

UNIVERSIDADE ESTADUAL DO SUDOESTE DA BAHIA  
CURSO DE CIÊNCIA DA COMPUTAÇÃO

**DESENVOLVIMENTO DE UM SISTEMA DE CONTROLE DE  
TCC PARA A UESB UTILIZANDO JSF E SPRING**

VITÓRIA DA CONQUISTA

2010

FABRÍCIO COSTA SILVA

DESENVOLVIMENTO DE UM SISTEMA DE CONTROLE DE  
TCC PARA A UESB UTILIZANDO JSF E SPRING

Trabalho de conclusão de curso apresentado à  
Universidade Estadual do Sudoeste da Bahia  
como requisito parcial à obtenção do título de  
bacharel em Ciência da Computação.

Orientador: Fabrício de Sousa Pinto.

VITÓRIA DA CONQUISTA

2010

**FABRÍCIO COSTA SILVA**

**ANÁLISE DA ACESSIBILIDADE DO COMÉRCIO ELETRÔNICO**

Trabalho de conclusão de curso apresentado à disciplina de Projeto de Computação Supervisionado II do curso de Ciência da Computação, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

**Banca examinadora:**

**Orientador:**

---

Fabício de Sousa Pinto  
Universidade Estadual do Sudoeste da Bahia - UESB

**Membro:**

---

Clésio Rubens de Matos  
Faculdade de Tecnologia e Ciências – FTC

**Membro:**

---

Maria Silvia Santos Barbosa  
Universidade Estadual do Sudoeste da Bahia – UESB

**Vitória da Conquista, 22/02/2010**

## DEDICATÓRIA

*Aos meus pais, pela compreensão e o estímulo em todos os momentos e aos meus irmãos Denilton e Cinara.*

## **AGRADECIMENTOS**

Primeiramente agradeço aos meus pais por terem sempre me incentivados no caminho da educação sem medir esforços para me oferecer uma educação de qualidade e por me ensinar os caminhos da ética e humildade.

Agradeço também aos meus irmãos, Denilton e Cinara por me apoiarem sempre nessa minha caminhada em busca de meus sonhos.

Agradeço aos meus amigos da Faculdade Juvêncio Terra onde trabalhei por mais de 2 anos. Em especial às pessoas de Virginia e Verônica por sempre me ajudar quando precisei.

Agradeço aos meus amigos da república “Epanós6”, Alvino, Antonio, Judson, Eudes, Manoel, Daniel, Alan Night.

Agradeço a minha grande amiga e colega de faculdade Mara, por ser uma pessoa que me ajudou muito no período de graduação, com sua amizade e carinho.

Agradeço aos meus amigos que contribuíram de forma direta ou indireta para eu conseguir chegar aqui.

## RESUMO

O desenvolvimento web está a cada dia mais em evidências e as aplicações para esse ambientes estão ganhando complexidade. Isto se deve ao fato da internet estar cada dia mais presente na vida das pessoas, passando a ser um ambiente propício para o desenvolvimento de aplicações. O *JSF* e o *Spring* são tecnologias que fornecem um ambiente sofisticado de programação, possibilitando obtermos códigos cada vez mais rápidos e em uma arquitetura de fácil entendimento e manuseio. Objetivou-se então, analisar esses dois frameworks para se identificar um ambiente adequado de programação e testes. Foi grande a importância da arquitetura do sistema e o uso da metodologia RAD na web, que foram pontos importantes para se atingir os objetivos propostos e o JSF juntamente com o Spring ajudam a atingir nossos objetivos.

Palavras-chaves:*Java Server Faces, Spring, framework, web.*

## LISTA DE FIGURAS

<i>Figura 1: Representação das fases do MVC</i> .....	9
<i>Figura 2: Hierarquia dos componentes do JSF</i> .....	14
<i>Figura 3: Representação de um componente no cliente e no servidor</i> .....	15
<i>Figura 4: Ciclo de vida de uma requisição JSF</i> .....	20
<i>Figura 5: Aquisição de dependência no modelo tradicional</i> .....	23
<i>Figura 6: Aquisição de dependência com DI</i> .....	24
<i>Figura 7: Membros fundamentais do Spring Security</i> .....	25
<i>Figura 8: Diagrama de Caso de uso do Sistema eDoc</i> .....	32
<i>Figura 9: Modelagem do Banco de dados</i> .....	33
<i>Figura 10: Arquitetura do sistema</i> .....	34
<i>Figura 11: Componente com listagem de professores</i> .....	35
<i>Figura 12: Componente de Cadastro/edição de Professor</i> .....	36
<i>Figura 13: Componente de Cadastro/edição de Professor com erro de validação</i> .....	36
<i>Figura 14: Tela do sistema com listagem de TCC</i> .....	39
<i>Figura 15: Tela do sistema com edição de TCC</i> .....	40
<i>Figura 16: Classe do Managed Bean de TCC</i> .....	41
<i>Figura 17: Regra de segurança do Spring Security</i> .....	42
<i>Figura 18: Comando condicional para restringir partes de uma página JSF</i> .....	42
<i>Figura 19: Tela do sistema logado com usuário comum</i> .....	43
<i>Figura 20: Tela de edição/cadastro de TCC</i> .....	47
<i>Figura 22: Tela de busca por curso e palavra-chave</i> .....	48
<i>Figura 23: Tela com resultado de pesquisa da figura 22</i> .....	48
<i>Figura 24: Tela com listagem de colegiados</i> .....	49
<i>Figura 25 : Diagrama de classe</i> .....	50

## LISTA DE QUADROS

<i>Quadro 1: Termos chaves do JSF</i> .....	12
<i>Quadro 2: Requisitos do eDoc</i> .....	30



## SIGLAS

<b>AOP</b>	<i>Aspect Oriented Programming</i>
<b>DI</b>	<i>Dependency Injection</i>
<b>GUI</b>	<i>Graphics User Interface</i>
<b>IDE</b>	<i>Integrated Development Enviroment</i>
<b>JEE</b>	<i>Java Enterprise Edition</i>
<b>JSF</b>	<i>Java Server Faces</i>
<b>JSP</b>	<i>Java Server Pages</i>
<b>MVC</b>	<i>Model View Controller</i>
<b>RAD</b>	<i>Rapid Application development</i>
<b>WWW</b>	<i>World Wide Web</i>
<b>CRUD</b>	<i>Create, read, update and delete</i>
<b>TCC</b>	Trabalho de conclusão de curso
<b>UI</b>	<i>User interface</i>

# Sumário

<b>1. Introdução.....</b>	<b>3</b>
<b>1.1. Situação problemática .....</b>	Erro! Indicador não definido.
<b>1.2. Objetivos .....</b>	Erro! Indicador não definido.
1.2.1. Objetivo Geral .....	Erro! Indicador não definido.
1.2.2. Objetivos Específicos.....	Erro! Indicador não definido.
<b>1.3. Justificativa .....</b>	Erro! Indicador não definido.
<b>1.4. Metodologia .....</b>	Erro! Indicador não definido.
<b>2. Tecnologias para desenvolvimento web.....</b>	<b>7</b>
<b>2.1. Servlets.....</b>	<b>7</b>
<b>2.2. JSP .....</b>	<b>7</b>
<b>2.3. MVC.....</b>	<b>8</b>
<b>2.4. JavaBeans .....</b>	<b>10</b>
<b>2.5. Java Server Faces.....</b>	<b>10</b>
2.5.1. Por que <i>JSF</i> ? .....	11
2.5.2. Principais componentes JFS .....	12
2.5.3. UI Componentes .....	13
2.5.3.1. Componentes UI padrões.....	13
2.5.4. Árvore de componentes.....	15
2.5.5. Renderizadores .....	15
2.5.6. Validadores .....	16
2.5.7. Backing beans.....	17
2.5.8. Conversores .....	18
2.5.9. Mensagens .....	19
2.5.10. Navegação .....	19
2.5.11. Ciclo de vida .....	20
2.5.11.1. <i>Restore view</i> .....	21
2.5.11.2. <i>Apply request values</i> .....	21
2.5.11.3. <i>Process validations</i> .....	21
2.5.11.4. <i>Update model values</i> .....	21
2.5.11.5. <i>Invoke application</i> .....	22
2.5.11.6. <i>Render response</i> .....	22
<b>2.6. Spring framework.....</b>	<b>22</b>
<b>2.6.1. Injeção de dependência.....</b>	<b>23</b>
<b>2.6.2. Spring security .....</b>	<b>24</b>
2.6.2.1. Interceptadores de segurança.....	25
2.6.2.2. Gerenciador de autenticação .....	25
2.6.2.3. Gerenciador de decisões de acesso .....	26
2.6.2.4. Gerenciador de execução (Run-as) .....	26
<b>2.7. Programação orientada a aspectos .....</b>	<b>26</b>
<b>2.8. Integração entre JSF e Spring .....</b>	<b>27</b>
<b>3. Desenvolvimento de um sistema de controle de TCC para a UESB utilizando JSF e Spring</b>	<b>29</b>

<b>3.1.</b>	<b>Introdução .....</b>	<b>29</b>
<b>3.2.</b>	<b>Metodologia de desenvolvimento.....</b>	<b>29</b>
<b>3.3.</b>	<b>Requisitos .....</b>	<b>30</b>
<b>3.4.</b>	<b>Diagrama do banco de dados.....</b>	<b>32</b>
<b>3.5.</b>	<b>Arquitetura do sistema.....</b>	<b>34</b>
3.5.1.	Interface com o usuário .....	35
3.5.2.	Autenticação de usuário .....	36
3.4.3.	Controle de acesso .....	37
3.4.4.	Acesso ao banco de dados .....	37
<b>3.6.</b>	<b>Implementação .....</b>	<b>37</b>
<b>4.</b>	<b><i>Análise dos Resultados .....</i></b>	<b>38</b>
<b>5.</b>	<b><i>Considerações Finais e Trabalhos Futuros .....</i></b>	<b>44</b>
<b>6.</b>	<b><i>Referências.....</i></b>	<b>45</b>
<b>7.</b>	<b><i>Apêndice 1 – Telas do Sistema E-Doc.....</i></b>	<b>47</b>
<b>7.1.</b>	<b><i>Apêndice 2 – Diagrama de classe da aplicação .....</i></b>	<b>49</b>

# 1. Introdução

O mundo está cada vez mais girando em torno da internet. O número de serviços oferecidos através dela é crescente e tornam-se necessárias tecnologias que dêem viabilidade técnica para que estes serviços possam ser implementados.

Desde a popularização da internet, quando Tim Berners-Lee (1989) publicou um projeto que viria a ser a World Wide Web (WWW), começaram a surgir as primeiras informações em forma de hipertexto estático que circulavam pela nova rede. Com o passar do tempo as informações ficaram mais complexas e necessitava-se então de um dinamismo para que aquilo que fosse oferecido servisse para alguma finalidade mais específica.

“Com a expansão da Web, as necessidades das pessoas e organizações cresceram também. Por exemplo, os primeiros navegadores, antes do Mosaic, suportavam apenas documentos textos. Em 1993, com a criação do Mosaic e alguns outros navegadores similares, o HTML incorporou também imagens (...) . Outros elementos foram adicionados também à especificação inicial criada por Berners-Lee para atender às demandas crescentes de estruturação de documentos. O HTML, inicialmente, era apenas uma aplicação do formato SGML, uma metalinguagem cujo propósito é criar linguagens de marcação de documentos.(...) Isso, por sua vez, permitiu o crescimento rápido do HTML pela adição desses novos elementos e atributos.” FERRAZ (2003)

Do dinamismo de informações surgem as aplicações Web, que vieram tirar proveito da infra-estrutura da internet para eliminar as barreiras físicas das empresas permitindo acesso às informações empresarias remotamente, oferecendo uma interface externa para que funcionários e/ou clientes interajam com a empresa.

Surge então algumas linguagens que permitem esse dinamismo, são as linguagens *server-side*. Dentre essas linguagens, está o *Java*. Apesar de não ser desenvolvido especificamente para a Web o *Java* adaptou-se a evolução do mundo e engajou-se na plataforma Web sendo hoje uma das principais linguagens para desenvolvimento Web no mundo.

Segundo Conceição (2008), o *Java* tem um desenvolvimento gradativo que visa a evolução, adaptando-se as várias vertentes do desenvolvimento de software. E um campo de grande evolução do *Java* foi no desenvolvimento Web, hoje chegamos a uma plataforma de desenvolvimento Web padrão do *Java* chamada de *Java Server Faces (JSF)*, que é um dos focos deste texto.

O *Java* evolui gradativamente para se chegar à atual plataforma de desenvolvimento Web, o *JSF*. Inicialmente surgiam os *Servlets*, seguidos pelo *Java Server Pages (JSP)* que evoluiu para o *JSF* que é uma tecnologia robusta e veloz que propicia rapidez e facilidade no desenvolvimento de aplicações Web.

Então mostrar-se-á os benefícios obtido através dessa integração de tecnologia, pois acredita-se ser uma excelente solução para os problemas que tocam o desenvolvimento de sistemas orientados a objeto no âmbito Web.

A complexidade das aplicações web será o ponto que este trabalho tratará. Promover uma arquitetura de desenvolvimento web baseada na plataforma *Java Enterprise Edition (JEE)*, aumenta a produtividade no desenvolvimento e gera código mais fácil de modificar, testar e com pouco acoplamento? A arquitetura de um sistema pode ser definida como a estrutura ou estruturas do sistema, englobando elementos, propriedades visíveis externamente e relacionamentos entre eles Bass (2005).

O objetivo desse trabalho é inicialmente mostrar as vantagens da tecnologia de desenvolvimento Web *JSF* juntamente com o *Spring Framework*, criados para lidar com a complexidade de desenvolvimento de aplicativos corporativos. Mostrando o *JSF* como uma ferramenta de ligação entre os componentes da interface e o comportamento da aplicação e usar o *Spring* para tirar do *JSF* a responsabilidade de gerenciar aspectos da arquitetura da aplicação do lado servidor, deixando o *JSF* basicamente com a lógica de apresentação.

