaForm.renderFone}"
partialSubmit="true" />
      </td>
      <td>
      <ice:outputText value="Fone"/>
      </td>
      </tr>
      <tr>
      <td width="100">
      <ice:selectBooleanCheckbox value="#{pessoaForm.renderEstado}"
partialSubmit="true" checked="true"/>
      </td>
      <td>
      <ice:outputText value="Estado"/>
      </td>
      </tr>
      <tr>
      <td width="100">
      <ice:selectBooleanCheckbox value="#{pessoaForm.renderCidade}"
partialSubmit="true" checked="true" />
      </td>
      <td>
      <ice:outputText value="Cidade"/>
      </td>
      </tr>
      </table>
      <ice:panelTab label="Dados Profissionais">
      <!-- Paginator with page controls -->
      <ice:dataPaginator id="dataScroll_3"
for="listPessoaOrdenada" paginator="true" fastStep="3"
paginatorMaxPages="4">
      <f:facet name="first">
      <ice:graphicImage url="./xmlhttp/css/xp/css-images/arrow-first.gif"
style="border:none;" title="First Page"/>
      </f:facet>
      <f:facet name="last">
      <ice:graphicImage url="./xmlhttp/css/xp/css-images/arrow-last.gif"
style="border:none;" title="Last Page"/>
```

```
      </f:facet>
      <f:facet name="previous">
      <ice:graphicImage url="./xmlhttp/css/xp/css-images/arrow-
previous.gif" style="border:none;" title="Previous Page"/>
      </f:facet>
      <f:facet name="next">
      <ice:graphicImage url="./xmlhttp/css/xp/css-images/arrow-next.gif"
style="border:none;" title="Next Page"/>
      </f:facet>
      <f:facet name="fastforward">
      <ice:graphicImage url="./xmlhttp/css/xp/css-images/arrow-ff.gif"
style="border:none;" title="Fast Forward"/>
      </f:facet>
      <f:facet name="fastrewind">
<ice:graphicImage url="./xmlhttp/css/xp/css-images/arrow-fr.gif"
style="border:none;" title="Fast Backwards"/>
</f:facet>
      </ice:dataPaginator>
      <ice:dataTable rows="5"  id="listPessoaOrdenada"
value="#{pessoaForm.gerenciadorPessoas.listPessoaOrdenada}"
var="item">
      <ice:column visible="false">
      <ice:outputText value="#{item.cdPessoa}" visible="false"/>
      </ice:column>
      <!-- NOME -->
      <ice:column width="200">
      <f:facet name="header">
      <ice:outputText value="Nome "/>
      </f:facet>
      <ice:outputText value="#{item.nome}"/>
      </ice:column>
      <!-- CPF -->
      <ice:column rendered="#{pessoaForm.renderCPF}" width="140">
      <f:facet name="header">
      <ice:outputText value="CPF "/>
      </f:facet>
      <ice:outputText value="#{item.nuCPF}"/>
      </ice:column>
      <!-- Nascimento -->
      <ice:column rendered="#{pessoaForm.renderNasc}" width="80">
      <f:facet name="header">
      <ice:outputText value="Data nascimento"/>
      </f:facet>
      <ice:outputText value="#{item.dataFormatada}"/>
      </ice:column>
      <!-- EMAIL -->
      <ice:column rendered="#{pessoaForm.renderEmail}" width="150">
      <f:facet name="header">
      <ice:outputText value="Email"/>
      </f:facet>
      <ice:outputText value="#{item.emailPessoal}"/>
      </ice:column>
      <!-- FONE -->
      <ice:column rendered="#{pessoaForm.renderFone}" width="80">
      <f:facet name="header">
      <ice:outputText value="Fone"/>
      </f:facet>
      <ice:outputText value="#{item.nuFonePessoal}"/>
      </ice:column>
      <!-- ESTADO -->
      <ice:column rendered="#{pessoaForm.renderEstado}" width="130">
```

```
      <f:facet name="header">
      <ice:outputText value="Estado"/>
      </f:facet>
      <ice:outputText value="#{item.endereco.estado.deEstado}"/>
      </ice:column>
      <!-- CIDADE -->
      <ice:column rendered="#{pessoaForm.renderCidade}" width="130">
      <f:facet name="header">
      <ice:outputText value="Cidade"/>
      </f:facet>
      <ice:outputText value="#{item.endereco.cidade.nmCidade}"/>
      </ice:column>
      <ice:column>
      <f:facet name="header">
      <ice:outputText value=""/>
      </f:facet>
      <ice:commandButton  style="font-weight: bold; font-size: 16px"
      actionListener="#{pessoaForm.editarPessoa}" value="Editar"
action="edita">
      <f:param id="cdPessoaedit" name="cdPessoa" value="#{item.cdPessoa}"/>
      </ice:commandButton>
      </ice:column>
      <ice:column>
      <f:facet name="header">
      <ice:outputText value=""/>
      </f:facet>
      <ice:commandButton  style="font-weight: bold; font-size: 16px"
      actionListener="#{pessoaForm.excluir}" value="Deletar"
action="deleta">
      <f:param id="cdPessoadel" name="cdPessoa" value="#{item.cdPessoa}"/>
      </ice:commandButton>
      </ice:column>
      </ice:dataTable>
      <ice:dataPaginator id="dataScroll_2" for="listPessoaOrdenada"
rowsCountVar="rowsCount" displayedRowsCountVar="displayedRowsCount"
firstRowIndexVar="firstRowIndex" lastRowIndexVar="lastRowIndex"
pageCountVar="pageCount" pageIndexVar="pageIndex">
      <ice:outputFormat value="{0} pessoas cadastradas, Exibindo registros
de {1} ate {2}. Exibindo pagina  {3} de {4}." styleClass="standard">
      <f:param value="#{rowsCount}"/>
      <f:param value="#{firstRowIndex}"/>
      <f:param value="#{lastRowIndex}"/>
      <f:param value="#{pageIndex}"/>
      <f:param value="#{pageCount}"/>
      </ice:outputFormat>
      </ice:dataPaginator>
      </ice:panelTab>
      <ice:commandButton  style="font-weight: bold; font-size: 16px"
action="#{pessoaForm.reiniciaCadastro}" value="Novo"/>
      </ice:form>
      </body>
      </html>
</f:view>
```

10.4 Exemplo AJAX4JSF

Classe Pessoa

```
package br.com.tcc.ajax4jsf.util;
import java.text.SimpleDateFormat;
import java.util.Collection;
import java.util.Date;
import java.util.List;
public class Pessoa implements Comparable{

	private Integer cdPessoa;
	//dados pessoais
	private String nome;
	private String sobrenome;
	private Date dtNascimento;
	private String nuCPF;
	private String emailPessoal;
	private String nuFonePessoal;
	private Endereco endereco;
	//dados profissionais
	private String emailProfissional;
	private String nuFoneProfissional;
	private String nmEmpresa;

	public Pessoa(){
		this.endereco = new Endereco();
	}
	public Pessoa(String nome){
		this.nome = nome;
	}
	public Pessoa(String nome, Date dtNasc, String nuCPF, String email,
String fone){
		this.nome = nome;
		this.dtNascimento = dtNasc;
		this.nuCPF = nuCPF;
		this.emailPessoal = email;
		this.nuFonePessoal = fone;
		this.endereco = new Endereco();
	}
	public Date getDtNascimento() {
		return dtNascimento;
	}
	public void setDtNascimento(Date dtNascimento) {
		this.dtNascimento = dtNascimento;
	}
	public String getNome() {
		return nome;
	}
	public void setNome(String nome) {
		this.nome = nome;
	}
	public String getNuCPF() {
		return nuCPF;
	}
	public void setNuCPF(String nuCPF) {
		this.nuCPF = nuCPF;
	}
	public String getSobrenome() {
		return sobrenome;
	}
```

```
	public void setSobrenome(String sobrenome) {
		this.sobrenome = sobrenome;
	}
	public String getEmailPessoal() {
		return emailPessoal;
	}
	public void setEmailPessoal(String emailPessoal) {
		this.emailPessoal = emailPessoal;
	}
	public String getEmailProfissional() {
		return emailProfissional;
	}
	public void setEmailProfissional(String emailProfissional) {
		this.emailProfissional = emailProfissional;
	}
	public String getNuFonePessoal() {
		return nuFonePessoal;
	}
	public void setNuFonePessoal(String nuFonePessoal) {
		this.nuFonePessoal = nuFonePessoal;
	}
	public String getNuFoneProfissional() {
		return nuFoneProfissional;
	}
	public void setNuFoneProfissional(String nuFoneProfissional) {
		this.nuFoneProfissional = nuFoneProfissional;
	}
	public String getNmEmpresa() {
		return nmEmpresa;
	}
	public void setNmEmpresa(String nmEmpresa) {
		this.nmEmpresa = nmEmpresa;
	}
	public Endereco getEndereco() {
		return endereco;
	}
	public void setEndereco(Endereco endereco) {
		this.endereco = endereco;
	}
	public String getDataFormatada(){
		SimpleDateFormat dateFormat = new
SimpleDateFormat("dd/MM/yyyy");
		String strData = dateFormat.format(this.getDtNascimento());

		return strData;
	}
	public int compareTo( Object obj ) {
		Pessoa outraPessoa = (Pessoa)obj;
		return this.getNome().compareTo(outraPessoa.getNome());

	}
	public Integer getCdPessoa() {
		return cdPessoa;
	}
	public void setCdPessoa(Integer cdPessoa) {
		this.cdPessoa = cdPessoa;
	}
}
```

Classe PessoaForm

```
package br.com.tcc.ajax4jsf.util;

import java.util.List;

import javax.faces.component.html.HtmlInputText;
import javax.faces.event.ActionEvent;

import org.ajax4jsf.ajax.html.HtmlAjaxSupport;

import br.com.tcc.util.validator.CPFValidator;

import com.exadel.vcp.components.ajax.SuggestionEvent;

public class PessoaForm {
	private Pessoa pessoa;
	private GerenciadorEstadoCidade gerenciadorEstadoCidade;
	private String mensagemCpfErro;
	private String mensagemEmailErro;
	private CPFValidator cpfvalidator;


	public PessoaForm(){
			this.pessoa = new Pessoa();
			this.gerenciadorEstadoCidade = new
GerenciadorEstadoCidade();
			this.cpfvalidator = new CPFValidator();
	}

	public void validaCPF(ActionEvent input){

		HtmlAjaxSupport value = (HtmlAjaxSupport)input.getSource();
		HtmlInputText inputText = (HtmlInputText)value.getParent();

		CPFValidator validator = new CPFValidator();
		if(!validator.confirmaCPF((String)inputText.getValue())){
			this.mensagemCpfErro = "CPF inválido";
		}else{
			this.mensagemCpfErro = "";
		}

	}

	public void validaEmail(ActionEvent input){

		HtmlAjaxSupport value = (HtmlAjaxSupport)input.getSource();
		HtmlInputText inputText = (HtmlInputText)value.getParent();

		String emailDigitado = (String)inputText.getValue();

		if(!emailDigitado.contains("@")){
			this.mensagemEmailErro = "Email deve conter o caracter
'@'";
		}else{
			this.mensagemEmailErro = "";
		}

	}


	public Pessoa getPessoa() {
		return pessoa;
```

```
	}
	public void setPessoa(Pessoa pessoa) {
		this.pessoa = pessoa;
	}
	public GerenciadorEstadoCidade getGerenciadorEstadoCidade() {
		return gerenciadorEstadoCidade;
	}
	public void setGerenciadorEstadoCidade(
			GerenciadorEstadoCidade gerenciadorEstadoCidade) {
		this.gerenciadorEstadoCidade = gerenciadorEstadoCidade;
	}
	public String getMensagemCpfErro() {
		return mensagemCpfErro;
	}
	public void setCpfErro(String cpfErro) {
		this.mensagemCpfErro = cpfErro;
	}

	public CPFValidator getCpfvalidator() {
		return cpfvalidator;
	}

	public void setCpfvalidator(CPFValidator cpfvalidator) {
		this.cpfvalidator = cpfvalidator;
	}

	public String getMensagemEmailErro() {
		return mensagemEmailErro;
	}

	public void setMensagemEmailErro(String mensagemEmailErro) {
		this.mensagemEmailErro = mensagemEmailErro;
	}

	public void setMensagemCpfErro(String mensagemCpfErro) {
		this.mensagemCpfErro = mensagemCpfErro;
	}
}
```

Classe GerenciadorEstadoCidade

```
package br.com.tcc.ajax4jsf.util;
import java.util.ArrayList;
import java.util.List;
import javax.faces.component.html.HtmlSelectOneMenu;
import javax.faces.event.ActionEvent;
import javax.faces.model.SelectItem;
import org.ajax4jsf.ajax.html.HtmlAjaxSupport;
public class GerenciadorEstadoCidade {
	private List<SelectItem> listaEstados;
	private List<SelectItem> listaCidades;
	public GerenciadorEstadoCidade(){
		listaEstados = new ArrayList<SelectItem>();
		listaCidades = new ArrayList<SelectItem>();
		this.populeListaEstados();
	}
	private void populeListaEstados(){
		listaEstados.add(new SelectItem("Paraná"));
		listaEstados.add(new SelectItem("Santa Catarina"));
		listaEstados.add(new SelectItem("Rio Grande do Sul"));