Com o crescimento da utilização da plataforma Web pelas aplicações corporativas, viu-se concorrentemente o crescimento da complexidade do desenvolvimento de tais aplicações nesse novo ambiente. A necessidade de produzir um código fácil de manter, pouco acoplado, fácil de testar e rápido de se produzir foram observados com o passar dos tempos. Então, a utilização de padrões de desenvolvimento específicos para Web se tornou uma necessidade, pois, seriam aplicadas soluções que tiveram êxito anteriormente.

O *JSF* e o *Spring* framework foram as tecnologias escolhidas que provêm alguns padrões já implementados. Podemos citar o padrão *Model View Controller* (MVC) e a injeção de dependência como algumas das funcionalidades providas por essas tecnologias.

A utilização de uma biblioteca de componentes gráficos, semelhante ao desenvolvimento de aplicativos em Delphi (clique e arrastar) também agilizará o desenvolvimento de aplicativos web. Esta é uma característica que o *JSF* traz consigo.

Foi observando os fatores acima citados, que escolheu-se o *JSF* e o *Spring* para serem estudados e mostrar seus benefícios no âmbito de facilitar o complexo processo de desenvolvimento de aplicativos Web.

Estudaremos características do *Spring* e *JSF* para obter um ambiente de programação que oferece facilidades tanto na parte gráfica quanto na parte arquitetural da aplicação. Pois acredita-se que os benefícios de separação de lógica, do baixo acoplamento, facilidade nos testes e nas modificações são características que tornam essa parceria *JSF/Spring* um ambiente que vale a pena ser estudado.

O estudo de caso deste trabalho é desenvolver um sistema de gerenciamento de TCC. Atualmente os Trabalhos de Conclusão de Curso ou Monografias entregues na UESB ficam perdidos na biblioteca sem uma forma eficaz de serem buscados. Muitos trabalhos ficam esquecidos nos CDs que são entregues e o resultado de um trabalho que custou a ser desenvolvido fica esquecido. Então, é importante um sistema que possa fazer o armazenamento e buscas desses trabalhos.

Tendo por base os objetivos propostos, fazer a integração de dois frameworks em um ambiente de programação web, será mostrado, utilizando pressupostos teóricos, que essa integração representa uma melhora no processo de desenvolvimento de software. Sendo essa afirmação corroborada, posteriormente serão desenvolvidos exemplos em software para mostrar como se faz a integração e os aspectos de reutilização de componentes gráficos do *JSF* e do controle da aplicação com o *Spring*.

De forma mais detalhada, as etapas da metodologia adotada são as seguintes:

1. Levantamentos preliminares sobre o *JSF* e o *Spring*.
2. Mostrar-se-á exemplos de melhorias e características que essa integração oferece;

3. Serão desenvolvidos exemplos práticos em software para mostrar o funcionamento dos frameworks integrados;
4. Análise dos testes realizados;

## 2. Tecnologias para desenvolvimento web

Nas seções seguintes iremos mostrar as principais tecnologias e padrões utilizados nesse trabalho. São elas:

- ✓ JSF
- ✓ Spring
- ✓ MVC
- ✓ Java

### 2.1. Servlets

Segundo Costa (2006), os *Servlets* surgiram de um esforço da *Sun Microsystems* na criação de uma arquitetura de aplicações WEB padronizada e teve sua primeira versão lançada em junho de 1997. Ainda segundo a *Sun Microsystems* um *Servlet* é uma tecnologia que prover aos desenvolvedores web um simples e consistente mecanismo de estender as funcionalidades de um servidor web. Um *Servlet* é um pequeno servidor baseados em requisições e respostas. Na prática um *Servlet* é uma classe *Java* que estende a classe ***Javax.servlet.HttpServlet***. Com ele podemos adicionar conteúdo dinâmico a uma página web em um servidor *Java*.

Um *Servlet* recebe requisições de, por exemplo, um formulário HTML e captura esses dados, faz o processamento e os devolvem à página destino. Tais requisições (*request*) retorna algo (*response*), podendo ser desde página HTML ou uma imagem.

Uma *Servlet* pode receber várias requisições ao mesmo tempo em um servidor web, por isso ela é mais rápida que um programa CGI (*Common Gateway Interface*) comum.

### 2.2. JSP

Segundo descrição da Sun Microsystems, *Java Server Pages* (JSP) habilita os desenvolvedores web criarem e manterem, rapidamente, aplicações web. Como parte da família de tecnologias do *Java*, o JSP permite desenvolver rapidamente aplicações baseadas na web que são independentes de plataforma. A tecnologia JSP separa a interface do usuário das regras de negócio, possibilitando ao *designer* trabalhar com o layout da página sem alterar o funcionamento da aplicação.



O JSP, apesar de facilitar a vida dos designers, ainda apresenta muitas tarefas manuais que tomam muito tempo do programador e que não representam nenhum avanço no objetivo do software. É comum perder tempo inserindo um calendário, um script de validação de algum campo, etc.

Quando adentrarmos na tecnologia *Java Server Faces (JSF)*, poderemos perceber que esta tem todas as facilidades do JSP e muito mais. Uma vantagem do *JSF* é que, segundo Pontes (2008), ele é de 30% a 50% mais rápido que o JSP devido ao fato de eliminar o compilador JSP. Pois, toda página *JSF* é transformada na verdade em um tipo de servlet. Todo o código HTML é compilado em forma de código *Java*.

### 2.3. MVC

O MVC (*Model View Controller*) popularizou-se com Smalltalk Goldberg & Robson (1983) e agora é utilizada em muitos frameworks modernos de interface com o usuário. Um dos benefícios primários do MVC é que ele prover reusabilidade de componentes para interfaces interativas com o usuário. Swing faz isso bem para aplicações desktop, enquanto um framework chamado *Struts* faz isso para a web. Como o *Struts*, o *JSF* é destinado para aplicações web em *Java*, mas segue princípios parecidos com os do Swing concentrando-se em componentes reusáveis de interfaces com o usuário.

Segundo Conceição (2008), a intenção do padrão MVC é particionar uma aplicação dinâmica em três camadas separada: *Model*, *View* e *Controller*. O *Model* representa as classes que representam os dados e as regras de negócios, a *View* renderiza os dados e os apresentam na tela para os usuários da aplicação, e o *Controller* lida com a interação do usuário ou a entrada de dados.

A parte mais dinâmica de um software interativo é a interface com o usuário (*View*) porque é a forma do usuário comunicar-se diretamente com a aplicação. Em outras palavras, esta é a mais visível da maioria dos softwares. Além disso, os sistemas mudam constantemente de requisitos, devendo ter a portabilidade para rodar em outros sistemas operacionais como (*Windows, Linux, Apple, etc*), até mesmo mudar a aparência do sistema (*look and feel*), que pode se tornar uma tarefa muito difícil a depender das tecnologias e padrões que são utilizados.

Segundo Dudney, et all (2004), aplicações com alto acoplamento fazem com que pequenas mudanças se tornem muito complicadas e propiciam a ocorrência de erros na aplicação. Futuramente, o software pode ter requisitos diferentes, então, será preciso manter

versões separadas de uma mesma aplicação se um grupo possuir requisitos de interfaces diferentes que outros. Em uma aplicação acoplada pode ser difícil juntar tudo isso novamente.

O padrão MVC propicia uma solução flexível para estes problemas através do desacoplamento dos componentes do *Model*, *View* e *Controller* promovendo uma comunicação uniforme entre eles. Temos algumas motivações para usar MVC:

- ❖ Várias apresentações para um mesmo dado (Tabela, gráfico).
- ❖ Mudança da aparência da aplicação (*look and feels*) ou mesmo executar em diferentes sistemas operacionais
- ❖ Quando o usuário altera dados do *Model*, eles são refletidos rapidamente nos componentes de apresentação do dado (*View*).
- ❖ Reutilizar um ou mais componentes de interface com o usuário independente os dados da aplicação.

A estrutura do padrão MVC é mostrada na Figura 1:

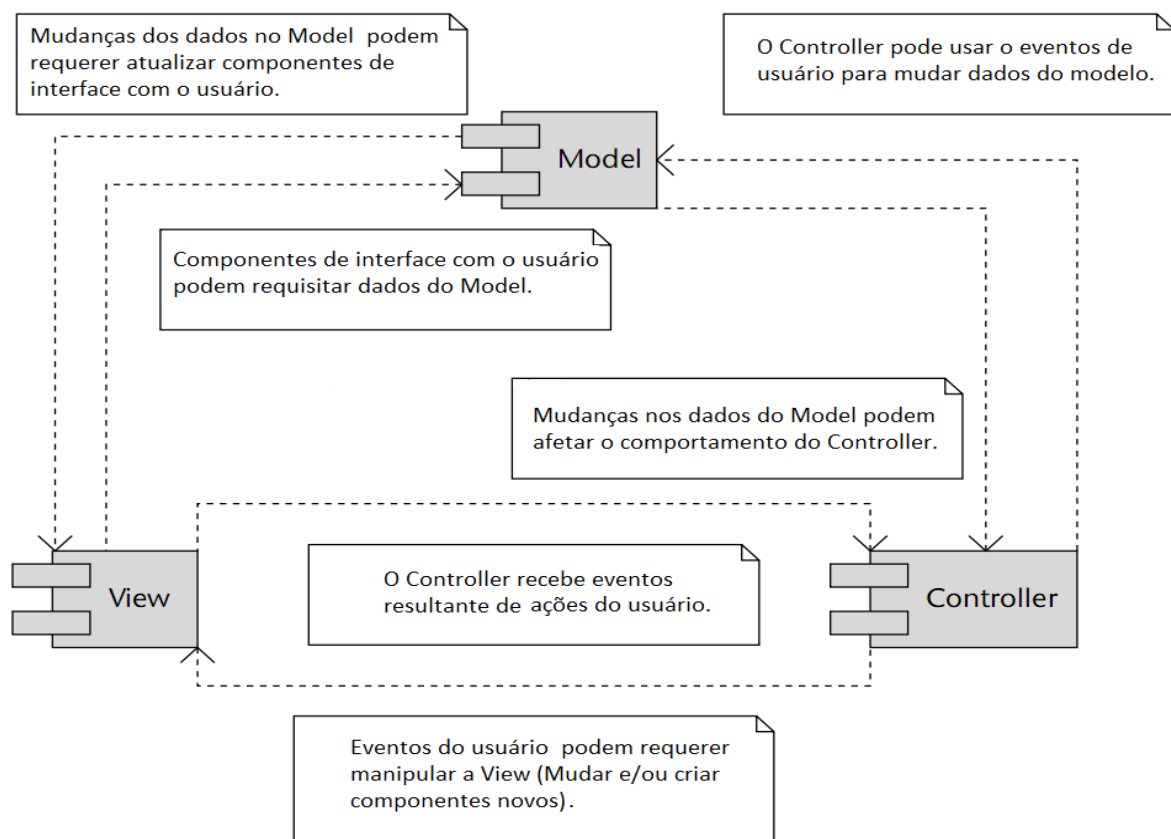


Figura 1: Representação das fases do MVC

Fonte: Adaptado de DUDNEY, Et all (2004)

## 2.4. JavaBeans

Antes de iniciar-se o estudo mais aprofundado do *JSF* e *Spring*, deve-se ficar claro a definição de *Java Beans*, que é um conceito que deve ser entendido por todo desenvolvendo *Java*. Alguns desenvolvedores de aplicações *Java* acham que *Java Beans* são simplesmente classes com algumas propriedades expostas por métodos *get* e *set* (acessores/mutatórios). Por exemplo, uma classe *Java* os métodos *getNome* e *setNome* representam uma propriedade de leitura-escrita chamada nome. Porém, propriedades são apenas uma parte do que um *Java Bean* pode ter.

Segundo Mann (2006), os *JavaBeans* são "componentes reutilizáveis de *software* que podem ser manipulados visualmente com a ajuda de uma ferramenta de desenvolvimento". Isto significa que há mais coisas que apenas propriedades. *Java Beans* conforme um conjunto de padrões, permite a implementação de tratamento de eventos e podem contar até com serviços de persistência.

Entender o poder de *Java Beans* ajudará a compreender o poder do *JSF*. Como *Swings* componentes, todo componente *JSF* é baseado em *JavaBeans* e, componentes do faces são projetados para trabalhar com backing beans— objetos que são implementados como *Java Beans* e também possuem tratadores de eventos.

Se o objetivo for apenas desenvolver uma aplicação em *Java*, somente o conhecimento básico dos métodos acessores e mutatórios são suficientes. Mas quando deseja-se trabalhar com desenvolvimentos de seu próprios componentes, entender os *JavaBeans* a fundo tornará a vida do desenvolvedor mais fácil.

## 2.5. Java Server Faces

O *JSF* (*Java Server Faces*) é um framework *Java* que está em foco atualmente. Podemos considerá-lo uma novidade no mundo dos frameworks web, que apesar de ser lançado desde 2001 no *JavaOne*, prometendo estender a natureza dos componentes gráficos do *swing* e *awt*, ficou algum tempo sem produzir resultados deixando a comunidade a achar que era só fogo de palha. Foi quando que, em 2002, McClanahan (criador do *Jakarta Struts*) juntou-se a equipe como líder de especificação e tudo mudou.

Em fevereiro de 2004, depois de uma longa espera, foi lançada a especificação do *JSF1.0* e em maio do mesmo ano já foi lançada a versão 1.1 do *JSF* e uma outra atualização em

agosto de 2005. A importância do *JSF* é tanta que hoje ele se tornou a plataforma padrão de desenvolvimento web e está sendo adotado por um grande número de desenvolvedores ao redor do mundo. Com tanta atenção dada ao *JSF*, é de se esperar que permita a integração com os diversos frameworks existentes no mercado e o que mostraremos aqui são os benefícios trazidos quando se colocam *JSF* e *SPRING* para trabalharem juntos, Walls (2008).

### 2.5.1. Por que *JSF*?

No que se trata de desenvolvimento de software em geral ou no caso específico de aplicações Web, existem duas abordagens de desenvolvimento de software:

1. *Rapid Application development* (RAD) como no ASP.NET, na qual pode-se clicar no componente e arrastá-lo para o local desejado, eliminando assim detalhes óbvios de implementação.
2. *Hard-Core coding*, na qual o desenvolvedor escreve todo o código manualmente para obter um maior desempenho no *backend*, como é o caso do JEE (*Java Enterprise Edition*).

Os desenvolvedores de software têm então a árdua tarefa de optar por uma dessas metodologias. Claro que eles desejam as duas coisas, a eliminação da programação tediosa provida pelo ASP.NET juntamente com a escalabilidade, portabilidade e suporte por vários fabricantes do JEE. A promessa do *JSF* é permitir desenvolvimento rápido de GUI (*Graphic User Interface*) para aplicações Web em Java.

Se você está familiarizado com aplicações Swing o *JSF* fornece uma biblioteca de componentes prontos para ser usado como no RAD, mas se você trabalha com JSP (*Java Server Pages*) o *JSF* elimina muito código tedioso que antes era necessário fazer manualmente, como navegação entre páginas e validações.

Existe uma gama de IDE's (*Integrated Development Environments*) que dão suporte ao *JSF* com ambientes visual de desenvolvimento e também completando código na hora do desenvolvimento. Vamos, por conseguinte, detalhar os benefícios do *JSF* para que você possa adotá-lo como uma tecnologia de desenvolvimento Web.

## 2.5.2. Principais componentes JFS

Toda tecnologia tem os seus termos, assim, para entrarmos no pontos principais do *JSF* primeiramente mostraremos termos chaves do *JSF*.

**Quadro 1: Termos chaves do *JSF***

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

Navegação	A capacidade de passar de uma página para a próxima. <i>JSF</i> possui um poderoso sistema de navegação que é integrado com os ouvintes dos eventos.
-----------	--

Fonte: Traduzido de Mann, 2005. 39

### 2.5.3. UI Componentes

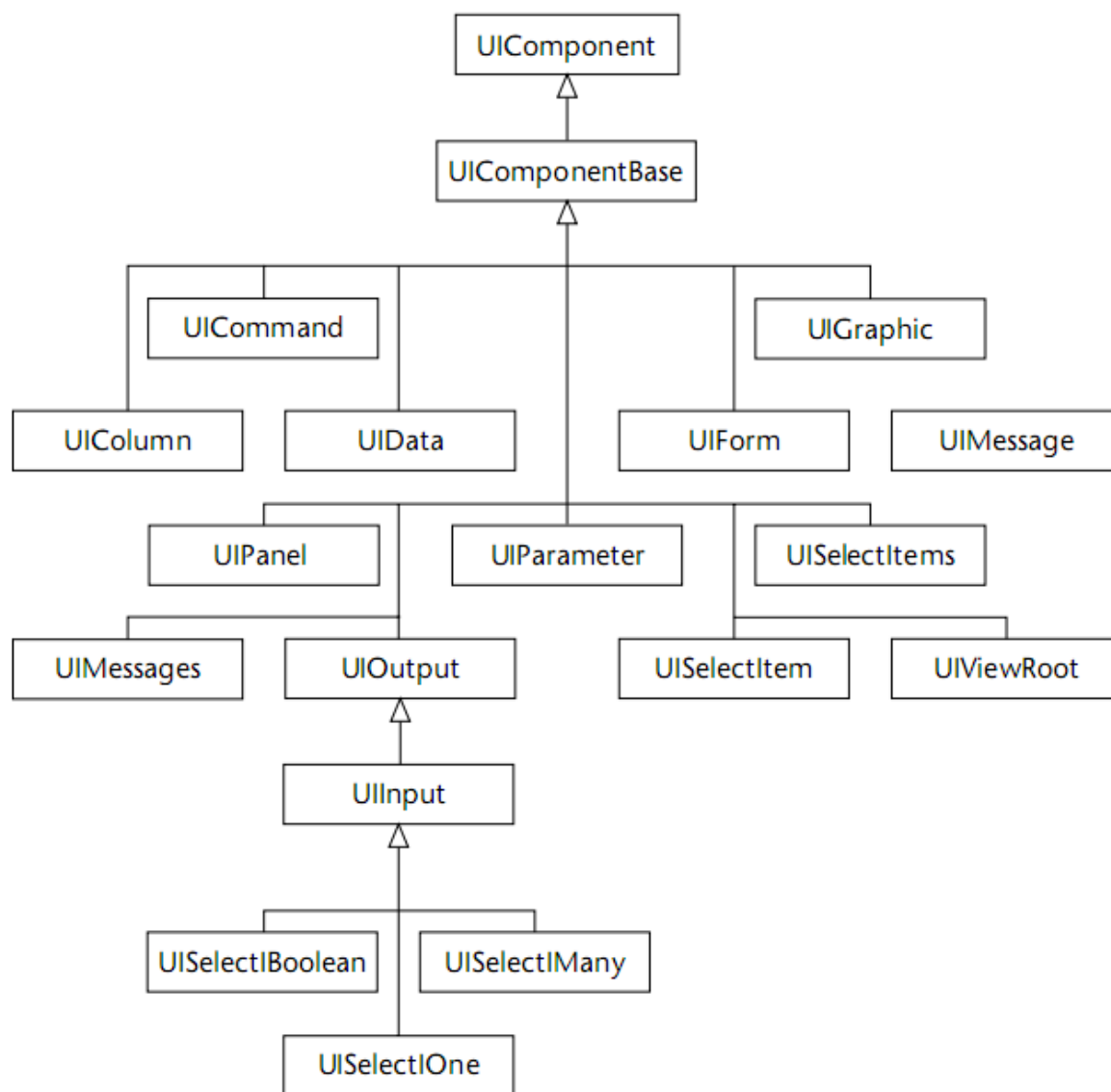
Componentes de interface do usuário é a peça central do *JSF* e são usados como blocos de construção para criar interfaces de usuário que vão do simples ao complexo. Você provavelmente está familiarizado com componentes *Swing* de interface do usuário, bem como os de outras linguagens ou ambientes de desenvolvimento como *Delphi* e *Visual Basic*. Como clientes *Swing*, *JSF* fornece um conjunto de componente de interface de usuário padrão, mas este é direcionado para a web. Esta padronização promete um poderoso e produtivo ambiente visual de desenvolvimento de aplicações Web com uma biblioteca rica em componentes de interface de usuário como calendários, árvores e tabelas, Mann (2005).

*JSF* também fornece uma série de outros tipos de componentes padrões que desempenham um papel de apoio aos componentes de interface do usuário. Estes incluem conversores, validadores, processadores, entre outros. Como os componentes de interface do usuário, estes componentes são intercambiáveis e podem ser reutilizados em toda a aplicação ou em outras aplicações, o que facilita o reuso.

Os componentes padrões *JSF* fornecem um número de componentes básicos que você precisa para construir suas aplicações web simples. Porém, na maioria dos casos, você vai usar um desses componentes padrão ou os de bibliotecas de terceiros, de modo que você normalmente não precisará se preocupar com a *UIComponent* ou *UIComponentBase*. No entanto, haverá momentos em que você talvez queira estender um componente existente ou que tenha necessidade de um componente que não é facilmente acessível, Mann (2005).

#### 2.5.3.1. Componentes UI padrões

Podemos observar na Figura 2 a hierarquia de componentes padrões do *JSF*:



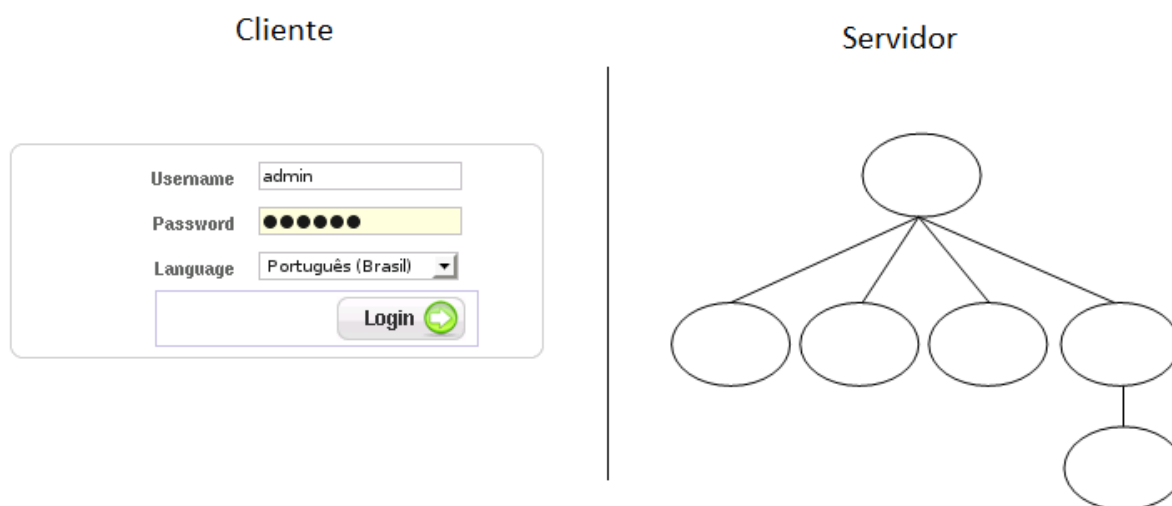
**Figura 2: Hierarquia dos componentes do JSF.**

Fonte: DUDNEY, et all (2004)

No *JSF*, a função é separada da aparência. Em outras palavras, um componente como *UInput* realmente representa uma classe de componentes de interface de usuário que você está acostumado a lidar, os inputs. Todos eles são funcionalmente semelhantes, mas aparecem de maneira diferente. Por exemplo, um campo de entrada de texto simples é funcionalmente semelhante a um campo de entrada de senha. Eles são representados pelo mesmo tipo de componente em *JSF*, mas eles têm processadores diferentes a eles associados. Este é um conceito simples, mas importante para compreender, especialmente se você está considerando desenvolver seus próprios componentes de interface do usuário. Desenvolver componentes de acordo com a função, em seguida, criar processadores para especificar diferentes visões que o usuário necessita.

### 2.5.4. Árvore de componentes

Os componentes de interface do usuário no *JSF* são combináveis. Em outras palavras, você pode aninhar um componente dentro de outro de forma a criar interfaces de usuário mais complexas. Essas composições são referidas como árvores de componentes do *JSF*.



**Figura 3: Representação de um componente no cliente e no servidor**

O *JSF* permite ao programador modificar um componente de interface com usuário no próprio servidor manipulando apenas código *Java*. Para isso ele precisa alterar as propriedades da árvore de componentes. Essa árvore é originada da classe padrão de componentes UI do *JSF* *UIComponent* do pacote *Javax.faces.component*, aos quais todos os componentes UIs do *JSF* precisam estender.

### 2.5.5. Renderizadores

Componentes UI's padrões são responsáveis por renderizar seus próprios componentes. Segundo Craig (2008), quando um componente faz a sua própria renderização, esta denominada de **modelo de implementação direta**. No entanto, *JSF* permite o **modelo de**



**implementação delegada**, na qual esse papel é delegado à outra classe. Estas classes são chamadas de Renderizadores.

Renderizadores são organizados em *Render kits*, na qual cada kit foca em um tipo de saída. Renderizadores contam com um render kit para HTML 4.01, mas um *render kit* poderia gerar vários tipos de saídas com diferentes aparências, diferentes dispositivos com telas de vários tamanhos, *Java* applets, aplicações *Java*, etc.

Um renderizador é como um tradutor entre os diversos servidores e os mais variados clientes. Quando um usuário faz uma solicitação, o renderizador decodifica os parâmetros e ajusta o componente de acordo com os parâmetros recebido.

Quando é usado o modelo de implementação delegada, alterar a visualização de uma página é simples. Basta alterar o *render kit*, os componentes são os mesmo, mas a visualização na tela será diferente de acordo com o dispositivo, ou *modelo* deseja de visualização.

É interessante notar que para tarefas de desenvolvimento de aplicações com *HTML*, renderizadores são muito transparentes. Todos os componentes padrões estão associados com um renderizador, nos bastidores, para que você não precise se preocupar muito com eles.

No entanto, quando precisa-se mudar dinamicamente a aparência de uma aplicação, ou quando desenvolve-se componentes personalizados, os renderizadores são uma parte essencial e poderosa do *JSF*.

### **2.5.6. Validadores**

Uma das alegrias dos desenvolvedores de interfaces de usuário é ter certeza que o usuário digitou a coisa certa. Muitas vezes, validar a informação requer muitas declarações escritas em *JavaScript*, *Java*, ou ambos. Além disso, o processo de indicação de erros é um pouco propenso a erros, se você não tiver uma estrutura para ajudá-lo. A verificação da correteude dos dados fornecidos pelo usuário é um aspecto importante de todas as aplicações que coletam informações (não apenas as aplicações Web), mesmo que essa informação só seja usada temporariamente para realizar um cálculo de algum tipo. Ser capaz de capturar dados mal formados ou incompatíveis é fundamental para assegurar a integridade da informação e para que o aplicativo execute conforme o esperado. Então, não é nenhuma surpresa que a validação de dados é uma parte integrante da estrutura do *JSF*.

*JSF* permite validação de três maneiras, Mann (2005): no nível do componente interface do usuário, através de métodos de validação dos *JavaBeans*, ou em classes de validação. Componentes de interface do usuário geralmente precisam de validação simples, como se um valor é exigido, ou lógica de validação que é específico para o componente em si (e, portanto, não utilizável com outros componentes). Métodos validadores são úteis quando você precisa validar um ou mais campos em um formulário (e se você não precisa compartilhar essa lógica com outros componentes).

Validadores externos são úteis para casos genéricos como o comprimento de um campo ou um intervalo de números que são plugáveis, o que significa que você pode anexar um ou mais deles a qualquer componente. A validação é feita no servidor, porque evita o problema de que nem todos os clientes suportam scripts. (Componentes *JSF* suportam validação no cliente, mas nenhum do componente padrão o faz)

Quando o validador encontra um erro, como uma sequência que é muito longa ou um número de cartão de crédito inválido, ele adiciona uma mensagem de erro para a lista de mensagens atuais. Isto torna mais fácil para mostrar erros de validação de volta para o usuário, utilizando componentes padrões *JSF*.

### **2.5.7. Backing beans**

Às vezes, é conveniente criar um *bean* que contém algum ou todos os componentes de um formulário web. Este recebe o nome de *Backing Beans*.