	}
```

```
	public void updateList(ActionEvent event){
		HtmlAjaxSupport ajaxSupport =
(HtmlAjaxSupport)event.getSource();
		if(ajaxSupport.getParent() instanceof HtmlSelectOneMenu ){
			listaCidades.clear();
			HtmlSelectOneMenu selectOneMenu =
(HtmlSelectOneMenu)ajaxSupport.getParent();
			String valorSelecionado =
(String)selectOneMenu.getValue();
			if("Santa Catarina".equalsIgnoreCase(valorSelecionado)){
				System.out.println("Selecionado Santa Catarina");
				listaCidades.add(new SelectItem("Florianópolis"));

				listaCidades.add(new SelectItem("Blumenau"));
				listaCidades.add(new SelectItem("Chapecó"));
				listaCidades.add(new SelectItem("Joinvile"));
				listaCidades.add(new SelectItem("Xavantina"));

			}else if("Paraná".equalsIgnoreCase(valorSelecionado)){
				System.out.println("Selecionado Paraná");
				listaCidades.add(new SelectItem("Curitiba"));
				listaCidades.add(new SelectItem("Londrina"));
				listaCidades.add(new SelectItem("Maringá"));
				listaCidades.add(new SelectItem("São José dos
Pinhais"));
			}else if("Rio Grande do
Sul".equalsIgnoreCase(valorSelecionado)){
				System.out.println("Selecionado Rio Grande do
Sul");
				listaCidades.add(new SelectItem("Porto Alegre"));
				listaCidades.add(new SelectItem("Alegrete"));
				listaCidades.add(new SelectItem("Bento
Gonçalves"));
				listaCidades.add(new SelectItem("Caxias"));
				listaCidades.add(new SelectItem("Cruz Alta"));

			}else{
				System.out.println("Nenhum selecionado");
			}
		}
	}
	public List<SelectItem> getListaEstados() {
		return listaEstados;
	}
	public void setListaEstados(List<SelectItem> listaEstados) {
		this.listaEstados = listaEstados;
	}
	public List<SelectItem> getListaCidades() {
		return listaCidades;
	}
	public void setListaCidades(List<SelectItem> listaCidades) {
		this.listaCidades = listaCidades;
	}
}
```

Classe Estado

```
package br.com.tcc.ajax4jsf.util;
public class Estado {
	private Integer cdEstado;
```

```
	private String deEstado;
	public Integer getCdEstado() {
		return cdEstado;
	}
	public void setCdEstado(Integer cdEstado) {
		this.cdEstado = cdEstado;
	}
	public String getDeEstado() {
		return deEstado;
	}
	public void setDeEstado(String deEstado) {
		this.deEstado = deEstado;
	}
}
```

Classe Endereco

```
package br.com.tcc.ajax4jsf.util;
public class Endereco {
	private String nmRua;
	private Integer nuResidencia;
	private String deComplemento;
	private String nmBairro;
	private Cidade cidade;
	private Estado estado;
	public Endereco(){
		this.cidade = new Cidade();
		this.estado = new Estado();
	}
	public Cidade getCidade() {
		return cidade;
	}
	public void setCidade(Cidade cidade) {
		this.cidade = cidade;
	}
	public String getDeComplemento() {
		return deComplemento;
	}
	public void setDeComplemento(String deComplemento) {
		this.deComplemento = deComplemento;
	}
	public String getNmBairro() {
		return nmBairro;
	}
	public void setNmBairro(String nmBairro) {
		this.nmBairro = nmBairro;
	}
	public String getNmRua() {
		return nmRua;
	}
	public void setNmRua(String nmRua) {
		this.nmRua = nmRua;
	}
	public Integer getNuResidencia() {
		return nuResidencia;
	}
	public void setNuResidencia(Integer nuResidencia) {
		this.nuResidencia = nuResidencia;
	}
	public Estado getEstado() {
```

```
		return estado;
	}
	public void setEstado(Estado estado) {
		this.estado = estado;
	}
}
```

Classe Cidade

```
package br.com.tcc.ajax4jsf.util;
public class Cidade {
	private Integer cdCidade;
	private String nmCidade;
	public Integer getCdCidade() {
		return cdCidade;
	}
	public void setCdCidade(Integer cdCidade) {
		this.cdCidade = cdCidade;
	}
	public String getNmCidade() {
		return nmCidade;
	}
	public void setNmCidade(String nmCidade) {
		this.nmCidade = nmCidade;
	}
}
```

Classe Autocomplete

```
package br.com.tcc.ajax4jsf.util;
import java.util.ArrayList;
import java.util.List;
import javax.faces.event.ValueChangeEvent;
import javax.faces.model.SelectItem;
import com.exadel.vcp.components.ajax.SuggestionEvent;
public class Autocomplete {
	private List<String> listaNomes;
	private String nomeAtual;
	private List<String> listaNomesFixo;
	public Autocomplete(){
		listaNomes = new ArrayList<String>();
		listaNomesFixo = new ArrayList<String>();
		nomeAtual = "";
		this.populeListaNomes();
		listaNomesFixo.addAll(listaNomes);
	}
	private void populeListaNomes(){
		listaNomes.add("Anderson");
		listaNomes.add("Andrey");
		listaNomes.add("Augusto");
		listaNomes.add("Bruno");
		listaNomes.add("Caio");
		listaNomes.add("Denise");
		listaNomes.add("Diego");
		listaNomes.add("Douglas");
		listaNomes.add("Eder");
		listaNomes.add("Edson");
		listaNomes.add("Eduardo");
		listaNomes.add("Felipe");
		listaNomes.add("Fernando");
		listaNomes.add("Gabriel");
```

```
            listaNomes.add("Gilberto");
            listaNomes.add("Guilherme");
            listaNomes.add("Jeronimo");
            listaNomes.add("Joao");
            listaNomes.add("Leonardo");
            listaNomes.add("Luiz");
            listaNomes.add("Marcelo");
            listaNomes.add("Martín");
            listaNomes.add("Maximiliano");
            listaNomes.add("Paulo");
            listaNomes.add("Rafael");
            listaNomes.add("Sabrina");
            listaNomes.add("Tiago");
            listaNomes.add("Thiago");
            listaNomes.add("Tie");
            listaNomes.add("Victor");
      }
      public List getListaNomes() {
            if(listaNomes == null){
                  listaNomes = new ArrayList();
            }
            return listaNomes;
      }
      public void setListaNomes(List listaNomes) {
            this.listaNomes = listaNomes;
      }
      public List<String> suggestionAction(SuggestionEvent event) {
          //return makeSuggestion(event.getSubmittedValue());
            listaNomes = new ArrayList<String>();

            String valorDigitado = event.getSubmittedValue();

            int tamanhoListaNomes = listaNomesFixo.size();
            String valorAtual;
            for(int i=0; i<tamanhoListaNomes;i++){
                  valorAtual = listaNomesFixo.get(i);

      if(valorAtual.toUpperCase().contains(valorDigitado.toUpperCase())){
                        listaNomes.add(valorAtual);
                  }
            }
            return listaNomes;
      }
      public String getNomeAtual() {
            return nomeAtual;
      }
      public void setNomeAtual(String nomeAtual) {
            this.nomeAtual = nomeAtual;
      }
      public List getListaNomesFixo() {
            return listaNomesFixo;
      }
      public void setListaNomesFixo(List listaNomesFixo) {
            this.listaNomesFixo = listaNomesFixo;
      }
}
```

Classe Autocomplete

```
package br.com.tcc.util.validator;
```

```
public class CPFValidator{
      /* Cálculo do CPF foi retirado da página
http://www2.fundao.pro.br/articles.asp?cod=23 */
      public boolean confirmaCPF (String strCpf ) {
            int     d1, d2;
            int     digito1, digito2, resto;
            int     digitoCPF;
            String  nDigResult;
            d1 = d2 = 0;
            digito1 = digito2 = resto = 0;
            for (int nCount = 1; nCount < strCpf.length() -1; nCount++){

               digitoCPF = Integer.valueOf (strCpf.substring(nCount -1,
nCount)).intValue();
               d1 = d1 + ( 11 - nCount ) * digitoCPF;
               d2 = d2 + ( 12 - nCount ) * digitoCPF;
            }
            resto = (d1 % 11);
            if (resto < 2)
               digito1 = 0;
            else
               digito1 = 11 - resto;
            d2 += 2 * digito1;
            resto = (d2 % 11);
            if (resto < 2)
               digito2 = 0;
            else
               digito2 = 11 - resto;
            String nDigVerific = strCpf.substring (strCpf.length()-2,
strCpf.length());
            nDigResult = String.valueOf(digito1) + String.valueOf(digito2);
            return nDigVerific.equals(nDigResult);
         }
}
```

Classe TabPanel

```
package com.exadel.vcp.demosite.demos.tabpanel;
/*
 Classe baseada no exemplo apresentado no site do RichFaces
 http://livedemo.exadel.com/richfaces-demo/products/vcp/tab/index.jsf;
            jsessionid=7359DD43AF358B4DD1488CA65BE67BFD#faces_beans_xml
*/
import java.util.ArrayList;
import java.util.List;
import javax.faces.component.UIComponent;
import com.exadel.vcp.components.UITabPanelPane;
public class TabPanel {
  private UIComponent tabPanel;
  private static final String NONE = "none";
  private String disabledTabName = NONE;
  public UIComponent getTabPanel() {
    return tabPanel;
  }
  public void setTabPanel(UIComponent tabPanel) {
    this.tabPanel = tabPanel;
  }
  public String getDisabledTabName() {
    return disabledTabName;
  }
  public void setDisabledTabName(String disabledTabId) {
    this.disabledTabName = disabledTabId;
```

```
  }
  public String disableTab() {
    UITabPanelPane tabPanel = (UITabPanelPane) getTabPanel();
    Object value = tabPanel.getValue();
    if (value != null && value.equals(disabledTabName)) {
      List<String> tabNames = getTabNames();
      int idx = tabNames.indexOf(value);
      int i = idx + 1;
      if (i >= tabNames.size()) {
        if (!tabNames.isEmpty()) {
          tabPanel.setValue(tabNames.get(0));
        }
      } else {
        tabPanel.setValue(tabNames.get(i));
      }
    }
     return null;
  }

  private static final List<String> tabNames = new ArrayList<String>();
   static {
    tabNames.add("dados");
    tabNames.add("contato");
  }
   public List<String> getTabNames() {
    return tabNames;
  }
}
```

Arquivo index.jsp

```
<%@ taglib uri="https://ajax4jsf.dev.java.net/ajax" prefix="a4j"%>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<%@ taglib uri="http://www.exadel.com/vcp/components" prefix="vcp"%>
<%@ taglib uri="http://www.exadel.com/vcp/components/ajax" prefix="ajax"%>

<%@ include file="./layout/cabecalhofamilia.jsp" %>

<html>
<head>
<title>Index</title>
</head>
<body bgcolor="white">
<f:view>
<f:verbatim>
</f:verbatim>
      <h:form>
      <vcp:tabpanel width="100%" switchType="client"
      binding="#{tabpanel.tabPanel}" id="tab_panel">

      <vcp:tab disabled="#{tabpanel.disabledTabName == 'dados'}"
      name="dados" label="Dados Gerais">
      <a4j:region id="regPessoa">
      <a4j:status for="regPessoa">
      <f:facet name="start">
      <h:graphicImage value="/images/ajax_process.gif" />
      </f:facet>
      <f:facet name="stop">
      <h:graphicImage value="/images/ajax_stoped.gif" />
      </f:facet>
      </a4j:status>
```

```
      <h:panelGroup style="display:block">
      <h:panelGrid columns="2">
      <h:outputText value="Olá: " style="width:125"/>
      <h:outputText value="#{pessoaForm.pessoa.nome}"  id="nome"
style="width:150"/>
      </h:panelGrid>
      <h:panelGrid columns="3">
      <h:outputText value="Nome: " style="width:125"/>
      <h:inputText value="#{pessoaForm.pessoa.nome}" id="inputNome"
style="width:200">
      <a4j:support event="onkeyup" reRender="nome"/>
      </h:inputText>
      <ajax:suggestionBox id="suggestionBox1" for="inputNome"
var="component" suggestionAction="#{autocomplete.suggestionAction}"
minChars="1" frame="box" frequency="0" width="300" rendered="true"
selfRendered="true" fetchValue="#{autocomplete.listaNomes}" cellspacing="0"
cellpadding="1" style="width:200">
      <h:column>
      <h:outputText value="#{component}"/>
      </h:column>
      </ajax:suggestionBox>
      </h:panelGrid>
      <h:panelGrid columns="3">
      <h:outputText value="CPF: " style="width:125"/>
      <h:inputText value="#{pessoaForm.pessoa.nuCPF}">
      <a4j:support actionListener="#{pessoaForm.validaCPF}" event="onblur"
reRender="msg"/>
      </h:inputText>
      <h:outputText id="msg" value="#{pessoaForm.mensagemCpfErro}" />
      </h:panelGrid>
      <h:panelGrid columns="3">
      <h:outputText value="Email: " style="width:125"/>
      <h:inputText value="#{pessoaForm.pessoa.emailPessoal}">
      <a4j:support actionListener="#{pessoaForm.validaEmail}"
event="onblur" reRender="msgEmail"/>
      </h:inputText>
      <h:outputText id="msgEmail" value="#{pessoaForm.mensagemEmailErro}"
/>
      </h:panelGrid>
      <h:panelGrid columns="2">
      <h:outputText value="Telefone: " style="width:125"/>
      <h:inputText value="#{pessoaForm.pessoa.nuFonePessoal}"/>
      <h:outputText value="Data aniversário: " style="width:125"/>
      <h:outputText value="#{pessoaForm.pessoa.dtNascimento}" id="data"/>
      </h:panelGrid>
      <vcp:calendar value="#{pessoaForm.pessoa.dtNascimento}"
id="calendar">
      <a4j:support event="onclick" reRender="data"/>
      </vcp:calendar>
      </h:panelGroup>
      </a4j:region>
      </vcp:tab>
      <vcp:tab disabled="#{tabpanel.disabledTabName == 'contato'}"
name="contato" label="Contato">
      <a4j:region id="regEndereco">
      <h:panelGroup style="display:block" >
      <h:outputText value="Estado : " />
      <h:selectOneMenu
value="#{pessoaForm.pessoa.endereco.estado.deEstado}" id="selectEstado">
```

```
      <f:selectItems
value="#{pessoaForm.gerenciadorEstadoCidade.listaEstados}"/>
      <a4j:support event="onchange"
actionListener="#{pessoaForm.gerenciadorEstadoCidade.updateList}"
ajaxSingle="true" reRender="selectCidade" id="listaEstados">
      </a4j:support>
      </h:selectOneMenu>
      </h:panelGroup>
      <h:panelGroup style="display:block" >
      <h:outputText value="Cidade : "/>
      <h:selectOneMenu
value="#{pessoaForm.pessoa.endereco.cidade.nmCidade}" id="selectCidade">
      <f:selectItems
value="#{pessoaForm.gerenciadorEstadoCidade.listaCidades}"
id="listaCidades"/>
      </h:selectOneMenu>
      </h:panelGroup>
      </a4j:region>
      </vcp:tab>
      </vcp:tabpanel>
      </h:form>
</f:view>
</body>
</html>
<%@ include file="./layout/rodape.htm" %>
```

Arquivo start.jsp

```
<html>
  <head>
    <title>My JSP 'start.jsp' starting page</title>
  </head>
  <body>
      <jsp:forward page="index.jsf"/>
  </body>
</html>
```

Arquivo faces-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE faces-config PUBLIC "-//Sun Microsystems, Inc.//DTD JavaServer
Faces Config 1.1//EN" "http://java.sun.com/dtd/web-facesconfig_1_1.dtd">

<faces-config>
      <managed-bean>
      <managed-bean-name>pessoa</managed-bean-name>
      <managed-bean-class>br.com.tcc.ajax4jsf.util.Pessoa</managed-bean-
class>
      <managed-bean-scope>session</managed-bean-scope>
      </managed-bean>
      <managed-bean>
            <managed-bean-name>cidade</managed-bean-name>
            <managed-bean-class>br.com.tcc.ajax4jsf.util.Cidade</managed-
bean-class>
            <managed-bean-scope>session</managed-bean-scope>
      </managed-bean>
      <managed-bean>
            <managed-bean-name>endereco</managed-bean-name>
            <managed-bean-class>br.com.tcc.ajax4jsf.util.Endereco</managed-
bean-class>
            <managed-bean-scope>session</managed-bean-scope>
```

```
	</managed-bean>
	<managed-bean>
		<managed-bean-name>estado</managed-bean-name>
		<managed-bean-class>br.com.tcc.ajax4jsf.util.Estado</managed-
bean-class>
		<managed-bean-scope>session</managed-bean-scope>
	</managed-bean>
	<managed-bean>
		<managed-bean-name>gerenciadorEstadoCidade</managed-bean-name>
		<managed-bean-
class>br.com.tcc.ajax4jsf.util.GerenciadorEstadoCidade</managed-bean-class>
		<managed-bean-scope>session</managed-bean-scope>
	</managed-bean>
	<managed-bean>
		<managed-bean-name>autocomplete</managed-bean-name>
		<managed-bean-
class>br.com.tcc.ajax4jsf.util.Autocomplete</managed-bean-class>
		<managed-bean-scope>session</managed-bean-scope>
	</managed-bean>
	<managed-bean>
		<managed-bean-name>pessoaForm</managed-bean-name>
		<managed-bean-
class>br.com.tcc.ajax4jsf.util.PessoaForm</managed-bean-class>
		<managed-bean-scope>session</managed-bean-scope>
	</managed-bean>
	<managed-bean>
		<managed-bean-name>tabpanel</managed-bean-name>
		<managed-bean-
class>com.exadel.vcp.demosite.demos.tabpanel.TabPanel</managed-bean-class>
		<managed-bean-scope>session</managed-bean-scope>
	</managed-bean>
	<managed-bean>
		<managed-bean-name>CPFValidator</managed-bean-name>
		<managed-bean-
class>br.com.tcc.util.validator.CPFValidator</managed-bean-class>
		<managed-bean-scope>session</managed-bean-scope>
	</managed-bean>
</faces-config>
```

Arquivo web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
 <display-name>a4j-simpleRepeator</display-name>
 <context-param>
  <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
  <param-value>server</param-value>
 </context-param>
 <context-param>
    <param-name>org.ajax4jsf.SKIN</param-name>
    <param-value>classic</param-value>
</context-param>
<context-param>
 <param-name>com.exadel.vcp.trialKey</param-name>
 <param-value>.jssxbgRjYHCth-zuiE0Mg__</param-value>
</context-param>
```

```
<filter>
    <display-name>Ajax4jsf Filter</display-name>
    <filter-name>ajax4jsf</filter-name>
    <filter-class>org.ajax4jsf.Filter</filter-class>
</filter>
<filter-mapping>
    <filter-name>ajax4jsf</filter-name>
    <servlet-name>Faces Servlet</servlet-name>
    <dispatcher>REQUEST</dispatcher>
    <dispatcher>FORWARD</dispatcher>
    <dispatcher>INCLUDE</dispatcher>
</filter-mapping>
 <listener>
  <listener-class>com.sun.faces.config.ConfigureListener</listener-class>
 </listener>
 <servlet>
  <servlet-name>Faces Servlet</servlet-name>
  <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
  <load-on-startup>1</load-on-startup>
 </servlet>
 <servlet-mapping>
  <servlet-name>Faces Servlet</servlet-name>
  <url-pattern>*.jsf</url-pattern>
 </servlet-mapping>
 <welcome-file-list>
  <welcome-file>start.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

10.5 Exemplo Backbase

Classe Pessoa

```
package br.com.tcc.backbase.util;
import java.io.File;
import java.util.Date;
import com.backbase.bjs.component.bxml.UIBackbaseDatePicker;
public class Pessoa {
      private String nome;
      private String email;
      private UIBackbaseDatePicker dtNascimento;
      private File arquivo;
      public Pessoa(){
            this.dtNascimento = new UIBackbaseDatePicker();
      }
      public Pessoa(String nome, String email){
            this.nome = nome;
            this.email = email;
      }
      public File getArquivo() {
            return arquivo;
      }
      public void setArquivo(File arquivo) {
            this.arquivo = arquivo;
      }
      public UIBackbaseDatePicker getDtNascimento() {
            return dtNascimento;
      }
      public void setDtNascimento(UIBackbaseDatePicker dtNascimento) {
```

```
            this.dtNascimento = dtNascimento;
      }
      public String getEmail() {
            return email;
      }
      public void setEmail(String email) {
            this.email = email;
      }
      public String getNome() {
            return nome;
      }
      public void setNome(String nome) {
            this.nome = nome;
      }
}
```

Classe PessoaForm

```
package br.com.tcc.backbase.util;
import java.util.ArrayList;
import java.util.List;
import com.backbase.bjs.component.UIBackbaseData;
import com.backbase.bjs.component.bxml.UIBackbasePanelSet;
import com.backbase.bjs.event.SortEvent;

public class PessoaForm {
      private List<Pessoa> listaPessoas;
      private UIBackbaseData listgrid;
      private UIBackbasePanelSet listPanelSet;
      public PessoaForm(){
            this.listaPessoas = new ArrayList<Pessoa>();
            this.populeListaPessoas();
      }
      private void populeListaPessoas(){
            this.listaPessoas.add(new Pessoa("Anderson",
"anderson@inf.ufsc.br"));
            this.listaPessoas.add(new
Pessoa("Andrey","andrey@inf.ufsc.br"));
            this.listaPessoas.add(new
Pessoa("Augusto","augusto@inf.ufsc.br"));
            this.listaPessoas.add(new Pessoa("Bruno","bruno@inf.ufsc.br"));
            this.listaPessoas.add(new Pessoa("Caio","caio@inf.ufsc.br"));
            this.listaPessoas.add(new
Pessoa("Denise","denise@inf.ufsc.br"));
            this.listaPessoas.add(new Pessoa("Diego","diego@inf.ufsc.br"));
            this.listaPessoas.add(new
Pessoa("Douglas","douglas@inf.ufsc.br"));
            this.listaPessoas.add(new Pessoa("Eder","eder@inf.ufsc.br"));
            this.listaPessoas.add(new Pessoa("Edson","edson@inf.ufsc.br"));
            this.listaPessoas.add(new
Pessoa("Eduardo","eduardo@inf.ufsc.br"));
            this.listaPessoas.add(new
Pessoa("Felipe","felipe@inf.ufsc.br"));
            this.listaPessoas.add(new
Pessoa("Fernando","fernando@inf.ufsc.br"));
            this.listaPessoas.add(new
Pessoa("Gabriel","gabriel@inf.ufsc.br"));
            this.listaPessoas.add(new
Pessoa("Gilberto","gilberto@inf.ufsc.br"));
            this.listaPessoas.add(new
```

```
Pessoa("Guilherme","guilherme@inf.ufsc.br"));
            this.listaPessoas.add(new
Pessoa("Jeronimo","jeronimo@inf.ufsc.br"));
            this.listaPessoas.add(new Pessoa("Joao","joao@inf.ufsc.br"));
            this.listaPessoas.add(new
Pessoa("Leonardo","leonardo@inf.ufsc.br"));
            this.listaPessoas.add(new Pessoa("Luiz","luiz@inf.ufsc.br"));
            this.listaPessoas.add(new
Pessoa("Marcelo","marcelo@inf.ufsc.br"));
            this.listaPessoas.add(new
Pessoa("Martín","martin@inf.ufsc.br"));
            this.listaPessoas.add(new
Pessoa("Maximiliano","max@inf.ufsc.br"));
            this.listaPessoas.add(new Pessoa("Paulo","paulo@inf.ufsc.br"));
            this.listaPessoas.add(new
Pessoa("Rafael","rafael@inf.ufsc.br"));
            this.listaPessoas.add(new
Pessoa("Sabrina","sabrina@inf.ufsc.br"));
            this.listaPessoas.add(new Pessoa("Tiago","tiago@inf.ufsc.br"));
            this.listaPessoas.add(new Pessoa("Tie","tie@inf.ufsc.br"));
            this.listaPessoas.add(new
Pessoa("Victor","victor@inf.ufsc.br"));
      }
      public List<Pessoa> getListaPessoas() {
            return listaPessoas;
      }
      public void setListaPessoas(List<Pessoa> listaPessoas) {
            this.listaPessoas = listaPessoas;
      }
      public UIBackbaseData getListgrid() {
            return listgrid;
      }
      public void setListgrid(UIBackbaseData listgrid) {
            this.listgrid = listgrid;
      }
      public UIBackbasePanelSet getListPanelSet() {
            return listPanelSet;
      }
      public void setListPanelSet(UIBackbasePanelSet listPanelSet) {
            this.listPanelSet = listPanelSet;
      }
      public void sort(SortEvent event){
            String coluna = event.getColumnLabel();
            if(coluna.contains("Nome")){
                  System.out.println("ordena por nome");
            }else if(coluna.contains("Email")){
                  System.out.println("ordena por email");
            }
            System.out.println("passou aqui na ordenação");
      }
}
```