Quando discutimos o padrão MVC, ficou claro que o *Model* é representado pelos dados e regras de negócios, a *view* são as interfaces vistas pelos usuários e o controle faz a conexão entre os dois.

Mann (2005), define *Backing Beans* como uma especialização do *JavaBeans* que coleta valores a partir de componentes da UI e implementam métodos dos eventos dos ouvintes. Eles também podem possuir referências a elementos da UI. Então, os *backing beans* geralmente contém propriedades que precisam ser recuperadas dos usuários e métodos ouvintes. Os ouvintes podem processar propriedades, manipular componentes UI ou processar regras de negócios da aplicação.

Ele também afirma que os *Backing Beans* são associados a propriedades dos componentes UI. Através dessa associação há uma sincronização entre o componente e o

backing bean. Se o componente for alterado pelo usuário o *Backing Bean* também terá seu valor atualizado.

Conceição (2008) fala que outra operação que pode ser feita, é associar diretamente uma propriedade do backing bean com uma instância do componente no lado do servidor, utilizando a propriedade 'binding'.

### 2.5.8. Conversores

Aplicações web capturam dados do usuário como parâmetros de uma requisição do protocolo HTTP. Tais parâmetros estão em formato de string, enquanto os *Backing Beans*, que representam os dados do sistema, estão em formato de objetos *Java*. Os componentes podem estar associados a propriedades de um back bean e podem ser de qualquer tipo, desde tipos padrões da linguagem *Java* (tais como *String*, *Date*, *int*) até algum tipo definido pelos desenvolvedores da aplicação, Mann (2005). Então, segundo Conceição (2008), o papel dos conversores é fazer a conversão para o *modelo* de objeto adequado, felizmente *JSF* possui suporte para reuso de conversores via *Converters*.

Para quem já trabalhou com desenvolvimento web é mais fácil entender o problema que os *converters* vieram resolver. Em JSP, por exemplo, quando se recebe um valor de data para ser processado pelo sistema, é muito trabalho para o programador fazer essa conversão para um tipo *Date*. Note também que o programador terá que fazer a operação inversa para mostrar a data na tela do usuário.

Os *converters* vieram para resolver o problema da conversão de dados, deixando transparente para o programador esses dois trabalhos. O *JSF* já possui um conjunto de *converters* pré-definido e que já trabalho com os tipos de dados mais comuns, mas permite também que o programador projete seus próprios conversores para tipos que ele mesmo criou.

Outro benefício dos conversores do *JSF* é que eles podem lidar com problemas de localização. Por exemplo, uma aplicação que é mundialmente acessada e tem um campo de data vai ser digitado de uma maneira nos Estados Unidos (mm/dd/aaaa), já no Brasil seria outro formato (dd/mm/aaaa). Isso muda tudo na hora da conversão manual como é feito no JSP, mas o *JSF* já mecanismos de identifica e fazer a conversão adequada dos dados.

### 2.5.9. Mensagens

Tudo que falamos sobre *JSF* é interessante, mas o que acontece quando algo dá errado? Mann (2005) afirma que uma das maiores preocupações quando se desenvolve componentes UI é informar as mensagens de erro ao usuário. Ele fala também que os erros podem ser divididos em duas classes: erros de aplicação (regras de negócios, banco de dados, ou erros de conexão, por exemplo) e erros de entrada (tais como texto inválido ou campos vazios).

Erros de aplicação geralmente são apresentados em uma página com o erro especificado; erros de entradas são mostrados na mesma página que foi digitada ao lado do campo com erro. Porém, mensagens não necessariamente precisam indicar erros; elas podem apresentar algum tipo de informação adicional para o usuário. Por exemplo, um ouvinte pode adicionar uma mensagem indicando que um novo registro foi armazenado no banco de dados.

Uma mensagem pode ser associada a um componente UI. É possível associar mensagens de erros a um componente através do componente *HtmlMessage*.

No *JSF* existe um sistema de mensagem para avisar o usuário sobre erros. Essas mensagens podem ser geradas em diversos idiomas automaticamente. Geralmente as mensagens do *JSF* são curtas e bem explicativas.

### 2.5.10. Navegação

Todos os conceitos aqui relatados envolvem a navegação entre diversas páginas. Quando construímos aplicações web ocorre que elas são compostas de várias páginas distintas e é necessária uma forma de navegar nelas de forma eficaz. O fato de mover-se entre as páginas é chamado pelo *JSF* de **navegação**.

*JSF* tem uma forma elegante de navegação. O *navigation handler* é uma classe do *JSF* responsável por decidir para qual página deve ser carregada baseada no resultado obtido na requisição. Para cada página, a regra de navegação define qual página carregar. Estas regras de mapeamento são chamadas de *navigation case*. Essas regras são definidas no arquivo de configuração do *JSF*.

Como podemos ver as regras de navegação não são estáticas, elas se comportam conforme o resultado que foi obtido. Esta funcionalidade do *JSF* ainda requer escrever muito código, mas como elas são centralizadas em um único arquivo e tem sintaxe fixa, não é

complicado gerenciar as regras de navegação. Hoje as IDEs *Eclipse* e *NetBeans* já têm ferramentas gráficas para editar essas regras de navegação, aumentando o nível de transparência da programação com *JSF*.

### 2.5.11. Ciclo de vida

Segundo Conceição (2008), o *JSF* possui um ciclo de vida de requisição baseado em fases, que são executadas seguindo uma ordem. Para um desenvolvedor não é obrigatório que ele conheça a fundo o funcionamento interno do *JSF*, mas é um diferencial importante ter um conhecimento sobre o ciclo de vida de uma aplicação *JSF*, sabendo a ordem e o que acontece em cada uma destas fases. Este conhecimento permite que sejam construídas aplicações melhores, pois o desenvolvedor sabe com detalhes quais operações estão acontecendo em cada fase, tornando mais fácil inclusive detectar um erro, sabendo em qual fase ele pode ter ocorrido. Podemos entender melhor este ciclo de vida observando a Figura 4.

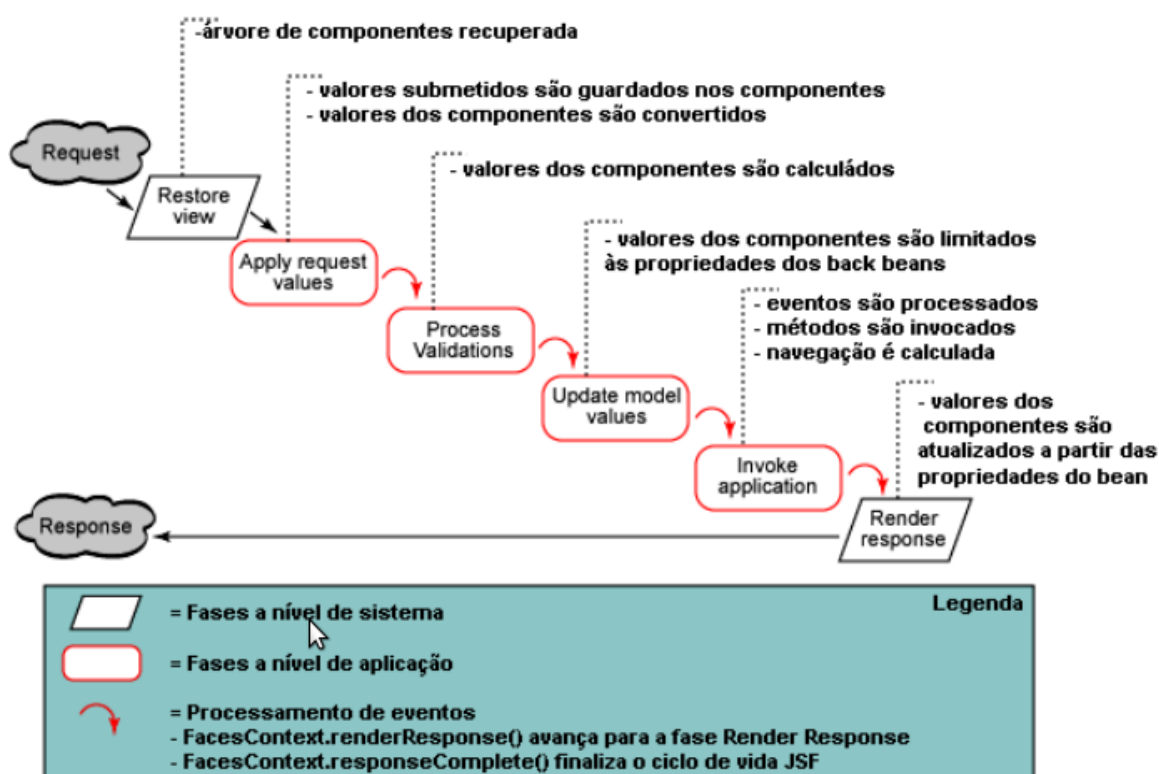


Figura 4: Ciclo de vida de uma requisição *JSF*.

Fonte: Conceição (2008)

Segundo Conceição (2008), uma das vantagens deste mecanismo de fases é permitir que um desenvolvedor concentre seus esforços em determinadas fases. Por exemplo, quem for trabalhar com desenvolvimento de componentes devem focar nas fases *restore view* e *render response*, o que facilita o trabalho é o velho dividir para conquistar.

#### **2.5.11.1. Restore view**

Depois de uma requisição de um cliente, o *JSF* tentará restaurar a árvore *view* correspondente. Conceição (2008) define uma *View* como uma árvore com todos os componentes de uma página. O *JSF* verifica primeiro, através de um *id*, se a *view* já existe. Se existir, então ela será restaurada do estado anterior. Se não existir, uma nova *view* será criada pelo *FacesContext* com os parâmetros contidos na requisição.

Ao final dessa fase a referência da *view* é passada para o *FacesContext*, que será a *view* restaurada ou a *view* criada com os valores padrões. Se existir parâmetros *POST* no corpo da requisição, serão puladas todas as fases do ciclo de vida indo direto para a *Render Response Phase*.

#### **2.5.11.2. Apply request values**

Os valores digitados pelos usuários serão extraídos e aplicados aos seus respectivos componentes no lado servidor, se o valor digitado não estiver de acordo com o esperado pelo componente, ocorrerá um erro que vai ser adicionado na classe *FacesContext* e mostrado na *Render Response Phase*.

#### **2.5.11.3. Process validations**

Na fase *process validation* serão executados todos os validadores da *view*, eles vão aplicar regras de validação ao valores digitados. Se houver erros, será adicionada uma mensagem de erro em *FacesContext* e o fluxo de execução vai ser direcionado para a fase *Render Response* e será mostrado ao usuário uma tela com as mensagens de erro.

#### **2.5.11.4. Update model values**

Quando chegamos nesse ponto, temos garantido que os valores foram digitados corretos. Então, será atualizado o status da *view* gravados nos componentes (O status da *view* é

no lado servidor). Se houver erros de conversão de tipos, um erro vai ser adicionado em *FacesContext* e o fluxo vai ser direcionado para a fase *Render Response Phase*.

#### **2.5.11.5. Invoke application**

Nesta fase são executados os métodos dos *backing beans* e as regras de navegação são aplicadas. E temos o encerramento do fluxo de execução da requisição.

#### **2.5.11.6. Render response**

Nessa fase, é mostrada para o usuário a página. Aqui sabemos que todo o ciclo de vida foi concluído e agora será mostrada na tela a resposta para o usuário. Caso haja as *tags message* na página e tenha havido erro durante no ciclo de vida da requisição, estes erros serão mostrados na página.

Segundo Conceição (2008), mostrar o resultado ao usuário é a principal função dessa fase, mas ela também grava o estado da *view*, para que ela possa ser recuperada na fase *restore view* caso seja necessário.

## **2.6. Spring framework**

O *Spring* é um framework de código aberto criado por Rod Johnson e descrito em seu livro, *Expert One-on-One: J2EE Design e Development*. Foi criado para lidar com a complexidade de desenvolvimento de aplicativos corporativos. O *Spring* torna possível usar simples *JavaBeans* para conseguir coisas que só eram possíveis com EJBs. Porém, a utilidade o *Spring* não é limitada ao desenvolvimento do lado servidor. Qualquer aplicativo *Java* pode se beneficiar do *Spring* em termos de simplicidade, testabilidade e baixo acoplamento (Craig Walls,2008).

O *Spring* é um framework leve que em média possui 2,5MB e é distribuído em arquivo de formato jar. Ele é considerado um *container* no sentido de que gerencia o ciclo de vida e a configuração de objetos do aplicativo. Ele também fornece muitas funcionalidades de estrutura como gerenciamento de transações, integração com frameworks de persistência de objetos, etc, deixando o desenvolvimento da lógica da aplicação para o programador.

Segundo Craig (2008), um dos conceitos chaves do *Spring* framework é a injeção de dependência (DI) e a programação orientada a aspectos. A injeção de dependência é um

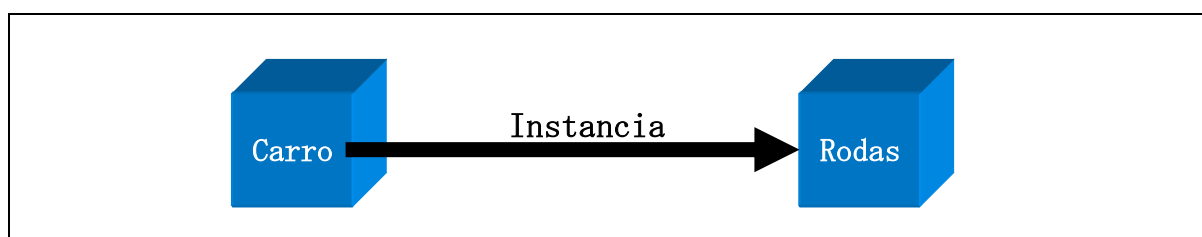
mecanismo que promove baixo acoplamento. Quando a DI é aplicada, ocorre que, ao invés de os objetos que dependem de outros buscarem suas dependências, o próprio *Spring* as injeta eliminando esse acoplamento existente no modo natural de se resolver tais dependências. Outro aspecto é a programação orientada a aspecto (AOP), que visa separar a lógica da aplicação de serviços de sistema como auditoria, gerenciamento de transações, etc.