Arquivo index.jsp

```
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<%@ taglib uri="http://www.backbase.com/bjs/core" prefix="bjs"%>
<f:view>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:b="http://www.backbase.com/b"
```

```
      xmlns:s="http://www.backbase.com/s">
  <head>
    <bjs:boot var="bootDir" />
  </head>
  <body onload="bpc.boot('<%=bootDir%>')"
b:skinbase="default_without_form.xml">
    <bjs:application skin="system" id="app1">
    <bjs:panelGrid columns="3" label="Cadastro de Pessoas"
headerClass="step-header"
        columnClasses="label,asterix,form-field,messages"
styleClass="grid">
    <bjs:outputText value="Nome:" id="labelnome" />
    <bjs:inputText value="#{pessoa.nome}" id="nome"/>
    <bjs:message for="nome" errorClass="error" tooltip="true"
id="mensagem1"/>
    <bjs:outputText value="Email:" id="labelemail"/>
    <bjs:inputText id="email" clientValidators="required" required="true"
value="#{pessoa.email}" >
    <f:validator validatorId="Email"/>
    </bjs:inputText>
    <bjs:message for="email" errorClass="error" tooltip="true"
id="mensagem2"/>
    <bjs:outputText value="Data:" />
    <bjs:datePicker id="data" binding="#{pessoa.dtNascimento}"/>
    <bjs:message for="data" errorClass="error" tooltip="true"
id="mensagem3"/>
    <bjs:outputText value="Arquivo:" />
    <bjs:fileUpload id="arquivo" target="#{pessoa.arquivo}"/>
    <bjs:message for="arquivo" errorClass="error" tooltip="true"
id="mensagem4"/>
    <bjs:commandButton action="cadastrar" value="Cadastrar" id="botao1"/>
    </bjs:panelGrid>
    <bjs:panelSet rows="300" id="list-detail-panelset"
binding="#{pessoaForm.listPanelSet}">
    <bjs:panel id="list-panel" styleClass="list-panel" resize="s"
resizeMode="line"
           minHeight="7px" resizeConstraint="..">
     <bjs:listGrid id="contacts-listgrid"
          binding="#{pessoaForm.listgrid}"
          style="height:200;width:450"
          value="#{pessoaForm.listaPessoas}" var="row" pagerOnBottom="true"
          pageSize="10" pageCache="3"
          sortListener="#{pessoaForm.sort}"
          dragSymbol="<img src='visuals/icon_contact_clean.gif'>">
      <bjs:column>
      <f:facet name="header">
      <bjs:outputText value="Nome" />
      </f:facet>
      <bjs:panelGroup>
      <bjs:outputText value="#{row.nome}" />
      </bjs:panelGroup>
</bjs:column>
<bjs:column>
<f:facet name="header">
<bjs:outputText value="Email" />
</f:facet>
<bjs:outputText value="#{row.email}" />
</bjs:column>
</bjs:listGrid>
</bjs:panel>
</bjs:panelSet>
```

```
  </bjs:application>
  </body>
</html>
</f:view>
```

Arquivo faces-config.xml

```
<?xml version="1.0"?>
<!DOCTYPE faces-config PUBLIC
  "-//Sun Microsystems, Inc.//DTD JavaServer Faces Config 1.0//EN"
  "http://java.sun.com/dtd/web-facesconfig_1_0.dtd">
<faces-config>
   <managed-bean>
      <managed-bean-name>sampleBackingBean</managed-bean-name>
      <managed-bean-
class>com.backbase.bjs.example.SampleBackingBean</managed-bean-class>
      <managed-bean-scope>session</managed-bean-scope>
   </managed-bean>
   <managed-bean>
      <managed-bean-name>pessoa</managed-bean-name>
      <managed-bean-class>br.com.tcc.backbase.util.Pessoa</managed-bean-
class>
      <managed-bean-scope>session</managed-bean-scope>
   </managed-bean>
   <managed-bean>
      <managed-bean-name>pessoaForm</managed-bean-name>
      <managed-bean-class>br.com.tcc.backbase.util.PessoaForm</managed-
bean-class>
      <managed-bean-scope>session</managed-bean-scope>
   </managed-bean>
   <navigation-rule>
        <navigation-case>
                 <from-outcome>cadastrar</from-outcome>
            <to-view-id>/grid.jsp</to-view-id>
        </navigation-case>
    </navigation-rule>
      <navigation-rule>
        <navigation-case>
                 <from-outcome>novo</from-outcome>
            <to-view-id>/index.jsp</to-view-id>
        </navigation-case>
    </navigation-rule>
</faces-config>
```

Arquivo web.xml

```
<?xml version="1.0"?>
<!DOCTYPE web-app PUBLIC
  "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
  "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
      <display-name>tccback</display-name>
      <context-param>
            <param-name>com.backbase.bjs.SESSION_TIMEOUT_PAGE</param-name>
            <param-value>/my-session-timeout-page.jsp</param-value>
      </context-param>
      <context-param>
            <param-name>com.backbase.bjs.BPC_BOOT_DIR</param-name>
            <param-value>/Backbase/3_1_8</param-value>
```

```
        </context-param>
        <context-param>
              <param-name>com.backbase.bjs.BPC_BOOT_FILE</param-name>
              <param-value>/bpc/boot.js</param-value>
        </context-param>
        <context-param>
              <param-name>com.backbase.bjs.PARSE_MOVE_EVENTS</param-name>
              <param-value>false</param-value>
              </context-param>
        <context-param>
              <param-name>com.backbase.bjs.UPDATE_MODEL_FIELDS</param-name>
              <param-value>true</param-value>
              </context-param>
        <context-param>
              <param-name>javax.faces.CONFIG_FILES</param-name>
              <param-value></param-value>
        </context-param>
        <context-param>
              <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
              <param-value>server</param-value>
        </context-param>
        <context-param>
              <param-name>
                    org.apache.myfaces.NUMBER_OF_VIEWS_IN_SESSION
              </param-name>
              <param-value>20</param-value>
        </context-param>
        <context-param>
              <param-name>
                    org.apache.myfaces.SERIALIZE_STATE_IN_SESSION
              </param-name>
              <param-value>false</param-value>
        </context-param>
        <context-param>
              <param-name>
                    org.apache.myfaces.COMPRESS_STATE_IN_SESSION
              </param-name>
              <param-value>true</param-value>
        </context-param>
        <context-param>
              <param-name>org.apache.myfaces.ALLOW_JAVASCRIPT</param-name>
              <param-value>true</param-value>
        </context-param>
        <context-param>
              <param-name>org.apache.myfaces.PRETTY_HTML</param-name>
              <param-value>false</param-value>
        </context-param>
        <context-param>
              <param-name>org.apache.myfaces.DETECT_JAVASCRIPT</param-name>
              <param-value>false</param-value>
        </context-param>
        <context-param>
              <param-name>org.apache.myfaces.AUTO_SCROLL</param-name>
              <param-value>false</param-value>
        </context-param>
        <filter>
              <filter-name>Upload Filter</filter-name>
              <filter-
  class>com.backbase.bjs.webapp.filter.UploadFilter</filter-class>
              <init-param>
```

```
            <param-
name>com.backbase.bjs.webapp.UploadFilter.repositoryPath</param-name>
            <param-value>/temp</param-value>
            </init-param>
            <init-param>
            <param-
name>com.backbase.bjs.webapp.UploadFilter.sizeThreshold</param-name>
            <param-value>1024</param-value>
            </init-param>
            <init-param>
            <param-
name>com.backbase.bjs.webapp.UploadFilter.sizeMax</param-name>
            <param-value>-1</param-value>
            </init-param>
      </filter>
      <filter-mapping>
            <filter-name>Upload Filter</filter-name>
            <url-pattern>*.jsf</url-pattern>
      </filter-mapping>
      <!-- Faces Servlet -->
      <servlet>
            <servlet-name>Faces Servlet</servlet-name>
            <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
            <load-on-startup>1</load-on-startup>
      </servlet>
      <!-- License Servlet -->
    <servlet>
        <servlet-name>Backbase License Servlet</servlet-name>
        <servlet-class>com.backbase.bjs.util.UploadLicenseServlet</servlet-
class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <url-pattern>*.jsf</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>Backbase License Servlet</servlet-name>
        <url-pattern>/start</url-pattern>
    </servlet-mapping>
      <!-- set the timeout to 30 minutes -->
      <session-config>
          <session-timeout>30</session-timeout>
      </session-config>
      <!-- Welcome files -->
      <welcome-file-list>
            <welcome-file>index.html</welcome-file>
      </welcome-file-list>
</web-app>
```

10.6 Exemplo BluePrints

Classe Autocomplete

```
package br.com.tcc.blueprints.util;
import java.util.ArrayList;
import java.util.List;
import javax.faces.context.FacesContext;
import com.sun.j2ee.blueprints.ui.autocomplete.CompletionResult;
public class Autocomplete {
	private List<String> listaNomes;
	private String nomeAtual;
	private List<String> listaNomesFixo;
	public Autocomplete(){
		listaNomes = new ArrayList<String>();
		listaNomesFixo = new ArrayList<String>();
		nomeAtual = "";
		this.populeListaNomes();
		listaNomesFixo.addAll(listaNomes);
	}
	private void populeListaNomes(){
		listaNomes.add("Anderson");
		listaNomes.add("Andrey");
		listaNomes.add("Augusto");
		listaNomes.add("Bruno");
		listaNomes.add("Caio");
		listaNomes.add("Denise");
		listaNomes.add("Diego");
		listaNomes.add("Douglas");
		listaNomes.add("Eder");
		listaNomes.add("Edson");
		listaNomes.add("Eduardo");
		listaNomes.add("Felipe");
		listaNomes.add("Fernando");
		listaNomes.add("Gabriel");
		listaNomes.add("Gilberto");
		listaNomes.add("Guilherme");
		listaNomes.add("Jeronimo");
		listaNomes.add("Joao");
		listaNomes.add("Leonardo");
		listaNomes.add("Luiz");
		listaNomes.add("Marcelo");
		listaNomes.add("Martín");
		listaNomes.add("Maximiliano");
		listaNomes.add("Paulo");
		listaNomes.add("Rafael");
		listaNomes.add("Sabrina");
		listaNomes.add("Tiago");
		listaNomes.add("Thiago");
		listaNomes.add("Tie");
		listaNomes.add("Victor");
	}
	public List getListaNomes() {
		if(listaNomes == null){
			listaNomes = new ArrayList();
		}
		return listaNomes;
	}
	public void setListaNomes(List listaNomes) {
```

```
            this.listaNomes = listaNomes;
      }
      public void sugereNome(FacesContext context, String prefix,
CompletionResult result) {
              listaNomes = new ArrayList<String>();
                  int tamanhoListaNomes = listaNomesFixo.size();
                  String valorAtual;
                  for(int i=0; i<tamanhoListaNomes;i++){
                        valorAtual = listaNomesFixo.get(i);

      if(valorAtual.toUpperCase().contains(prefix.toUpperCase())){
                              listaNomes.add(valorAtual);
                        }
                  }
      }
      public String getNomeAtual() {
            return nomeAtual;
      }
      public void setNomeAtual(String nomeAtual) {
            this.nomeAtual = nomeAtual;
      }
      public List getListaNomesFixo() {
            return listaNomesFixo;
      }
      public void setListaNomesFixo(List listaNomesFixo) {
            this.listaNomesFixo = listaNomesFixo;
      }
}
```

Classe Cidade

```
package br.com.tcc.blueprints.util;
public class Cidade {
      private Integer cdCidade;
      private String nmCidade;
      public Integer getCdCidade() {
            return cdCidade;
      }
      public void setCdCidade(Integer cdCidade) {
            this.cdCidade = cdCidade;
      }
      public String getNmCidade() {
            return nmCidade;
      }
      public void setNmCidade(String nmCidade) {
            this.nmCidade = nmCidade;
      }
}
```

Classe Estado

```
package br.com.tcc.blueprints.util;
public class Estado {
      private Integer cdEstado;
      private String deEstado;
      public Integer getCdEstado() {
            return cdEstado;
      }
      public void setCdEstado(Integer cdEstado) {
            this.cdEstado = cdEstado;
      }
      public String getDeEstado() {
```

```
            return deEstado;
      }
      public void setDeEstado(String deEstado) {
            this.deEstado = deEstado;
      }
}
```

Classe Endereco

```
package br.com.tcc.blueprints.util;
public class Endereco {
      private String nmRua;
      private Integer nuResidencia;
      private String deComplemento;
      private String nmBairro;
      private Cidade cidade;
      private Estado estado;
      public Endereco(){
            this.cidade = new Cidade();
            this.estado = new Estado();
      }
      public Cidade getCidade() {
            return cidade;
      }
      public void setCidade(Cidade cidade) {
            this.cidade = cidade;
      }
      public String getDeComplemento() {
            return deComplemento;
      }
      public void setDeComplemento(String deComplemento) {
            this.deComplemento = deComplemento;
      }
      public String getNmBairro() {
            return nmBairro;
      }
      public void setNmBairro(String nmBairro) {
            this.nmBairro = nmBairro;
      }
      public String getNmRua() {
            return nmRua;
      }
      public void setNmRua(String nmRua) {
            this.nmRua = nmRua;
      }
      public Integer getNuResidencia() {
            return nuResidencia;
      }
      public void setNuResidencia(Integer nuResidencia) {
            this.nuResidencia = nuResidencia;
      }
      public Estado getEstado() {
            return estado;
      }
      public void setEstado(Estado estado) {
            this.estado = estado;
      }
}
```

Classe GerenciadorEstadoCidade

```
package br.com.tcc.blueprints.util;
import java.util.ArrayList;
import java.util.List;
import javax.faces.event.ActionEvent;
import javax.faces.model.SelectItem;

public class GerenciadorEstadoCidade {
	private List<SelectItem> listaEstados;
	private List<SelectItem> listaCidades;

	public GerenciadorEstadoCidade(){
		listaEstados = new ArrayList<SelectItem>();
		listaCidades = new ArrayList<SelectItem>();
		this.populeListaEstados();
	}
	private void populeListaEstados(){
		listaEstados.add(new SelectItem("Paraná"));
		listaEstados.add(new SelectItem("Santa Catarina"));
		listaEstados.add(new SelectItem("Rio Grande do Sul"));

	}
	public List<SelectItem> getListaEstados() {
		return listaEstados;
	}
	public void setListaEstados(List<SelectItem> listaEstados) {
		this.listaEstados = listaEstados;
	}
	public List<SelectItem> getListaCidades() {
		return listaCidades;
	}
	public void setListaCidades(List<SelectItem> listaCidades) {
		this.listaCidades = listaCidades;
	}
}
```

Classe Pessoa

```
package br.com.tcc.blueprints.util;
import java.text.SimpleDateFormat;
import java.util.Date;

public class Pessoa implements Comparable{
	private Integer cdPessoa;
	//dados pessoais
	private String nome;
	private String sobrenome;
	private Date dtNascimento;
	private String nuCPF;
	private String emailPessoal;
	private String nuFonePessoal;
	private Endereco endereco;
	//dados profissionais
	private String emailProfissional;
	private String nuFoneProfissional;
	private String nmEmpresa;
	public Pessoa(){
		this.endereco = new Endereco();
	}
	public Pessoa(String nome){
		this.nome = nome;
```

```
	}
	public Pessoa(String nome, Date dtNasc, String nuCPF, String email,
String fone){
		this.nome = nome;
		this.dtNascimento = dtNasc;
		this.nuCPF = nuCPF;
		this.emailPessoal = email;
		this.nuFonePessoal = fone;
		this.endereco = new Endereco();
	}
	public Date getDtNascimento() {
		return dtNascimento;
	}
	public void setDtNascimento(Date dtNascimento) {
		this.dtNascimento = dtNascimento;
	}
	public String getNome() {
		return nome;
	}
	public void setNome(String nome) {
		this.nome = nome;
	}
	public String getNuCPF() {
		return nuCPF;
	}
	public void setNuCPF(String nuCPF) {
		this.nuCPF = nuCPF;
	}
	public String getSobrenome() {
		return sobrenome;
	}
	public void setSobrenome(String sobrenome) {
		this.sobrenome = sobrenome;
	}
	public String getEmailPessoal() {
		return emailPessoal;
	}
	public void setEmailPessoal(String emailPessoal) {
		this.emailPessoal = emailPessoal;
	}
	public String getEmailProfissional() {
		return emailProfissional;
	}
	public void setEmailProfissional(String emailProfissional) {
		this.emailProfissional = emailProfissional;
	}
	public String getNuFonePessoal() {
		return nuFonePessoal;
	}
	public void setNuFonePessoal(String nuFonePessoal) {
		this.nuFonePessoal = nuFonePessoal;
	}
	public String getNuFoneProfissional() {
		return nuFoneProfissional;
	}
	public void setNuFoneProfissional(String nuFoneProfissional) {
		this.nuFoneProfissional = nuFoneProfissional;
	}
	public String getNmEmpresa() {
		return nmEmpresa;
	}
```

```
	public void setNmEmpresa(String nmEmpresa) {
		this.nmEmpresa = nmEmpresa;
	}
	public Endereco getEndereco() {
		return endereco;
	}
	public void setEndereco(Endereco endereco) {
		this.endereco = endereco;
	}
	public String getDataFormatada(){

		SimpleDateFormat dateFormat = new
SimpleDateFormat("dd/MM/yyyy");
		String strData = dateFormat.format(this.getDtNascimento());

		return strData;
	}
	public int compareTo( Object obj ) {
		Pessoa outraPessoa = (Pessoa)obj;
		return this.getNome().compareTo(outraPessoa.getNome());
	}
	public Integer getCdPessoa() {
		return cdPessoa;
	}
	public void setCdPessoa(Integer cdPessoa) {
		this.cdPessoa = cdPessoa;
	}
}
```

Classe PessoaForm

```
package br.com.tcc.blueprints.util;
import javax.faces.event.ActionEvent;
import br.com.tcc.util.validator.CPFValidator;
public class PessoaForm {
	private Pessoa pessoa;
	private GerenciadorEstadoCidade gerenciadorEstadoCidade;
	private String mensagemCpfErro;
	private String mensagemEmailErro;
	private CPFValidator cpfvalidator;

	public PessoaForm(){
			this.pessoa = new Pessoa();
			this.gerenciadorEstadoCidade = new
GerenciadorEstadoCidade();
			this.cpfvalidator = new CPFValidator();
	}
	public Pessoa getPessoa() {
		return pessoa;
	}
	public void setPessoa(Pessoa pessoa) {
		this.pessoa = pessoa;
	}
	public GerenciadorEstadoCidade getGerenciadorEstadoCidade() {
		return gerenciadorEstadoCidade;
	}
	public void setGerenciadorEstadoCidade(
			GerenciadorEstadoCidade gerenciadorEstadoCidade) {
		this.gerenciadorEstadoCidade = gerenciadorEstadoCidade;
	}
	public String getMensagemCpfErro() {
		return mensagemCpfErro;
```

```
	}
	public void setCpfErro(String cpfErro) {
		this.mensagemCpfErro = cpfErro;
	}
	public CPFValidator getCpfvalidator() {
		return cpfvalidator;
	}
	public void setCpfvalidator(CPFValidator cpfvalidator) {
		this.cpfvalidator = cpfvalidator;
	}
	public String getMensagemEmailErro() {
		return mensagemEmailErro;
	}
	public void setMensagemEmailErro(String mensagemEmailErro) {
		this.mensagemEmailErro = mensagemEmailErro;
	}
	public void setMensagemCpfErro(String mensagemCpfErro) {
		this.mensagemCpfErro = mensagemCpfErro;
	}
}
```

Classe CPFValidator

```
package br.com.tcc.util.validator;
public class CPFValidator{
	/* Cálculo do CPF foi retirado da página
http://www2.fundao.pro.br/articles.asp?cod=23 */
	public boolean confirmaCPF (String strCpf ) {
		int	d1, d2;
		int	digito1, digito2, resto;
		int	digitoCPF;
		String  nDigResult;
		d1 = d2 = 0;
		digito1 = digito2 = resto = 0;
		for (int nCount = 1; nCount < strCpf.length() -1; nCount++){

		   digitoCPF = Integer.valueOf (strCpf.substring(nCount -1,
nCount)).intValue();
		   d1 = d1 + ( 11 - nCount ) * digitoCPF;
		   d2 = d2 + ( 12 - nCount ) * digitoCPF;
		}
		resto = (d1 % 11);
		if (resto < 2)
		   digito1 = 0;
		else
		   digito1 = 11 - resto;
		d2 += 2 * digito1;
		resto = (d2 % 11);
		if (resto < 2)
		   digito2 = 0;
		else
		   digito2 = 11 - resto;
		String nDigVerific = strCpf.substring (strCpf.length()-2,
strCpf.length());
		nDigResult = String.valueOf(digito1) + String.valueOf(digito2);
		return nDigVerific.equals(nDigResult);
	    }
}
```

Arquivo index.jsp

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<jsp:root version="1.2" xmlns:bp="http://java.sun.com/blueprints/ui/14"
xmlns:f="http://java.sun.com/jsf/core"
xmlns:h="http://java.sun.com/jsf/html"
    xmlns:jsp="http://java.sun.com/JSP/Page"
xmlns:ui="http://www.sun.com/web/ui">
    <jsp:directive.page contentType="text/html;charset=UTF-8"
pageEncoding="UTF-8"/>
    <f:view>
        <h:form>
        <h:panelGroup style="display:block">
        <h:panelGrid columns="2">
        <h:outputText value="Olá: " style="width:125"/>
        <h:outputText value="#{pessoaForm.pessoa.nome}"  id="nome"
style="width:150"/>
        </h:panelGrid>
        <h:panelGrid columns="3">
        <h:outputText value="Nome: " style="width:125"/>
        <bp:autoComplete value="#{pessoaForm.pessoa.nome}"
binding="#{autocomplete.listaNomes}"
completionMethod="#{autocomplete.sugereNome}" id="autoComplete1" />
        </h:panelGrid>
        <h:panelGrid columns="3">
        <h:outputText value="CPF: " style="width:125"/>
        <h:inputText value="#{pessoaForm.pessoa.nuCPF}">
        </h:inputText>
        <h:outputText id="msg" value="#{pessoaForm.mensagemCpfErro}" />
        </h:panelGrid>
        <h:panelGrid columns="3">
        <h:outputText value="Email: " style="width:125"/>
        <h:inputText value="#{pessoaForm.pessoa.emailPessoal}">
        </h:inputText>
        <h:outputText id="msgEmail" value="#{pessoaForm.mensagemEmailErro}"
/>
        </h:panelGrid>
       </h:panelGroup>
       <h:panelGroup style="display:block" >
       <h:outputText value="Estado : " />
       <h:selectOneMenu
value="#{pessoaForm.pessoa.endereco.estado.deEstado}" id="selectEstado" >
       <f:selectItems
value="#{pessoaForm.gerenciadorEstadoCidade.listaEstados}"/>
      </h:selectOneMenu>
      </h:panelGroup>
      <h:panelGroup style="display:block" >
      <h:outputText value="Cidade : "/>
      <h:selectOneMenu
value="#{pessoaForm.pessoa.endereco.cidade.nmCidade}" id="selectCidade">
      <f:selectItems
value="#{pessoaForm.gerenciadorEstadoCidade.listaCidades}"
id="listaCidades"/>
      </h:selectOneMenu>
      </h:panelGroup>
      </h:form>
     </f:view>
</jsp:root>
```

Arquivo start.jsp

```
<html>
  <head>
```

```
    <title>My JSP 'start.jsp' starting page</title>
  </head>
  <body>
      <jsp:forward page="index.faces"/>
  </body>
</html>
```

Arquivo web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="2.4"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
  <context-param>
    <param-name>javax.faces.CONFIG_FILES</param-name>
    <param-value>/WEB-INF/faces-config.xml</param-value>
  </context-param>
  <servlet>
    <servlet-name>Faces Servlet</servlet-name>
    <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
    <load-on-startup>0</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>*.faces</url-pattern>
  </servlet-mapping>
  <welcome-file-list>
      <welcome-file>start.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

Arquivo faces-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE faces-config PUBLIC "-//Sun Microsystems, Inc.//DTD JavaServer
Faces Config 1.1//EN" "http://java.sun.com/dtd/web-facesconfig_1_1.dtd">
<faces-config>
      <managed-bean>
            <managed-bean-name>pessoa</managed-bean-name>
            <managed-bean-class>br.com.tcc.blueprints.util.Pessoa</managed-
bean-class>
            <managed-bean-scope>session</managed-bean-scope>
      </managed-bean>
      <managed-bean>
            <managed-bean-name>cidade</managed-bean-name>
            <managed-bean-class>br.com.tcc.blueprints.util.Cidade</managed-
bean-class>
            <managed-bean-scope>session</managed-bean-scope>
      </managed-bean>
      <managed-bean>
            <managed-bean-name>endereco</managed-bean-name>
            <managed-bean-
class>br.com.tcc.blueprints.util.Endereco</managed-bean-class>
            <managed-bean-scope>session</managed-bean-scope>
      </managed-bean>
      <managed-bean>
            <managed-bean-name>estado</managed-bean-name>
            <managed-bean-class>br.com.tcc.blueprints.util.Estado</managed-
bean-class>
            <managed-bean-scope>session</managed-bean-scope>
```

```
	</managed-bean>
	<managed-bean>
		<managed-bean-name>gerenciadorEstadoCidade</managed-bean-name>
		<managed-bean-
class>br.com.tcc.blueprints.util.GerenciadorEstadoCidade</managed-bean-
class>
		<managed-bean-scope>session</managed-bean-scope>
	</managed-bean>
	<managed-bean>
		<managed-bean-name>autocomplete</managed-bean-name>
		<managed-bean-
class>br.com.tcc.blueprints.util.Autocomplete</managed-bean-class>
		<managed-bean-scope>session</managed-bean-scope>
	</managed-bean>
	<managed-bean>
		<managed-bean-name>pessoaForm</managed-bean-name>
		<managed-bean-
class>br.com.tcc.blueprints.util.PessoaForm</managed-bean-class>
		<managed-bean-scope>session</managed-bean-scope>
	</managed-bean>
	<managed-bean>
		<managed-bean-name>CPFValidator</managed-bean-name>
		<managed-bean-
class>br.com.tcc.util.validator.CPFValidator</managed-bean-class>
		<managed-bean-scope>session</managed-bean-scope>
	</managed-bean>
</faces-config>
```