### 2.6.1. Injeção de dependência

A injeção de dependência (DI) está no coração do *Spring Framework*. O nome assusta alguns, mas a DI não é tão complexa quanto se pode imaginar. Usando DI é possível obter código mais simples, fácil de entender e de testar.

A injeção de dependências inicialmente era conhecida como inversão de controle. Porém, em 2004, Martin Fowler questionou qual aspecto do controle era realmente invertido e concluiu que era a aquisição de dependências que era invertida. Então o termo injeção de dependência reflete melhor o que está acontecendo.

Quando estamos fazendo qualquer aplicação que difere-se de um *helloWorld*, temos no mínimo uma associação (*wiring*) entre duas classes para implementar alguma lógica de negócios, mesmo que simples. No *modelo* tradicional de desenvolvimento cada objeto que necessite de uma dependência é responsável por instanciá-la o que gera código acoplado e difícil de testar.

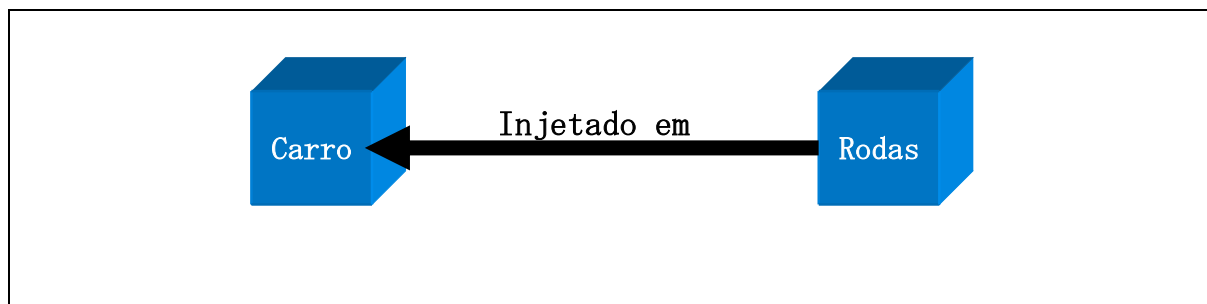


**Figura 5: Aquisição de dependência no modelo tradicional.**

Fonte: Próprio autor

Aplicando DI os objetos recebem suas dependências em tempo de criação por algum mecanismo que controla os objetos do sistema, mostraremos posteriormente como isso é feito na prática.





**Figura 6: Aquisição de dependência com DI.**

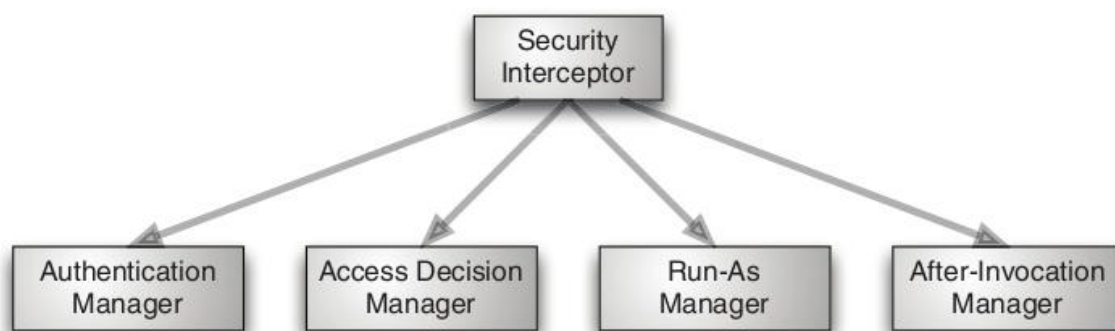
Fonte: Próprio autor

### 2.6.2. Spring security

Segundo Craig (2008), o *Spring Security* é um framework de segurança que fornece segurança declarativa para suas aplicações com base em *Spring*. Ele fornece uma abrangente solução de segurança, tratando de autenticação e autorização, no nível de solicitação web, e no de chamada de método. *Spring Security* tira vantagens da injeção de dependências e orientação a aspectos do *Spring Framework*.

O *Spring Security* também é conhecido como *Security Acegi* (ou simplesmente *Acegi*). *Acegi* é um subprojeto do *Spring*, mas os planos são de trazê-lo para mais perto do *Spring*, e existe um movimento que muda o nome do *Acegi* para *Spring Security*. Então, o trataremos de *Spring Security*, no entanto existem literaturas que ainda utilizam o nome *Acegi*.

Para assegurar aplicações corporativas, *Spring Security* utiliza filtros de *servlets* que interceptam pedidos do *Servlet* para executar a autenticação e reforçar a segurança. Quando é feita uma requisição web, esta passa pelos filtros de segurança do *Spring Security* para que possa ser liberada ou não. Segundo Craig (2008), se você precisa assegurar o nível de solicitação web ou se requer segurança de métodos de baixo nível, *Spring Security* emprega cinco componentes centrais que reforçam a segurança de uma aplicação, como mostrado na Figura 7.



**Figura 7: Membros fundamentais do *Spring Security***

Fonte: Mann (2005)

### **2.6.2.1. Interceptadores de segurança**

Quando você chega em casa, é preciso colocar a chave para destravar a fechadura da porta que te dar acesso ao seu lar, e esta deve ser aceita para que a fechadura libere o acesso. Em *Spring Security*, os interceptadores de segurança trabalham de forma análoga a esta situação e pode ser imaginado como uma trava que evita o acesso a um recurso de uma aplicação. Para inverter a trava e passar pelo interceptador de segurança, deve-se ser fornecida uma “chave” (geralmente usuário e senha) no sistema. Então, a chave tentará abrir a “fechadura” do interceptador de segurança para permitir o acesso ao recurso assegurado.

Um interceptador de segurança faz pouco mais que interceptar acessos a recursos para reforçar a segurança. Na verdade ele não aplica regras de segurança, delega esta responsabilidade aos vários gerenciadores que são ilustrados na parte de baixo da Figura 7.

### **2.6.2.2. Gerenciador de autenticação**

A primeira fechadura de interceptador de segurança a ser destravada é o gerenciador de autenticação. Segundo Craig (2008), o gerenciador de segurança é responsável por determinar quem você é, utilizando para isso, na maioria dos casos, um usuário e uma senha. O *Spring Security*, assim como o *Spring*, é baseado em componentes plugáveis baseados em interfaces, sendo assim pode trabalhar com uma grande quantidade de provedores de autenticação.

### **2.6.2.3. Gerenciador de decisões de acesso**

Uma vez que o *Spring Security* determinou quem você é, deve-se agora decidir se o usuário tem acesso ao recurso assegurado. O que foi feito anteriormente foi apenas para saber quem é o usuário que está acessando a aplicação. Craig (2008) descreve um gerenciador de decisões de acesso como sendo a segunda fechadura da trava do *Spring Security* a ser liberada. Ele executa a autorização, decidindo se o usuário tem permissão de acesso ao recurso assegurado em questão.

Com este gerenciador pode-se definir, por exemplo, que um determinado recurso seja acessado apenas por supervisores. Se por exemplo, o usuário tiver privilégios de supervisor, será liberado o acesso ao recurso desejado. Segundo Craig (2008) está é a segunda e última fechadura que depois de destravada, o interceptador de segurança é removido permitindo acesso ao recurso.

### **2.6.2.4. Gerenciador de execução (Run-as)**

Se o usuário passar pelos gerenciadores de autenticação e de decisão de acesso então o interceptador de segurança será destravado e a porta estará pronta para ser aberta. Mas antes de girar a maçaneta e entre, há mais uma coisa que o interceptador de segurança pode fazer.

Mesmo que o usuário tenha passado pela autenticação e garantido o acesso a um recurso, pode haver mais restrições atrás da porta. Por exemplo, o usuário pode ter acesso a uma página web e não ter acesso aos componentes que são usados para criar essa página tem requisitos de segurança diferentes que o da página. Um gerenciador de execução (run-as) pode ser usado para recolocar a autenticação do usuário de modo que ele tenha acesso aos objetos assegurados que são mais profundos em seus aplicativos. Nem todos os aplicativos precisam de uma substituição de identidade. Portanto, gerenciadores de execução são opcionais e é pouco usado em aplicações *Spring Security*, Walls (2008).

## **2.7. Programação orientada a aspectos**

Sáímos no nível das linguagens de máquina para a programação OOP passando ainda pela programação procedural e atualmente estamos em um nível mais elevado que a algumas década atrás. Hoje não preocupamos mais com instruções de máquinas e como os objetos colaboram entre si para se formar uma aplicação final.

Segundo Chavez (2009), o desenvolvimento de um novo paradigma de engenharia de software frequentemente progride da metodologia de programação em direção às metodologias de projeto e análise, fornecendo um caminho completo pelo ciclo de vida de desenvolvimento de software, alcançando assim a maturidade. A programação orientada a aspectos (POA) após uma década de pesquisas encontra-se nesse ponto, resultando em ferramentas eficientes, uma comunidade de usuários capacitados e aplicações iniciando a sua trajetória para o mercado.

A programação orientada a aspectos foi criada no ano de 1997, nos laboratórios da Xerox, por Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier e John Irwin. Segundo Goetten & Winck (2006), o objetivo era construir uma abordagem que fosse um conjunto não necessariamente homogêneo, que permitisse à linguagem de programação, composta por linguagem central e várias linguagens específicas de domínio, expressar de forma ideal as características sistêmicas (também chamadas de ortogonais ou transversais) do comportamento do programa. A essa abordagem dá-se o nome de meta-programação.

## 2.8. Integração entre JSF e Spring

O que ocorre na maioria das vezes, é que os programadores adquirem certos gostos por um determinado framework e isso o torna bom trabalhando em determinado framework. Mesmo tendo ouvido vários benefícios do *Spring*, você não se sente seguro ainda para usar os recursos do *Spring*. Entretanto, muitas vezes queremos usar recursos novos como DI, POA, etc, sem abandonar totalmente a plataforma atual de trabalho.

Uma das características do *Spring* é permitir isso. Terá liberdade para optar pelo que funciona melhor no seu aplicativo. O próprio *Spring* oferece um framework web, mas você terá a liberdade de escolher entre os principais frameworks web como *Struts*, *WebWork*, *Tapestry*, *JSF*, podendo ainda tirar proveito do *Spring* nas outras camadas da aplicação. Então, trataremos aqui da integração entre *Spring* e *JSF*.

Como já citamos, o *JavaServer Faces (JSF)* já vem com algum suporte a injeção de dependências, mas lembre-se que o *Spring* oferece muitos outros recursos como controle de transações, segurança, acesso remoto, etc.

Segundo Craig (2008), mesmo que o *JSF* destina-se à camada de apresentação, os componentes das camadas de serviços e de acesso a dados frequentemente são declarados em

um arquivo de configuração *JSF*. E que de certa forma isso não parece apropriado. Então, é melhor separar as responsabilidades e deixar o *JSF* com a camada de apresentação e o *Spring* com o resto do aplicativo.

Antes de começar a integração, é importante mostrar como o *JSF* determina as variáveis em suas páginas sem o *Spring*. Vamos então revisar o funcionamento do *JSF* se o *Spring* e iniciaremos a integração. A integração é mais simples do que se possa imaginar, mas os benefícios são muitos.

O *JSF* internamente usa uma variável determinadora para procurar em seu arquivo de configuração (*faces-config.xml*) os *managed beans*.

Para integração com o *Spring*, gostaríamos que o *JSF* configurasse suas variáveis a partir do contexto do *Spring*. Para isso temos que substituir a variável determinadora do *JSF* com uma variável determinadora que enxergue o *Spring*.

A *DelegatingVariableResolver* do *Spring* é tal variável. Em vez de determinar variáveis apenas entre os *managed beans* controlados pelo *JSF*, a *DelegatingVariableResolver* também as procura no contexto do *Spring*.

Quando o *JSF* precisa determinar uma variável, a *DelegatingVariableResolver* olhará primeiro para a variável determinadora original. Se um bean controlado pelo *JSF* apropriado puder ser achado, então ele será usado. Senão, a *DelegatingVariableResolver* procurará no contexto do *Spring* um bean cujo nome seja o mesmo da variável do *JSF*.

### **3. Desenvolvimento de um sistema de controle de TCC para a UESB utilizando JSF e Spring**

#### **3.1. Introdução**

Para a obtenção de qualquer título de graduação, mestrado ou doutorado na Universidade Estadual do Sudoeste da Bahia (UESB), é necessária, além da aprovação nas disciplinas obrigatórias oferecidas pelos cursos, a realização de atividades que têm como resultado a elaboração de um trabalho, podendo este ser um relatório de estágio curricular, o Trabalho de Conclusão de Curso (TCC), etc.

Devido ao número de formandos a cada ano, estes trabalhos acabam sendo perdidos ou esquecidos na biblioteca, ocasionando que as pesquisas são esquecidas e o conhecimento fica perdido.

O controle de entrega destes documentos na UESB está deficiente. Ocorre que não tem como ter acesso a estes devido à dificuldade de armazenamento e recuperação destes trabalhos.

Com a finalidade de organizar esses trabalhos, foi desenvolvido um sistema de informação e-Doc para informatizar o armazenamento e o acesso aos dados dos trabalhos produzidos na UESB.

O desenvolvimento deste estudo de caso também teve como objetivo aplicar os conceitos pesquisados sobre o *JSF* e o *Spring* para que pudessem ser comprovados na prática os benefícios dessas duas tecnologias.

Mostraremos agora os requisitos do e-Doc, sua arquitetura, metodologia de desenvolvimento, e também as dificuldades encontradas na implementação do sistema. Também serão mostrados aspectos relacionados ao *Spring* e *JSF* que foram os assuntos estudados nesse TCC.

#### **3.2. Metodologia de desenvolvimento**

A metodologia adotada para o desenvolvimento do software e-Doc consistiu no levantamento de requisitos por meio de entrevista. Os requisitos foram importantes para a definição de prioridades nas funcionalidades descritas na especificação dos requisitos do sistema.

Do levantamento de requisitos passamos para a formalização deste. Construiu-se então

um quadro identificando os tipos de requisitos e um diagrama de *Caso de uso* para definir responsabilidades de cada ator do sistema. Fez-se então a *modelagem* do banco de dados e a *modelagem* de classes de projeto do sistema. Esta análise permitiu a definição dos módulos apresentados no capítulo 4, agrupando as funcionalidades que deveriam estar presentes no sistema.