10.7 Exemplo DWR

Classe Cidade

```
package br.com.tcc.dwr.util;
public class Cidade {
	private Integer cdCidade;
	private String nmCidade;
	public Integer getCdCidade() {
		return cdCidade;
	}
	public void setCdCidade(Integer cdCidade) {
		this.cdCidade = cdCidade;
	}
	public String getNmCidade() {
		return nmCidade;
	}
	public void setNmCidade(String nmCidade) {
		this.nmCidade = nmCidade;
	}
}
```

Classe Estado

```
package br.com.tcc.dwr.util;
public class Estado {
	private Integer cdEstado;
	private String deEstado;
	public Integer getCdEstado() {
		return cdEstado;
	}
```

```
	public void setCdEstado(Integer cdEstado) {
		this.cdEstado = cdEstado;
	}
	public String getDeEstado() {
		return deEstado;
	}
	public void setDeEstado(String deEstado) {
		this.deEstado = deEstado;
	}
}
```

Classe Pessoa

```
package br.com.tcc.dwr.util;
import java.util.Date;
public class Pessoa implements Comparable{
	private String nome;
	private String nuCpf;
	private String email;
	private Date dtNascimento;
	private Integer cdPessoa;
	public Pessoa(){}
	public Pessoa(String nome){
		this.nome = nome;
	}
	public Pessoa(String nome, Date dtNasc, String nuCPF, String email){

		this.nome = nome;
		this.dtNascimento = dtNasc;
		this.nuCpf = nuCPF;
		this.email = email;
	}
	public String getEmail() {
		return email;
	}
	public void setEmail(String email) {
		this.email = email;
	}
	public String getNome() {
		return nome;
	}
	public void setNome(String nome) {
		this.nome = nome;
	}
	public String getNuCpf() {
		return nuCpf;
	}
	public void setNuCpf(String nuCpf) {
		this.nuCpf = nuCpf;
	}
	public Date getDtNascimento() {
		return dtNascimento;
	}
	public void setDtNascimento(Date dtNascimento) {
		this.dtNascimento = dtNascimento;
	}
	public Integer getCdPessoa() {
		return cdPessoa;
	}
	public void setCdPessoa(Integer cdPessoa) {
		this.cdPessoa = cdPessoa;
	}
```

```
	public int compareTo(Object o) {
		Pessoa pessoaAux = (Pessoa)o;
		return this.getCdPessoa().compareTo( pessoaAux.getCdPessoa() );
	}
	public boolean verificaEmail(String email){
		System.out.println("email "+email);
		return email.contains("@");
	}
}
```

Classe PessoaForm

```
package br.com.tcc.dwr.util;
public class PessoaForm {
	private Pessoa pessoa;
	public PessoaForm(){
			this.pessoa = new Pessoa();
	}
	public Pessoa getPessoa() {
		return pessoa;
	}
	public void setPessoa(Pessoa pessoa) {
		this.pessoa = pessoa;
	}
}
```

Classe CPFValidator

```
package br.com.util.validator;
import javax.faces.application.FacesMessage;
import javax.faces.component.UIComponent;
import javax.faces.context.FacesContext;
import javax.faces.validator.Validator;
import javax.faces.validator.ValidatorException;
public class CPFValidator implements Validator{
	private final int tamanhoCpf = 11;
	public void validate(FacesContext context, UIComponent component,
Object value) throws ValidatorException {
		if(value == null){
			return;
		}
		if(!confirmaCPF(value.toString())){
			FacesMessage msg = new
FacesMessage(FacesMessage.SEVERITY_ERROR, "ERROR1", "CPF inválido");
			throw new ValidatorException(msg);
		}
	}
	/* Cálculo do CPF foi retirado da página
http://www2.fundao.pro.br/articles.asp?cod=23 */
	private boolean confirmaCPF (String strCpf ) {
		int     d1, d2;
		int     digito1, digito2, resto;
		int     digitoCPF;
		String  nDigResult;
		d1 = d2 = 0;
		digito1 = digito2 = resto = 0;
		for (int nCount = 1; nCount < strCpf.length() -1; nCount++){

		   digitoCPF = Integer.valueOf (strCpf.substring(nCount -1,
nCount)).intValue();
		   d1 = d1 + ( 11 - nCount ) * digitoCPF;
		   d2 = d2 + ( 12 - nCount ) * digitoCPF;
```

```
            }
            resto = (d1 % 11);
            if (resto < 2)
               digito1 = 0;
            else
               digito1 = 11 - resto;
            d2 += 2 * digito1;
            resto = (d2 % 11);
            if (resto < 2)
               digito2 = 0;
            else
               digito2 = 11 - resto;
            String nDigVerific = strCpf.substring (strCpf.length()-2,
strCpf.length());
            nDigResult = String.valueOf(digito1) + String.valueOf(digito2);
            return nDigVerific.equals(nDigResult);
         }
}
```

Arquivo dwr.xml

```
<!DOCTYPE dwr PUBLIC
    "-//GetAhead Limited//DTD Direct Web Remoting 1.0//EN"
    "http://www.getahead.ltd.uk/dwr/dwr10.dtd">
<dwr>
      <init>
            <creator id="jsf"
                  class="uk.ltd.getahead.dwr.create.JsfCreator" />
      </init>
      <allow>
            <create creator="jsf" javascript="Pessoa">
                  <param name="managedBeanName"  value="pessoa" />
                  <param name="class" value="br.com.tcc.dwr.util.Pessoa" />
                  <include method="verificaEmail"/>
            </create>
      </allow>
</dwr>
```

Arquivo web.xml

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

      <filter>
            <filter-name>ajaxExtensionFilter</filter-name>
            <filter-
class>uk.ltd.getahead.dwr.servlet.FacesExtensionFilter</filter-class>
      </filter>
      <filter-mapping>
            <filter-name>ajaxExtensionFilter</filter-name>
            <url-pattern>/dwr/*</url-pattern>
      </filter-mapping>
      <context-param>
            <param-name>javax.faces.CONFIG_FILES</param-name>
            <param-value>/WEB-INF/faces-config.xml</param-value>
      </context-param>
      <servlet>
            <servlet-name>dwr-invoker</servlet-name>
```

```
            <servlet-class>uk.ltd.getahead.dwr.DWRServlet</servlet-class>
            <init-param>
                  <param-name>debug</param-name>
                  <param-value>true</param-value>
            </init-param>
      </servlet>
      <servlet>
            <servlet-name>Faces Servlet</servlet-name>
            <servlet-class>
            javax.faces.webapp.FacesServlet</servlet-class>
            <init-param>
                  <param-name>debug</param-name>
                  <param-value>3</param-value>
            </init-param>
            <init-param>
                  <param-name>detail</param-name>
                  <param-value>3</param-value>
            </init-param>
            <load-on-startup>1</load-on-startup>
      </servlet>
      <servlet-mapping>
            <servlet-name>dwr-invoker</servlet-name>
            <url-pattern>/dwr/*</url-pattern>
      </servlet-mapping>
      <servlet-mapping>
            <servlet-name>Faces Servlet</servlet-name>
            <url-pattern>*.faces</url-pattern>
      </servlet-mapping>
      <servlet-mapping>
            <servlet-name>Faces Servlet</servlet-name>
            <url-pattern>/faces/*</url-pattern>
      </servlet-mapping>
      <welcome-file-list>
        <welcome-file>start.jsp</welcome-file>
    </welcome-file-list>
</web-app>
```

Arquivo faces-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE faces-config PUBLIC "-//Sun Microsystems, Inc.//DTD JavaServer
Faces Config 1.1//EN" "http://java.sun.com/dtd/web-facesconfig_1_1.dtd">
<faces-config>
 <navigation-rule>
            <from-view-id>/index.jsp</from-view-id>
            <navigation-case>
                  <from-outcome>login</from-outcome>
                  <to-view-id>/result.jsp</to-view-id>
            </navigation-case>
 </navigation-rule>
 <managed-bean>
      <managed-bean-name>pessoaForm</managed-bean-name>
      <managed-bean-class>br.com.tcc.dwr.util.PessoaForm</managed-bean-
class>
      <managed-bean-scope>session</managed-bean-scope>
 </managed-bean>

 <managed-bean>
      <managed-bean-name>pessoa</managed-bean-name>
```

```
      <managed-bean-class>br.com.tcc.dwr.util.Pessoa</managed-bean-class>
      <managed-bean-scope>session</managed-bean-scope>
 </managed-bean>
</faces-config>
```

Arquivo start.jsp

```
<html>
  <head>
    <title>My JSP 'start.jsp' starting page</title>
  </head>
  <body>
      <jsp:forward page="index.faces"/>
  </body>
</html>
```

Arquivo index.jsp

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
 <script type='text/javascript'
src='/tccdwr/dwr/interface/Pessoa.js'></script>
 <script type='text/javascript'
src='/tccdwr/dwr/interface/GerenciadorEstadoCidade.js'></script>
 <script type='text/javascript' src='/tccdwr/dwr/engine.js'></script>
 <script type='text/javascript' src='/tccdwr/dwr/util.js'></script>
<%@ include file="./layout/cabecalhofamilia.jsp" %>

<html>
  <head>
    <title>Index</title>
  </head>
  <body>
      <f:view>
      <h:form>
      Nome:<h:inputText value="#{pessoaForm.pessoa.nome}"/>
      <br>
      Email: <h:inputText value="#{pessoaForm.pessoa.email}"
onblur="verifica(this.value)"/>
      <div id="msgEmail"><h:outputText value="Email inválido"/></div>
      <br>
      <h:commandButton value="OK" action="login"/>
      </h:form>
      </f:view>
</body>
</html>
<%@ include file="./layout/rodape.htm" %>
<script>
      idErroEmail = document.getElementById('msgEmail');
      idErroEmail.style.display = 'none';
      function verifica( email ){
            Pessoa.verificaEmail(mostraErro, email);
      }
      function mostraErro( displayError ){
            idMensagem = document.getElementById('msgEmail');
            if(displayError == true){
                  idMensagem.style.display = 'none';
            }else{
                  idMensagem.style.display = 'inline';
            }
```

```
      }
</script>
```

Arquivo result.jsp

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ include file="./layout/cabecalhofamilia.jsp" %>
<html>
  <head>
    <title>Resultado</title>
  </head>
  <body>
      <f:view>
      <h:form>
      Cadastro efetuado com sucesso!
      <table>
      <tr>
      <td width="150">
      <h:outputText value="Nome:"/>
      </td>
      <td width="200">
      <h:outputText value="#{pessoaForm.pessoa.nome}"/>
      </td>
      <tr>
      <td width="150"><h:outputText value="Email:"/></td>
      <td width="200"><h:inputText
value="#{pessoaForm.pessoa.email}"/></td>
      </tr>
      </table>
      </h:form>
      </f:view>
</body>
</html>
<%@ include file="./layout/rodape.htm" %>
```

10.8 Exemplo ValidatorAjax

Classe AjaxValidatorRenderer

```
package br.com.tcc.validatorAjax.util;
import java.io.IOException;
import java.util.Map;
import javax.faces.component.UIComponent;
import javax.faces.component.UIInput;
import javax.faces.context.FacesContext;
import javax.faces.context.ResponseWriter;
import javax.faces.render.Renderer;
public class AjaxValidatorRenderer extends Renderer{
      public void encodeBegin(FacesContext context, UIComponent
component)throws IOException{
            ResponseWriter writer = context.getResponseWriter();
            String clienteId = component.getClientId(context);
            writer.startElement("input", component);
            writer.writeAttribute("name", clienteId, "clientId");
            Object valor = ((UIInput)component).getValue();
            if( valor != null){
                  writer.writeAttribute("value", valor, "value");
            }
```

```
            String size = (String)component.getAttributes().get("size");
            if(size != null){
                  writer.writeAttribute("size", size, "size");
            }
            String idValidator =
(String)component.getAttributes().get("idValidator");
            if(idValidator == null){
                  idValidator = "";
            }
            writer.writeAttribute("idValidator", idValidator,
"idValidator");
            String strIdCampo =
(String)component.getAttributes().get("id");
            if(strIdCampo == null){
                  strIdCampo = "";
            }
            writer.writeAttribute("id",strIdCampo,"id");
            writer.writeAttribute("onblur",
"validate(this.id,this.value,this.idValidator)", "onblur");
            writer.writeAttribute("onmouseover","mouseSobre(this.id)",
"onmouseover");
            writer.endElement("input");
      }

      public void decode(FacesContext context, UIComponent component){
            Map requestMap =
context.getExternalContext().getRequestParameterMap();
            String clienteId = component.getClientId(context);
            String strValor =
(String)component.getAttributes().get("value");
            ((UIInput)component).setSubmittedValue(strValor);
            ((UIInput)component).setValid(true);
      }
}
```

Classe CPFValidator

```
package br.com.tcc.validatorAjax.util;
public class CPFValidator extends Validate {
      @Override
      public boolean valide(String valor) {
            /* Cálculo do CPF foi retirado da página
http://www2.fundao.pro.br/articles.asp?cod=23 */
            int     d1, d2;
            int     digito1, digito2, resto;
            int     digitoCPF;
            String  nDigResult;
            d1 = d2 = 0;
            digito1 = digito2 = resto = 0;
            for (int nCount = 1; nCount < valor.length() -1; nCount++){
               digitoCPF = Integer.valueOf (valor.substring(nCount -1,
nCount)).intValue();
               d1 = d1 + ( 11 - nCount ) * digitoCPF;
               d2 = d2 + ( 12 - nCount ) * digitoCPF;
            }
            resto = (d1 % 11);
            if (resto < 2)
               digito1 = 0;
            else
               digito1 = 11 - resto;
```

```
            d2 += 2 * digito1;
            resto = (d2 % 11);
            if (resto < 2)
               digito2 = 0;
            else
               digito2 = 11 - resto;
            String nDigVerific = valor.substring (valor.length()-2,
valor.length());
            nDigResult = String.valueOf(digito1) + String.valueOf(digito2);
            return nDigVerific.equals(nDigResult);
      }
}
```

Classe EmailValidator

```
package br.com.tcc.validatorAjax.util;
import java.util.Vector;
import javax.faces.application.FacesMessage;
import javax.faces.component.UIComponent;
import javax.faces.context.FacesContext;
import javax.faces.validator.Validator;
import javax.faces.validator.ValidatorException;

public class EmailValidator extends Validate{
      public boolean valide(String valor){
            return valor.contains("@");
      }
}
```

Classe GerenciadorServlet

```
package br.com.tcc.validatorAjax.util;

import javax.faces.validator.Validator;
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class GerenciadorServlet extends HttpServlet{

      private ServletConfig servletConfig = null;

       public void destroy() {
        servletConfig = null;
       }

       public ServletConfig getServletConfig() {
         return (this.servletConfig);
       }

       public String getServletInfo() {
        return (this.getClass().getName());
       }

       public void init(ServletConfig servletConfig) throws
ServletException {
             this.servletConfig = servletConfig;
       }

       public void doGet(HttpServletRequest request,HttpServletResponse
```

```
response)throws java.io.IOException,ServletException {

            java.io.PrintWriter out=response.getWriter();

            String strIdCampo        = (String)request.getParameter("id");
            String strValor          =
(String)request.getParameter("valorDigitado");
            String strValidatorClass =
(String)request.getParameter("validatorClass");


            try{
                  Validate validator =
(Validate)Class.forName(strValidatorClass).newInstance();
                  boolean validou = validator.valide(strValor);
                  String resposta = strIdCampo;
                  if(!validou){
                        resposta = resposta +"=false";
                  }else{
                        resposta = resposta +"=true";
                  }
                  out.print(resposta);

            }catch(Exception e){
                  System.out.println("Erro na validação");
                  out.print(strIdCampo+"=false");
            }
            out.flush();
      }



       public void doPost(HttpServletRequest request,HttpServletResponse
response) throws java.io.IOException, ServletException {
            doGet(request, response);
       }
}
```

Classe Pessoa

```
package br.com.tcc.validatorAjax.util;
public class Pessoa {
      private String nome;
      private String email;
      private String nuCPF;
      public String getEmail() {
            return email;
      }
      public void setEmail(String email) {
            this.email = email;
      }
      public String getNome() {
            return nome;
      }
      public void setNome(String nome) {
            this.nome = nome;
      }
      public String getNuCPF() {
            return nuCPF;
      }
```

```
	public void setNuCPF(String nuCPF) {
		this.nuCPF = nuCPF;
	}
}
```

Classe UIValidatorAjax

```
package br.com.tcc.validatorAjax.util;
public class Pessoa {
	private String nome;
	private String email;
	private String nuCPF;
	public String getEmail() {
		return email;
	}
	public void setEmail(String email) {
		this.email = email;
	}
	public String getNome() {
		return nome;
	}
	public void setNome(String nome) {
		this.nome = nome;
	}
	public String getNuCPF() {
		return nuCPF;
	}
	public void setNuCPF(String nuCPF) {
		this.nuCPF = nuCPF;
	}
}
```

Classe Validate

```
package br.com.tcc.validatorAjax.util;
public class Pessoa {
	private String nome;
	private String email;
	private String nuCPF;
	public String getEmail() {
		return email;
	}
	public void setEmail(String email) {
		this.email = email;
	}
	public String getNome() {
		return nome;
	}
	public void setNome(String nome) {
		this.nome = nome;
	}
	public String getNuCPF() {
		return nuCPF;
	}
	public void setNuCPF(String nuCPF) {
		this.nuCPF = nuCPF;
	}
}
```

Classe ValidatorAjaxTag

```
package br.com.tcc.validatorAjax.util;
import javax.faces.application.Application;
import javax.faces.component.UIComponent;
import javax.faces.context.FacesContext;
import javax.faces.el.ValueBinding;
import javax.faces.webapp.UIComponentTag;
public class ValidatorAjaxTag extends UIComponentTag{
      private String size;
      private String value;
      private String idValidator;
      public String getRendererType(){
            return null;
      }
      public String getComponentType(){
            return "br.com.tcc.validatorAjax.util.ValidatorAjax";
      }
      public void setProperties(UIComponent component){
            super.setProperties(component);
            setString(component,"size", size);
            setString(component,"value", value);
            setString(component,"idValidator", idValidator);
            component.getAttributes().put("onblur",
"validate(this.id,this.value,this.idValidator)");

      component.getAttributes().put("onmouseover","mouseSobre(this.id)");

      }
      public void setInteger(UIComponent component, String nomeAtributo,
String valorAtributo){
            if(valorAtributo == null){
                  return;
            }
            if(isValueReference(valorAtributo)){
                  setValueBinding(component,nomeAtributo, valorAtributo);
            }else{
                  component.getAttributes().put(nomeAtributo,
valorAtributo);
            }
      }
      public void setString(UIComponent component, String nomeAtributo,
String valorAtributo){
            if(valorAtributo == null){
                  return;
            }
            if(isValueReference(valorAtributo)){
                  setValueBinding(component, nomeAtributo, valorAtributo);
            }else{
                  component.getAttributes().put(nomeAtributo,
valorAtributo);
            }
      }
      public void setValueBinding(UIComponent component, String
nomeAtributo, String valorAtributo){
            FacesContext context = FacesContext.getCurrentInstance();
            Application app = context.getApplication();
            ValueBinding vb = app.createValueBinding(valorAtributo);
            component.setValueBinding(nomeAtributo, vb);
      }
      public void release(){
            super.release();
            size  = null;
```

```
            value = null;
            idValidator = null;
      }
      public void setIdValidator(String idValidator) {
            this.idValidator = idValidator;
      }
      public void setSize(String size) {
            this.size = size;
      }
      public void setValue(String value) {
            this.value = value;
      }
}
```

Arquivo web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="2.4"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
  <context-param>
    <param-name>javax.faces.CONFIG_FILES</param-name>
    <param-value>/WEB-INF/faces-config.xml</param-value>
  </context-param>
  <servlet>
    <servlet-name>Faces Servlet</servlet-name>
    <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
    <load-on-startup>0</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>*.faces</url-pattern>
  </servlet-mapping>
  <welcome-file-list>
  <welcome-file>start.jsp</welcome-file>
  </welcome-file-list>
  <servlet>
      <servlet-name>GerenciadorServlet</servlet-name>
      <servlet-
class>br.com.tcc.validatorAjax.util.GerenciadorServlet</servlet-class>
  </servlet>
  <servlet-mapping>
      <servlet-name>GerenciadorServlet</servlet-name>
      <url-pattern>/GerenciadorServlet</url-pattern>
  </servlet-mapping>
</web-app>
```

Arquivo faces-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE faces-config PUBLIC "-//Sun Microsystems, Inc.//DTD JavaServer
Faces Config 1.1//EN" "http://java.sun.com/dtd/web-facesconfig_1_1.dtd">
<faces-config>
      <managed-bean>
            <managed-bean-name>pessoa</managed-bean-name>
            <managed-bean-
class>br.com.tcc.validatorAjax.util.Pessoa</managed-bean-class>
            <managed-bean-scope>session</managed-bean-scope>
      </managed-bean>
      <managed-bean>
```

```
            <managed-bean-
name>br.com.tcc.validatorAjax.util.CPFValidator</managed-bean-name>
            <managed-bean-
class>br.com.tcc.validatorAjax.util.CPFValidator</managed-bean-class>
            <managed-bean-scope>session</managed-bean-scope>
      </managed-bean>
            <managed-bean>
            <managed-bean-
name>br.com.tcc.validatorAjax.util.EmailValidator</managed-bean-name>
            <managed-bean-
class>br.com.tcc.validatorAjax.util.EmailValidator</managed-bean-class>
            <managed-bean-scope>session</managed-bean-scope>
      </managed-bean>
      <component>
            <component-
type>br.com.tcc.validatorAjax.util.ValidatorAjax</component-type>
            <component-
class>br.com.tcc.validatorAjax.util.UIValidatorAjax</component-class>
      </component>
      <render-kit>
            <renderer>
                  <component-family>javax.faces.Input</component-family>
                  <renderer-
type>br.com.tcc.validatorAjax.util.AjaxValidatorRenderer</renderer-type>
                  <renderer-
class>br.com.tcc.validatorAjax.util.AjaxValidatorRenderer</renderer-class>
            </renderer>
      </render-kit>
</faces-config>
```

Arquivo start.jsp

```
<html>
  <head>
  </head>
  <body>
            <jsp:forward page="index.faces"/>
  </body>
</html>
```

Arquivo index.jsp

```
<%@ include file="./layout/cabecalhofamilia.jsp" %>
<%@ include file="./WEB-INF/js/ajax.js"%>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://www.inf.ufsc.br/tcc" prefix="ajax"%>

<html>
  <head>
      <TITLE>index</TITLE>
  </head>

  <body>
      <f:view>
            <h:form>
            <table width="100%" border="0">
            <tr>
                  <td width="80">Nome:</td>
                  <td width="*">
                  <h:inputText value="#{pessoa.nome}" size="80"/>
                  </td>
```

```
                    <td width="400"> </td>
                    </tr>
                    <tr>
                    <td width="80">Email:</td>
                    <td width="*">
                    <ajax:validatorAjax value="#{pessoa.email}"
idValidator="br.com.tcc.validatorAjax.util.EmailValidator" id="teste"
size="80" rendered="true" >
                    </ajax:validatorAjax>
                    </td>
                    <td width="400"> </td>
                    </tr>
                    <tr>
                    <td width="80">CPF:</td>
                    <td width="*">
                    <ajax:validatorAjax value="#{pessoa.email}"
idValidator="br.com.tcc.validatorAjax.util.CPFValidator" id="testeCPF"
size="80" rendered="true" >
                    </ajax:validatorAjax>
                    </td>
                    <td width="400"> </td>
             </tr>
             </table>
             <h:commandButton value="OK" action="login"/>
             </h:form>
      </f:view>
</body>
</html>
```

Arquivo ajax.js

```
<script language="javascript" type="text/javascript">
   var request = false;
   try {
     request = new XMLHttpRequest();
   } catch (trymicrosoft) {
     try {
       request = new ActiveXObject("Msxml2.XMLHTTP");
     } catch (othermicrosoft) {
       try {
         request = new ActiveXObject("Microsoft.XMLHTTP");
       } catch (failed) {
         request = false;
       }
     }
   }
   if (!request)
     alert("Error initializing XMLHttpRequest!");
   function validate( idCampo, valorDigitado, validatorClass ) {
       campo = document.getElementById(idCampo);
       classeValidadora =
campo.attributes.getNamedItem('idValidator').value;
       var url =
"/validatorAjax/GerenciadorServlet?id="+idCampo+"&valorDigitado="+valorDigi
tado+"&validatorClass="+classeValidadora;
     request.open("GET", url, true);
     request.onreadystatechange = updatePage;
     request.send(null);
   }

   function updatePage() {
     if (request.readyState == 4) {
```

```
        if (request.status == 200) {
          var response = request.responseText.split("=");
          campo = document.getElementById(response[0]);
          if((response.length < 2)||(response[1]=='false')){
             campo.style.color="red";
          }else{
             campo.style.color="";
          }
        } else if (request.status == 404) {
          alert ("Requested URL is not found.");
        } else if (request.status == 403) {
          alert("Access denied.");
        } else
          alert("status is " + request.status);
      }
    }
    function mouseSobre(id){
        campo = document.getElementById(id);
        if(campo.style.color == "red"){
             mensagem = "O valor deste campo é inválido";

        }
    }
</script>
```

Integração JavaServer Faces e AJAX: Estudo da Integração entre as tecnologias JSF e AJAX

Diego Luiz Marafon

Bacharelado em Ciências da Computação, 2006
Departamento de Informática e Estatística
Universidade Federal de Santa Catarina (UFSC), Brasil, 88040-900
dmarafon@inf.ufsc.br

Resumo

Um projeto em evidência no mundo do desenvolvimento Web é o framework JavaServer Face (JSF), o qual surgiu graças ao grande investimento da empresa Sun Microsystems e é atualmente um dos frameworks mais utilizado para a construção de aplicativos para a Internet.Porém este framework tem como uma de suas características a utilização exclusiva de requisições síncronas, exigindo assim que a página seja totalmente construída a cada evento realizado, causando desta forma certo desconforto aos usuários da aplicação.Uma possível solução para tal problema é a integração JSF com o conjunto de tecnologias AJAX, tornando possível a utilização de requisições assíncronas,o que permite alterar apenas alguns dados da página.Este artigo se propõe a analisar e comparar as principais formas de integração entre o framework JSF e o conjunto de tecnologias AJAX.