Depois de ter feito a análise e a *modelagem* do banco de dados e classes iniciou-se a construção do sistema utilizando como principais tecnologias o *JSF* e o *Spring*. Depois foram feitos testes de inserção, edição, exclusão e busca do sistema. Com o termino dos testes, partiu-se então para a documentação do sistema, que faz parte dos requisitos levantados pelo sistema.

Ainda não foi implantado o sistema na Unidade de Informática (UINFOR).

### 3.3. Requisitos

Os requisitos foram colhidos no colegiado de Ciência da Computação, levantados por meio de entrevista e com o auxílio do aluno do próprio curso (Jorge Farias Herculano).

Foram identificadas três categorias:

1. Essencial (E), indispensável os sistema;
2. Desejável (D), só devem ser descartados devidos a inviabilidades técnicas ou de cronograma;
3. Secundário (S), que não precisam obrigatoriamente se atendido.

Cada requisito foi qualificado com : Funcionais (F) ou Não Funcionais (NF). Foi criado um campo “Implementado” indicando se a funcionalidade foi implementada ou não.

Mostraremos no Quadro 2 os requisitos do sistema:

**Quadro 2: Requisitos do eDoc**

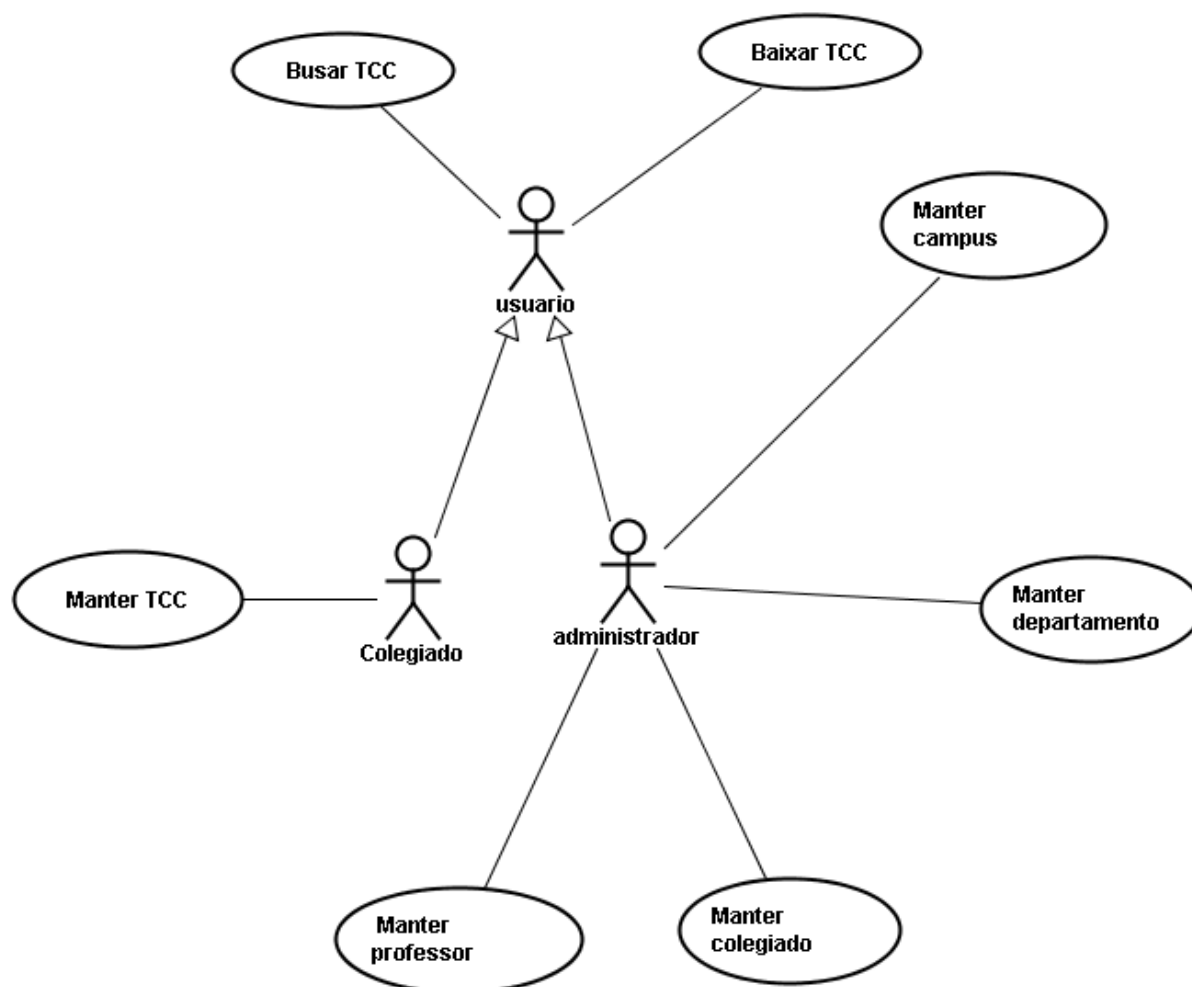
	<b>Requisito</b>	<b>Categoria</b>	<b>Classe</b>	<b>Implementado</b>
1	Níveis de acessos a partes do sistema por usuário.	E	NF	SIM
2	Armazenar dados sobre os trabalhos (Título, banca, pdf, curso, etc)	E	F	SIM
3	Armazenar dados sobre os professores	E	F	SIM

4	Armazenar dados sobre os colegiados.	E	F	SIM
5	Armazenar dados sobre os departamentos.	E	F	SIM
6	Armazenar dados sobre os campi.	D	F	SIM
7	Permitir consultas (busca pelo orientador, título, palavra-chave, departamento, ano, curso, área).	E	F	SIM
8	Cadastro de banca avaliadora para cada trabalho.	E	F	SIM
9	Fazer agendamento de banca.	D	F	NÃO
10	Permitir a emissão de relatórios.	D	F	SIM
11	Funcionar totalmente com <i>software</i> livre.	E	F	SIM
11	Desenvolver um manual do sistema.	E	NF	SIM
12	Realizar a instalação do sistema junta à UINFOR.	E	F	SIM

O agendamento da apresentação de trabalhos não foi implementado devido à falta de requisitos consistentes, mas fica como um trabalho futuro o desenvolvimento.

Com o fim do levantamento de requisito podemos identificar atores e responsabilidades que veremos no diagrama de caso de uso da Figura 8.



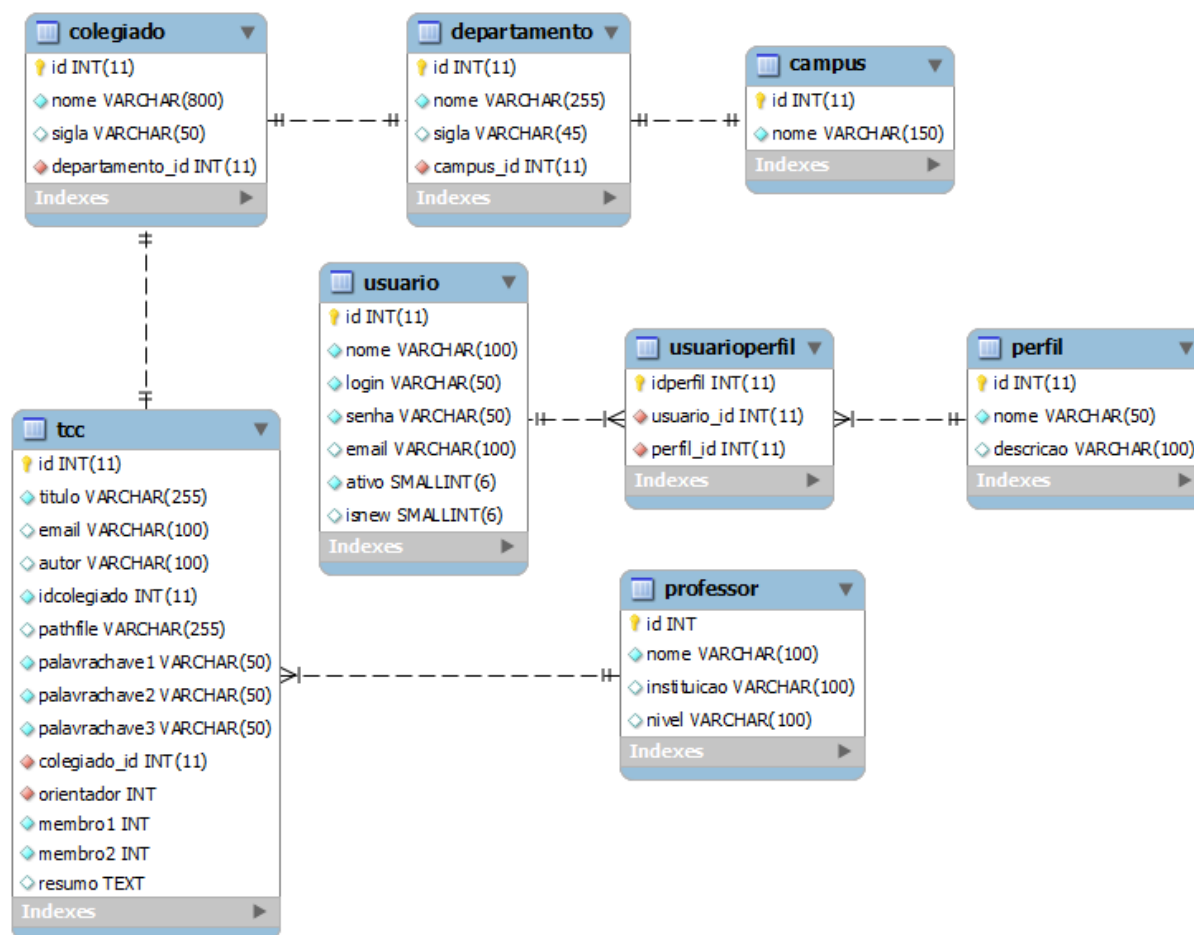


**Figura 8: Diagrama de Caso de uso do Sistema eDoc**

Podemos identificar os atores que irão interagir com o e-Doc. Usuários comuns podem pesquisar e baixar TCC's e não precisam de senha de acesso. O colegiado é responsável por manter os TCC's e o administrador é responsável pelos dados chave que são necessário nos cadastros dos TCC's.

### **3.4. Diagrama do banco de dados**

Vamos mostrar agora como ficou a *modelagem* do banco de dados e como as tabelas relacionam entre si, bem como a funcionalidade de cada uma. Podemos ver na Figura 9 o diagrama do banco de dados:



**Figura 9: Modelagem do Banco de dados**

Foram identificadas nove tabelas no sistema. O banco de dados foi desenvolvido de forma a permitir a implementação dos requisitos levantados.

Inicialmente vale destacar que cada tabela tem um identificador único chamado de **ID**. Na tabela **campus** será registrado o nome do campus, visto que a UESB possui 3 campi. O cadastramento do campus é dependência para que possa ser cadastrado o departamento que possui como chave estrangeira um identificador para campus. Por conseguinte tempo a tabela **colegiado** que cadastra informações do colegiado de cada curso. Vejamos na Figura 9 que colegiado possui uma chave estrangeira de departamento. A tabela **tcc** possui uma chave estrangeira para colegiado ao qual será identificado o curso do TCC. Observe que devido a inexistência de uma tabela curso, durante as busca de TCC's por curso será utilizada a tabela colegiado para identificar o curso referente ao TCC. Temos agora a tabela **banca** que armazena dados sobre o TCC e sobre os professores da banca. A tabela **professor** contém os dados do professor que é usado como chave estrangeira de **banca**.

Temos agora três tabelas que dão suporte para que os requisitos possam ser implementado. As tabelas **usuário**, **perfil** e **usuarioperfil** são responsáveis por armazenar o usuário e senha dos usuário e suas permissões de acesso. Cada módulo do sistema terá um nível de acesso e a cada usuário será atribuído um nível armazenados nessa tabela. Em usuário armazenamos dados como *login* e senha. Em **perfil** armazenamos os tipos de perfis do usuário e em **usuarioperfil** temos a interligação de um usuário com um determinado perfil. Um usuário pode ter vários perfis podendo ser administrador e usuário comum ao mesmo tempo.

### 3.5. Arquitetura do sistema

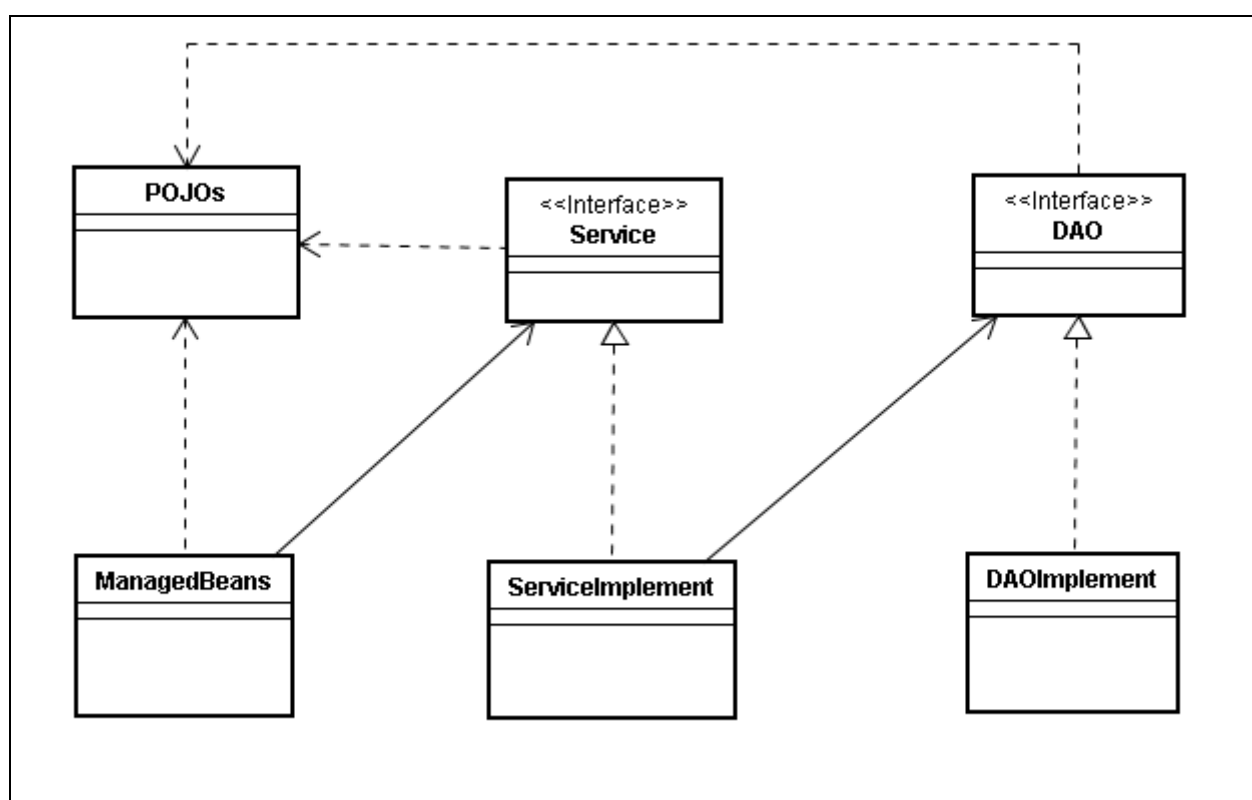


Figura 10: Arquitetura do sistema

O e-Doc foi implementado utilizando a arquitetura padrão do *JSF*. Foi separado a lógica da apresentação, ou seja as regras de negócios estão encapsuladas em classes *Java* e as páginas de interface com usuário foram desenvolvidas utilizando os componentes do *JSF*.







Modularizar o sistema ajuda no desenvolvimento aumentando a velocidade de desenvolvimento e ajuda na manutenção do sistema. Permite também a divisão de responsabilidades e desenvolver a aplicação por partes mais urgentes.

O e-Doc é composto pelos seguintes módulos:

- Interface com o usuário, que define a interface para a entrada e saída de dados, possibilitando a comunicação entre os usuários e o banco de dados. Este módulo é baseado basicamente em CRUD's.
- Autenticação de Usuários, que tem por objetivo realizar a validação dos usuários do sistema;
- Controle de Acesso, que é sempre chamado pelo Módulo de Interface do Usuário antes do acesso a áreas restritas do sistema, a fim de verificar as permissões de acesso dos usuários;
- Acesso ao Banco de Dados, responsável por realizar as ações requisitas pelos usuários no banco de dados;

### 3.5.1. Interface com o usuário

A interface com o usuário foi desenvolvida com componentes *JSF*. Ela é basicamente baseada em CRUD's. um cadastro de professor, por exemplo, possui três partes. A parte de listagem e feita com um componente personalizado do *JSF* e as de edição e cadastro são a mesma página, portanto se comporta de maneira diferente quando for edição ou cadastro do professor. Vemos na Figura 11 um exemplo do componente *JSF* personalizado para listagem:

Listagem de Professores			
Nome	Instituicao	Nivel	+
Cinara Costa Silva	UFBA	Doutorado	 
Denilton Costa Silva	UFS	Graduado	 
Fabricio Costa Silva	UESB	Mestre	 

**Figura 11: Componente com listagem de professores**

Neste componente de listagem temos um ícone representado por um sinal de adição que é usado para chamar a tela de cadastro de novos professores por exemplo. No final da linha de cada registro temos outros dois ícones. O representado pelo lápis é o de edição. Quando clicado nele é aberta uma páginas com os dados para edição como podemos ver na Figura 12.

Formulário de Cadastro de Professores com os seguintes dados:

Nome:	Cinara Costa Silva	✓
Instituicao:	UFBA	✓
Nivel:	Doutorado	✓

Botões: Salvar, Cancelar

**Figura 12: Componente de Cadastro/edição de Professor**

O símbolo verde na frente de cada componente é um recurso configurável do *JSF* que diz que aquele campo é obrigatório. Quando não são digitados dados nos campos obrigatórios o mecanismo de validação do *JSF* entra em ação e aborta a requisição na fase de validação e mostra uma mensagem que podemos ver na Figura 13:

Formulário de Cadastro de Professores com o seguinte estado:

Nome:	Cinara Costa Silva	✓
Instituicao:	UFBA	✓
Nivel:		✗ Campo Obrigatório

Botões: Salvar, Cancelar

**Figura 13: Componente de Cadastro/edição de Professor com erro de validação**

O Segundo ícone na frente de cada registro é um botão para excluir o registro correspondente.

### 3.5.2. Autenticação de usuário

O e-Doc é dividido em dois módulos quanto à autenticação de usuário. Um módulo com senha e outro sem. O módulo acessado pelo colegiado requisita uma senha. Mas o módulo de busca de TCC's pela comunidade acadêmica não necessita de senha.

Quando o usuário do colegiado se autentica com sucesso é restaurado seu perfil de acesso e ele só terá acesso às partes inerentes ao seu perfil. Caso a autenticação do usuário falhe, uma mensagem de erro mostrada.

### 3.4.3. Controle de acesso

O controle de acesso às partes do e-Doc é feito pelo *Spring Security*. O *Spring* tem um ótimo mecanismo de segurança que é cadastrado os módulos em um arquivo com as respectivas regras de acesso. No momento que o usuário clicar em uma página o *Spring Security* intercepta a requisição e verifica se existe no perfil do usuário algo correspondente à regra de acesso. Caso não haja, o usuário é impedido de abrir aquela página.

### 3.4.4. Acesso ao banco de dados

No e-Doc existe um módulo separado para acesso ao banco de dados. É configurada uma conexão pelo *Spring* que é usada por uma classe genérica de acesso a dados. Essa classe implementa todas as operações de acesso e gravação de dados necessárias para todas as classes do sistema. Quando há uma operação muito específica que não exista nessa classe genérica é então implementado a operação na camada de negócios específica da entidade que necessite. Sendo que esta classe de regra de negócio específica de cada entidade herda da classe genérica. Vamos ainda detalhar a arquitetura do e-Doc.

## 3.6. Implementação

Como resultado dos estudos sobre os *frameworks* JFS e *Spring* foi desenvolvido o e-Doc que funciona sob plataforma web. Tendo em vista que ele deveria ser acessado pela comunidade acadêmica e por qualquer pessoa em qualquer lugar, seriam necessárias tecnologias web para satisfazer os requisitos. O *Spring* e o *JSF* resolveram esse problema de forma satisfatória.

A implementação do e-Doc foi feita em plataforma Windows por motivos técnicos do autor sem prejuízo aos requisitos do sistema, utilizando o servidor *Tomcat* 6 em conjunto com o banco de dados *MYSQL*. A escolha do servidor e do banco de dados foi influenciada pela grande aceitação e utilização destes softwares pela comunidade e pelo fato de serem software livres. Tendo em conta que a UESB está migrando toda para software livre tornou-se necessário somente a utilização tecnologias livres.

Durante o desenvolvimento do software, foram realizados testes sobre o comportamento do sistema na plataforma Linux, com o propósito de serem encontrados possíveis problemas de configuração, de modo especial nos métodos que salvam os arquivos

dos TCC's.

Foi utilizado o padrão MVC para o desenvolvimento do software. Com o MVC foi possível separar a lógica da aplicação do desenvolvimento da interface. O sistema está muito bem modularizado e as responsabilidades estão bem organizadas. Cada camada realiza determinados tipos de função.

Outro quesito observado foi a segurança. Utilizou-se o mecanismo de segurança do *Spring*, o *Spring Security*. Ele tem uma forma simples e eficaz de garantir a segurança, tanto na camada de visão, quanto nas demais camadas. Além de permitir o acesso apenas aos módulos cadastrados ao usuário, ele ainda intercepta as requisições e verifica se o perfil é compatível com o do usuário.

## 4. Análise dos Resultados

Levando em conta os objetivos propostos vamos agora analisar os resultados obtidos no desenvolvimento desse trabalho. Os resultados serão analisados sob as óticas da rapidez no desenvolvimento, segurança, padronização e estilização.

Pudemos aplicar todos os conceitos estudados no levantamento teórico, tais como conversores, validadores, componentes UI, injeção de dependência, *Spring Security*, etc. Pode-se perceber que o ganho de tempo no desenvolvimento foi muito bom. Para desenvolver um módulo de cadastro, exclusão, edição e listagem de um TCC, por exemplo, gastou-se em média 10 a 15 minutos, que é um tempo excelente. Então, foram comprovados os benefícios do JSF devido à agilidade na criação dos CRUD's. O *Spring* se destacou principalmente pelo seu mecanismo de segurança, tornando muito simples a restrição de módulos aos usuários, dando permissões adequadas a cada um. Em resumo, o JSF e *Spring* têm muitas coisas boas e vale a pena ser usado. Mostraremos inicialmente as contribuições do *JSF* para o desenvolvimento do estudo de caso. Na Figura 14 uma tela de listagem de TCC's.

The screenshot displays the 'eDoc UESB' system interface. At the top left is the UESB logo. The main header features the 'eDoc UESB' branding. Below the header is a navigation bar with 'Cadastros', 'Buscar TCC's', and 'Sobre'. The central area shows a table titled 'Listagem de TCCs' with the following data:

Título	Autor	E-mail	Orientador		
Análise dos Frameworks JSF e Spring	Fabricio Costa Silva	fabricioesb@yahoo.com.br	Denilton Costa Silva	✓	✗
Medicina alternativa	Cinara Costa Silva	cinara@bol.com.br	Fabricio Costa Silva	✓	✗

At the bottom of the interface, there are logos for 'spring source', 'MySQL', 'JSF', and 'RichFaces'. The text 'Sistema de controle de TCC' is visible on the left side.

**Figura 14: Tela do sistema com listagem de TCC**

Nesta são mostrados os principais dados do TCC em um componente UI<sup>1</sup> personalizado. Este componente é responsável por listar em forma de tabela os dados consultados no banco de dados. Este componente é do tipo *create, read, update and delete* (CRUD) que possui as opções de criar, apagar e atualizar um registro listado. Vale notar que estamos logado com um usuário com permissões administrativas, uma vez que os ícones de edição e exclusão estão sendo mostrados. Se clicarmos no ícone de edição, representado por um lápis, vamos para a tela de edição que pode ser vista na Figura 15:

<sup>1</sup> UI – User Interface traduzindo para o português seria interface com o usuário. Trata-se de componentes que o usuário pode enxergar e manipular manualmente.



**Cadastro de TCC's**

Matricula:	14200-5	✓
Autor:	Fabricio Costa Silva	✓
Título:	Análise dos Framework	✓
CPF:	022.714.405-80	✓
E-mail:	fabriciouesb@yahoo.cc	✓
Palavra-chave 1:	JSF	✓
Palavra-chave 2:	Spring	✓
Palavra-chave 3:	Framework	✓
Colegiado:	Colegiado de Ciência da Computação	✓
Orientador:	Denilton Costa Silva	✓
Membro 1:	Cinara Costa Silva	✓
Membro 2:	José da Silva Sauro	✓

Salvar Cancelar

Sistema de controle de TCC

Tecnologias

**Figura 15: Tela do sistema com edição de TCC**

Foi identificado nesse trabalho que a componentização do *JSF* é uma característica muito importante no ganho de tempo de desenvolvimento de interface com o usuário.

Observamos também que os componentes de validação do *JSF* são de grande valia no ganho de tempo de desenvolvimento. Se toda essa validação fosse feita em *JavaScript* consumiria um tempo de desenvolvimento razoável. Quando o *JSF* detecta que um campo foi preenchido de maneira incorreta ou mesmo não foi preenchido, sendo ele um campo obrigatório, é abortada a requisição. E informado o erro ao usuário, como podemos ver na Figura 13, tudo isso se forma transparente ao programador, basta usar os componentes do *JSF*.

Outra característica que também ajudou bastante foram os conversores do *JSF*. Como foi mostrado na sessão 2.5.8, os conversores fazem o papel de ajustar uma entrada do usuário para o tipo correspondente no seu *backingbean*, e o inverso também é verdadeiro. Podemos exemplificar na Figura 15 onde temos um *comboBox* listando o colegiado de um determinado TCC. Colegiado é na verdade um objeto *Java* que compõe o objeto *tcc*. Como no *comboBox* é mostrado o nome do colegiado, e o nome não é o objeto completo, quando é feita a requisição o conversor faz essa conversão e transforma em um objeto colegiado e nas fases posteriores é

atribuído o colegiado ao TCC correspondente. O mesmo ocorre quando chamamos a tela de edição, os dados precisam ser convertidos para que haja sentido para usuário. Mas para os programadores, é necessário analisar com outros olhos, pois aquilo é um objeto Java que precisa ser convertido para que faça sentido para o usuário. Agora imaginemos como seria isso em JSP. Teríamos que capturar o id do colegiado, instanciar manualmente um objeto colegiado, conectar com o banco de dados que é relacional, buscar o registro, inserir cada propriedade do colegiado manualmente e depois atribuir a um TCCb. O *JSF* elimina todos esses passos descritos, o que ajudou bastante no ganho de tempo.

Já deu para perceber os benefícios da componentização do *JSF*. Agora vamos falar um pouco a respeito dos benefícios do *Spring*. Por trabalhar nos bastidores o *Spring* é pouco notado, mas temos características dele que ajuda muito na construção de aplicações. Dois fatores que devemos destacar no *Spring* são a injeção de dependência e o *Spring Security*.