Palavras-chave: Java, J2EE, JavaServer Faces, AJAX, MVC, JavaScript, aplicativos Web.

Abstract

The web development has been increasing a lot in the last years. Many investments have been applied in projects and technologies to improve the development of applications that runs on the internet, especially the ones that use the Java programming language and the platform J2EE.One of these projects is the JavaServer Faces framework, that acts mainly in the View and Control in the MVC design pattern and has the followings objectives: facilitate components development and use event-oriented programming like in desktop systems development. Another project that is making a lot of success is the one that develop a technology that uses JavaScript named AJAX. One of the benefits that AJAX brings is that the application can change some data that are showed to the user without rebuilding the whole page. Using AJAX, the application becomes more dynamic, because the interface that is showed to the user doesn't have to "blink" every time that the system changes a data. Since both technologies could be integrated, this work is going to show: the ways that both could work together, some projects that are working with this objective and study if in the future these couldn't be totally integrated.

Keywords: Java, J2EE, JavaServer Faces, AJAX, MVC, JavaScript, Web applications.

Introdução

Com a evolução e popularização da internet, muitos investimentos estão sendo aplicados em projetos, com o intuito de facilitar e padronizar o desenvolvimento de aplicativos *Web*.

Um desses projetos que está em moda atualmente é o que desenvolve o *framework* JavaServer Faces, o qual atua principalmente nas camadas de Visão e Controle dos aplicativos, segundo o modelo MVC. Entre os principais fatores do seu sucesso, podemos citar: possui um conjunto de componentes prontos e padronizados, tem um modelo de programação dirigida a eventos e permite que novos componentes possam ser facilmente criados pelos desenvolvedores.

Outra característica deste *framework* é a utilização de apenas requisições síncronas. Adicionalmente, sabe-se que os aplicativos *Web* se baseiam no modelo de programação Cliente/Servidor. Assim, a cada evento realizado pelo cliente, é criada uma requisição que vai até o servidor para realizar um processamento. Estas duas características associadas fazem com que, a cada evento, uma nova página deva ser totalmente construída e exibida, mesmo que esta seja igual à anterior, sem permitir que o usuário interaja com o sistema durante esses acontecimentos.

Uma possível solução para este problema é a utilização de AJAX(*Asynchronous JavaScript and XML*), um conjunto de tecnologias em evidência no contexto da programação *Web*, devido o fato de que as páginas não precisam mais ser totalmente reconstruídas quando determinado evento ocorre no *browser* do cliente. Utilizando este conjunto de tecnologias, as páginas podem atualizar apenas os dados que foram modificados, tornando os aplicativos muito mais interativos na visão do usuário.

Porém, ainda há pouca integração do AJAX com os *frameworks* que atuam nas camadas de Visão e Controle, que são as camadas mais interessadas nas vantagens oferecidas por esse conjunto de tecnologias.

Este artigo tem como objetivo analisar e comparar as diversas formas de se efetuar a integração JSF e AJAX, objetivando facilitar a sua utilização no desenvolvimento de aplicativos *Web.*

JavaServer Faces

O JSF surgiu em setembro de 2002, graças ao grande investimento da empresa *Sun Microsystems*, a qual objetiva torná-lo o *framework*, para as camadas de Visão e Controle dos aplicativos *Web,* padrão de mercado.

Mesmo tendo pouco tempo de existência, JSF já faz parte da especificação J2EE 5.0 e é um dos *frameworks* mais utilizados no desenvolvimento de aplicativos *Web*.

Uma outra característica deste *framework* é conhecida como ciclo de vida das requisições JSF, o qual é a representação do caminho percorrido por uma requisição, desde sua criação, com a realização de um evento, até a renderização da página a ser exibida para o usuário.

Este ciclo de vida das requisições JSF é dividido em seis fases, porém esse *framework* permite que os desenvolvedores acrescentem novas etapas ou implementem artifícios para que a requisição não execute determinada fase.

AJAX

O Internet Explorer 5 foi lançado pela Microsoft no final do século passado, contendo um novo objeto, conhecido como *ActiveX*, o qual permitia a realização de requisições assíncronas.

Com o sucesso do *ActiveX*, o W3C(*World Wide Web Consortium*), conjunto de empresas que desenvolvem e

definem padrões do WWW, padronizou a implementação nos *browsers* de um objeto similar ao *ActiveX*, o qual foi chamado de *XMLHttpRequest*.

Com a implementação de alguns padrões nos navegadores, como o *XMLHttpRequest* e do DOM, e a utilização do XML e da linguagem JavaScript, os desenvolvedores puderam implementar recursos nos aplicativos para atualizar apenas alguns dados de uma página sem ter que reconstruí-la integralmente, tarefa esta que anteriormente só era possível de ser feita através de artifícios de programação.

Em fevereiro de 2005, um desenvolvedor chamado Jesse James Garret nomeou de AJAX a integração de várias tecnologias, tais como: XML, JavaScript, DOM, e outras, no intuito de acrescentar mais dinamismo aos programas.

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 necessitando esperar enquanto o servidor processa a informação. O resto da sigla, "JAX", demonstra os dois principais componentes utilizados: JavaScript e XML.

Formas de integração

Pode-se dividir a integração entre JSF e AJAX de duas maneiras: com o uso de componentes JSF, adicionando na página as funcionalidades AJAX ou com a confecção de componentes personalizados.

O único fator que faz com que a adoção de componentes JSF seja mais vantajoso é quando a funcionalidade é utilizada apenas uma vez no sistema e dificilmente será usada em outro aplicativo.

Caso contrário, sempre é recomendável que o desenvolvedor crie seus próprios componentes, obtendo assim inúmeras vantagens, tais como: evita duplicação de código, facilitando assim a manutenção e resolve problemas genéricos.

Sem a utilização de componentes personalizados

A primeira forma de integração que foi utilizada pelos desenvolvedores é conhecida como "Desenvolvedor responsável por toda a integração", a qual não necessita a construção de componentes personalizados.

Esta forma de integração está cada vez mais em desuso devido aos seguintes fatores: necessidade de duplicação de código caso a funcionalidade seja utilizada em várias partes do sistema e por resolver apenas problemas específicos.

Outra forma de integração JSF e AJAX é conhecida como "Modificando a especificação JSF", apresentada no último JavaOne, a qual está sendo vista com muita simpatia pelos desenvolvedores.

A idéia principal é modificar a especificação JSF, incorporando assim o conjunto de tecnologias AJAX de maneira nativa aos componentes deste *framework*. Porém, esta idéia está numa fase inicial e pode levar certo tempo para ser incorporada a especificação JSF.

Utilizando componentes personalizados

A forma de integração mais utilizada atualmente é aquela que desenvolve componentes personalizados, a qual pode ser dividida em duas idéias principais, dependendo do número de *Servlets* utilizados.

Uma das formas mais utilizada é a que faz uso de apenas o *Servlet* disponibilizado pelo JavaServer Faces, conhecido como *FacesServlet* e inseri uma nova fase para ser executada após a primeira fase do ciclo de vida das requisições JSF. Uma de suas vantagens é

a facilidade de implementação. Porém, utilizando-se esta idéia, o desempenho do aplicativo decai à medida que aumenta o número de novas fases inseridas no ciclo de vida das requisições JSF.

Outra forma de integração muito utilizada atualmente e que desenvolve componentes personalizados é aquela na qual o desenvolvedor cria um novo *Servlet* apenas para gerenciar as requisições assíncronas e continua usando o *FacesServlet* para administrar as requisições síncronas.

Segundo este trabalho, esta é a melhor forma de integração JSF e AJAX, pois: faz uma clara separação entre requisições síncronas e assíncronas, pode ser facilmente implementada, é eficiente e não enfrenta os problemas citados nas idéias anteriores.

Bibliotecas de componentes

Foi realizado um estudo comparativo entre diversas bibliotecas, as quais disponibilizam componentes JSF que fazem uso do conjunto de tecnologias AJAX, nas quais foram analisados os seguintes quesitos: custo, qualidade dos componentes e principalmente forma de integração utilizada.

As bibliotecas comparadas foram: *Icefaces*, *Ajaxfaces*, *Ajax4Jsf*, *Backbase*, *Blueprints*, *DWR*, *Mabon* e *Dynafaces*.

Considerando as bibliotecas estudadas e os quesitos analisados, o projeto *Icefaces* é o melhor para o aproveitamento dos recursos da integração AJAX e JSF para o desenvolvimento de aplicativos.

Componente desenvolvido

Segundo estudo realizado foi concluído que a melhor forma de integração JSF e AJAX para ser utilizada atualmente é aquela que desenvolve componentes personalizados e cria-se um novo *Servlet* para gerenciar as requisições assíncronas.

Fazendo uso dessa idéia foi desenvolvido um componente, chamado de *validatorAjax*, com a principal funcionalidade de permitir a validação de campos de entrada de conteúdo dos aplicativos, interagindo com o usuário tão logo o foco seja retirado do referido campo, sem a necessidade de renderizar toda a página.

Conclusão

A importância dada ao assunto integração JSF e AJAX, motivou os responsáveis pela especificação JSF para que a versão 2.0, a qual deve ser lançada em 2008, contemple a integração deste *framework* com a tecnologia AJAX. A maneira mais cotada a ser adotada para implementar tal integração é a que foi apresentado anteriormente conhecido como “Modificando a especificação JSF”.

Caso tais requisitos forem implementados no JSF 2.0, a construção de bibliotecas de componentes JavaServer Faces que disponibilizam as funcionalidades AJAX se tornará uma tarefa muito mais fácil de ser realizada.

Enquanto isso não se tornar realidade, segundo este estudo desenvolvido, a melhor forma de integração existente até o momento, devido principalmente à facilidade de implementação e eficiência, é aquela a qual desenvolve componentes personalizados e cria um novo *Servlet* para gerenciar as requisições assíncronas.

Referências Bibliográficas

[1] ADAPTATIVE PATH. **Adaptative Path. Ajax: A New Approach To Web Applications**. Disponível em: <http://adaptivepath.com/publications/essays/archives/000385.php>.Acesso em: 10

jun. 2006.

[2] BASLER. BASLER, Mark. **Using PhaseListener Approach for JavaServer Faces Technology with AJAX**. Disponível em: <https://blueprints.dev.java.net/bpcatalog/ee5/ajax/phaselistener.html>.Acesso em: 05 out. 2006.

[3] BASLER. BASLER, Mark. **Using a Servlet with JavaServer Faces Technology and AJAX**. Disponível em: <https://blueprints.dev.java.net/bpcatalog/ee5/ajax/servletControllerwithJSF.html>.Acesso em: 05 out. 2006.

[4] BREAU. BREAU, Philip. **JSF1.2 and JSF2.0** [mensagem pessoal]. Mensagem recebida por <dmarafon@gmail.com> em 22 nov. 2006.

[5] BURNS&HOOKOM&WINER. BURNS, Ed; HOOKOM, Jacob; WINER, Adam. **Evolving JavaServer Faces Technology: AJAX Done Right**. Disponível em: <http://weblogs.java.net/blog/edburns/20060519-ajax-done-right-00.html>.Acesso em: 11 out. 2006.

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

[7] GEARY&HORSTANN. GEARY, D.;HORSTANN, Cay. **Core JavaServer Faces**. Alta Books. 2005. p.1-234.

[8] HOOKOM. 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 nov. 2006.

[9] ICESOFT. **ICEfaces the rich web application**. Disponível em: <http://www.icesoft.com>.Acesso em: 1º jul. 2006.

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

[11] JACOBI&FALLOWS. 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.

[12] JAVAONE. JavaOne. **Suns 2006 Worldwide Java Developer**. Disponível em: <http://sunapp1.whardy.com:8090/jsf-j12/home.jsf>.Acesso em: 1º nov. 2006.

[13] ORT&BASLER. ORT, Ed; BASLER, Mark. **AJAX Design Strategies**. Disponível em: <http://java.sun.com/developer/technicalArticles/J2EE/AJAX/DesignStrategies>. Acesso em: 12 out. 2006.

[14] SUN. **Asynchronous JavaScript and XML**. Disponível em: <http://java.sun.com/developer/technicalArticles/J2EE/AJAX>.Acesso em: 15 jun. 2006.