Na Figura 16 temos um trecho de código do e-Doc. Podemos notar que os objetos *tccService*, *departamentoService*, etc, não estão sendo instanciados, temos apenas referências na memória, mas por enquanto não podemos fazer nada pois não temos objetos instanciados na memória. Certamente quando for executada a penúltima linha do código da Figura 16 teremos um erro, pois o objeto não existe. Mas o *Spring* faz automaticamente a resolução dessa dependência, o que desacopla o código.

```
public class TccMB{

    private TccServiceImpl tccService;
    private DepartamentoServiceImpl departamentoService;
    private ColegiadoServiceImpl colegiadoService;
    private ProfessorServiceImpl professorService;
    private Tcc tcc;

    public TccMB() throws InstantiationException, IllegalAccessException
    {
        super();
    }

    ...

    public Collection<SelectItem> getDepartamentos(String l,String w,
String o) {

        return departamentoService.getDepartamentosItems(l, w, o);

    }

}
```

**Figura 16: Classe do Managed Bean de TCC**

O e-Doc tem como requisito ter o controle de acesso a cada módulo da aplicação. Cada usuário só deve acessar as partes que lhe diz respeito, e conseguimos fazer esse controle com a

ajudar do *Spring Security*. Adicionamos no arquivo de configuração dele as regras de acesso de maneira muito simples e objetiva. Quando uma requisição é feita ela é interceptada pelo *Spring Security* e só será liberado se o perfil do usuário for de acordo com a regra. Na Figura 17 mostramos como é simples essa regra. Com ela só pode acessar qualquer URL que tenha “`/pages/tcc/tccE*`” compondo parte da requisição se for usuário com as regras “`ROLE_ADM,ROLE_COLEGIADO`”.

```
<intercept-url pattern="/pages/tcc/tccE*" access="ROLE_ADM,ROLE_COLEGIADO">
```

**Figura 17: Regra de segurança do *Spring Security***

Outra maneira de garantir segurança com o *Spring Security* é com uma tag chamada “`sec:ifAnyGranted`”. Com ela podemos omitir partes da página baseado no perfil de cada usuário.

```
<sec:ifAnyGranted roles="ROLE_ADM">
    Conteúdo restringido à regra ROLE_ADM
</sec:ifAnyGranted>
```

**Figura 18: Comando condicional para restringir partes de uma página JSF**


Por exemplo, na listagem de TCC's mostrado na Figura 14 estamos logado como administrador. Vejamos então que aparecem as opções de excluir e editar TCC's. Já na Figura 19, como representamos a comunidade acadêmica fazendo uma busca de TCC, não aparece estes ícones, apenas o ícone de *download* do TCC. Isso porque o usuário comum não tem as regras de perfil necessárias para editar e excluir TCC's.



# eDoc UESB

Buscar TCC's    Sobre



Listagem de TCCs				
Titulo	Autor	E-mail	Orientador	
Analise dos Frameworks JSF e Spring	Fabricio Costa Silva	fabriciouesb@yahoo.com.br	Denilton Costa Silva	
Medicina alternativa	Cinara Costa Silva	cinara@bol.com.br	Fabricio Costa Silva	

Sistema de controle de TCC

Tecnologias   JSF  RichFaces

Figura 19: Tela do sistema logado com usuário comum

## 5. Considerações Finais e Trabalhos Futuros

Tendo como base o levantamento teórico sobre o *JSF* e o *Spring* framework, conclui-se que podemos obter um ambiente de programação web que permita desenvolvimento rápido de código baseado nos componentes UI do *JSF*, podemos também montar uma arquitetura que permita desenvolver código testável, pouco acoplado e reutilizável. Quando unimos esses dois frameworks temos um ambiente de desenvolvimento rápido sobre uma arquitetura eficiente para que possamos desenvolver as complexas aplicações do mundo atual.

Pôde-se perceber que a complexidade relacionada à arquitetura e interface das aplicações web pode ser diminuída com o uso do *JSF* integrado com o *Spring*. O *JSF* mostrou-se um excelente framework para a camada de apresentação, propiciando o desenvolvimento rápido e customizável no quesito interface. Já o *Spring* se mostrou um ótimo framework para trabalhar na camada de negócios. Ele faz o seu trabalho sem maiores interferências nas classes da aplicação por ser um framework não invasivo, permitindo também um sofisticado esquema de segurança e controle de permissões.

Notamos muitas características boas dessa parceria *JSF/Spring*. Analisamos aqui as propriedades que mostraram-se mais importantes no desenvolvimento de aplicações web, são elas: eliminação de trabalho que não está ligado ao objeto fim da aplicação. Por exemplo, validar campos, converter valores, etc, são atividades necessária, mas não fazem parte do escopo da aplicação e não é plausível perder muito tempo com essas tarefas secundárias. O *JSF* eliminou isso. Outro fator é a segurança em uma aplicação web e que o *Spring* incumbiu-se de assegurar. De maneira simples e eficiente podemos assegurar nossas aplicações web com *Spring Security*.

Então temos ao final deste trabalho um ambiente produtivo, seguro, visualmente agradável e configurável para desenvolver nossas aplicações web e ainda um sistema que ajudar na disseminação dos trabalhos produzidos na UESB.

Ao final ficam algumas sugestões que podem ser estudadas e implementadas para complementar esse trabalho.

- Estudar a integração do Hibernate junto com o *JSF* e *Spring*;
- Fazer a persistência dos TCC's em formato BLOB no banco de dados;
- Desenvolver um módulo de agendamento de TCC on-line;
- Integrar com o banco de dados da UESB;
- Melhorar interface das pesquisas.

## 6. Referências

BASS, P. et al. *Software Architecture in Practice*. Addison-Wesley, Boston, 2005

CHAVEZ, C. et al., **Composing Architectural Aspects based on Style Semantics. In: Proceedings of the ACM Intl. Conference on Aspect-Oriented Software Development (AOSD)**, March 2009, Charlottesville, USA, pp. 111-122.

CONCEIÇÃO, Rodrigo Menezes Da. **Javasever faces (jsf): UM ESTUDO COMPARATIVO ENTRE BIBLIOTECAS DE COMPONENTES**. 2008. 78 p. 94 f. Monografia (Graduação em Sistema de informação) - UNIVERSIDADE TIRADENTES. Aracaju.

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

DUDNEY, Bill; LEHR, Jonathan; MATTINGLY, Leroy. *Mastering Javasever™ faces*. Indianapolis: Wiley Publishing, 2004.

FERRAZ, Ronaldo M.. Construindo sites com padrões web. , 2003. Disponível em: <<http://kb.reflectivesurface.com/br/artigos/sitesComPadroesWeb/introducao> >. Acesso em: 15 jan. 2010.

GEARY, David; HORSTMANN, Cay. *Core Javasever faces. 2.* ed. California: Prentice Hall, 2007. 723 p. 745 f.

GOETTEN Junior, WINCK Diogo, 2006. **AspectJ - Programação Orientada a Aspectos com Java**. São Paulo - SP: Novatec Editora, 2006.

LADDAD, Ramnivas. **Aspectj in action: PRACTICAL ASPECT-ORIENTED PROGRAMMING**. Virginia: Manning, 2003. 513 p. 498 f.

MANN, Kito D. *Java server faces in action. 2.* ed. California: Manning, 2005. 1038 p. 1074 f.

RAFAEL PONTES. Entity converters pra dar e vender. , 2008. Disponível em: <<http://http://www.rponte.com.br/tag/javasever-faces/page/9/>>. Acesso em: 12 jan. 2010.

SUN MICROSYSTEMS. Java servlet technology. , 2010. Disponível em: <<http://java.sun.com/products/servlet/>>. Acesso em: 15 jan. 2010.

SUN MICROSYSTEMS. Java servlet technology. , 2010. Disponível em:  
<<http://java.sun.com/products/servlet/>>. Acesso em: 15 jan. 2010.

WALLS, Craig e Ryan Breidenbach . “Spring in Action ”. Greenwich: Manning Publications Co, 2008.

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

## 7. Apêndices

### 7.1. Telas do Sistema E-Doc

The screenshot displays the 'Cadastro de TCC's' form within the E-Doc UESB system. The form is titled 'Cadastro de TCC's' and contains the following fields and values:

Field	Value	Status
Autor:	Fabricio Costa Silva	✓
Título:	Integração do Framewo	✓
CPF:	022.714.405-80	
E-mail:	fabriciouesb@yahoo.cc	
Palavra-chave 1:	JSF	✓
Palavra-chave 2:	Spring	✓
Palavra-chave 3:	ramework	✓
Colegiado:	Colegiado de Ciência da Computação	✓
Orientador:	Cinara Costa Almeida	✓
Membro 1:	Denilton Costa Silva	✓
Membro 2:	José da Silva Sauro	✓

At the bottom of the form, there are two buttons: 'Salvar' and 'Cancelar'.

The interface also features a navigation bar with 'Cadastros', 'Buscar TCC's', and 'Sobre'. The footer includes the text 'Sistema de controle de TCC' and logos for 'Tecnologias', 'spring SOURCE', 'MYSQL', 'JSF', and 'RichFaces'.

Figura 20: Tela de edição/cadastro de TCC



Cadastros    **Buscar TCC's**    Sobre

- Por curso
- Por palavra-chave
- Por curso e palavra-chave**
- Por título
- Por autor
- Por departamento

**Pesquisa por curso e palavra chave**

Escolha o colegiado do curso:  
Colegiado de Ciência da Computação ▾

Digite a palavra chave:  
JSF

Pesquisar

Sistema de controle de TCC

Tecnologias    spring source    MySQL    JSF    RichFaces

Figura 21: Tela de busca por curso e palavra-chave

Cadastros    **Buscar TCC's**    Sobre

**Listagem de TCCs**

Titulo	Autor	E-mail	Orientador	Colegiado			
Integração do Framework JSF e Spring	Fabricio Costa Silva	fabriciouesb@yahoo.com.br	Cinara Costa Almeida	Colegiado de Ciência da Computação	✓	✗	✗

Sistema de controle de TCC

Tecnologias    spring source    MySQL    JSF    RichFaces

Figura 22: Tela com resultado de pesquisa da figura 22

The screenshot displays the 'eDoc UESB' web application interface. At the top left is the UESB logo. The header features the text 'eDoc UESB' in a large, stylized font. Below the header is a navigation bar with links for 'Cadastros', 'Buscar TCC's', and 'Sobre'. A dropdown menu is open under 'Cadastros', listing 'TCC', 'Colegiado', 'Departamento', 'Professor', and 'Campus'. The 'Colegiado' option is selected. The main content area shows a table titled 'Listagem de Colegiados' with the following data:

Nome	Sigla	Departamento	+
Colegiado de Ciência da Computação	CCComp	DCE	
Colegiado de Engenharia mecanica	CEM	DCE	
Colegiado de Medicina	CM	DCN	

At the bottom of the page, there is a footer with the text 'Sistema de controle de TCC' on the left and a row of logos for 'Tecnologias', 'spring source', 'MySQL', 'JSF', and 'RichFaces' on the right.

Figura 23: Tela com listagem de colegiados

## 7.2. Diagrama de classe da aplicação

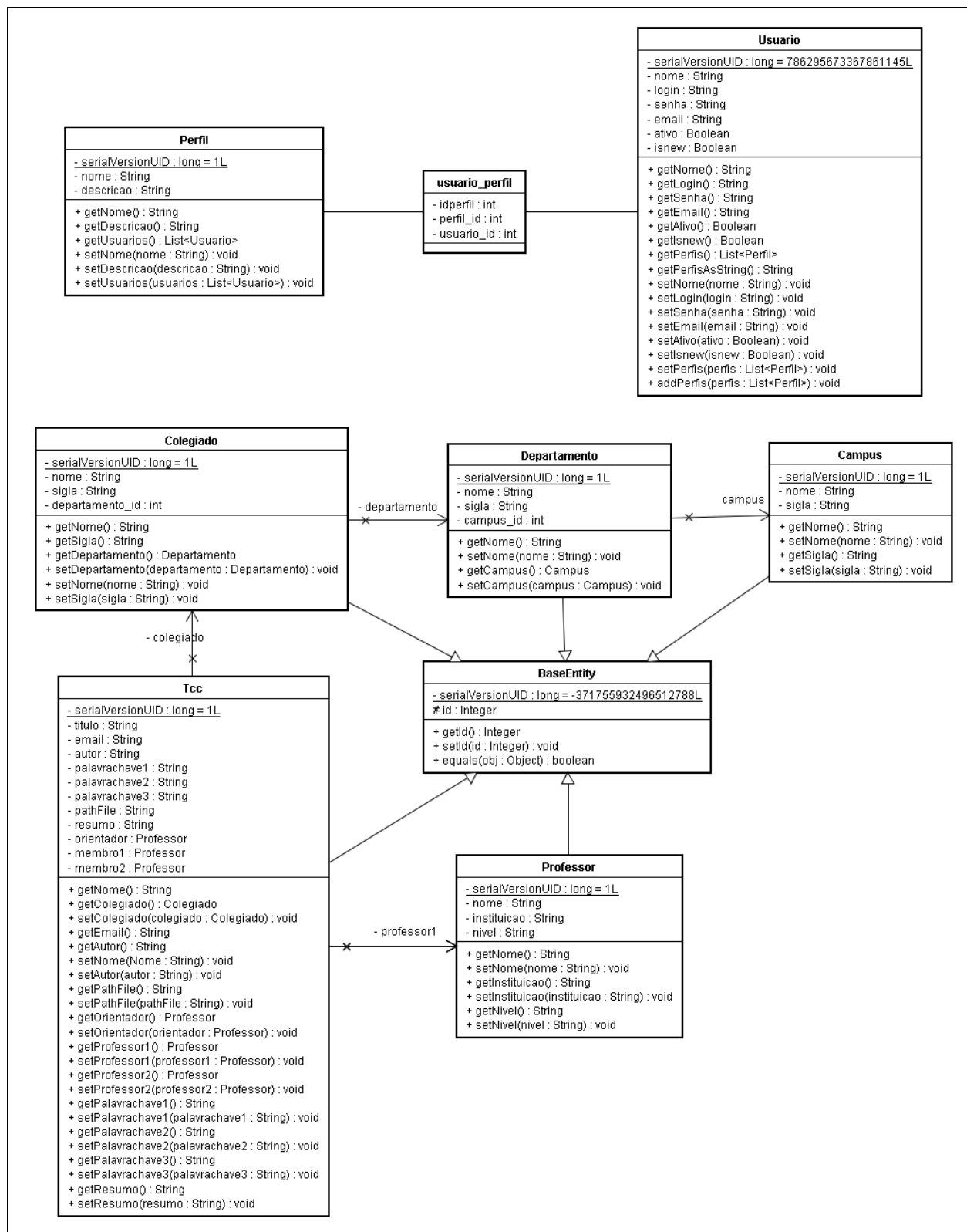


Figura 24 : Diagrama de classe