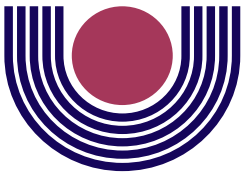

Unioeste - Universidade Estadual do Oeste do Paraná
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
Colegiado de Informática
Curso de Bacharelado em Informática

Estudo da Utilização de Frameworks no Desenvolvimento de Aplicações Web

Marcelo Teixeira

CASCAVEL
2008

MARCELO TEIXEIRA

Estudo da Utilização de Frameworks no Desenvolvimento de aplicações Web

Monografia apresentada como requisito parcial para obtenção do grau de Bacharel em Informática, do Centro de Ciências Exatas e Tecnológicas da Universidade Estadual do Oeste do Paraná - Campus de Cascavel

Orientador: Prof. MSc. Anibal Mantovani Diniz

CASCAVEL
2008

MARCELO TEIXEIRA

ESTUDO DA UTILIZAÇÃO DE FRAMEWORKS NO DESENVOLVIMENTO DE APLICAÇÕES WEB

Monografia apresentada como requisito parcial para obtenção do Título de Bacharel em Informática, pela Universidade Estadual do Oeste do Paraná, Campus de Cascavel, aprovada pela Comissão formada pelos professores:

Prof. MSc. Anibal Mantovani Diniz(Orientador)
Colegiado de Informática, UNIOESTE

Prof. MSc. Ivonei Freitas da Silva
Colegiado de Informática, UNIOESTE

Profª. Dra. Claudia Brandelero Rizzi
Colegiado de Informática, UNIOESTE

Cascavel, 11 de dezembro de 2008

AGRADECIMENTOS

Agradeço primeiramente a Deus por me dar forças para enfrentar as dificuldades, por iluminar a minha vida, e pela família e amigos que me destes.

À João Genésio *(in memorian)*, meu pai, pelos ensinamentos que me passou e por viabilizar recursos para que eu chegasse até aqui. Sua vida terrena, foi um exemplo de luta, trabalho e amor a família. Sua conduta nunca será esquecida.

À Delfina, minha mãe, por ter me cuidado desde de criança, pelo carinho e pela educação.

À Vanessa, minha esposa, que vem compartilhando comigo todas as alegrias e dificuldades, e por ter me dado muita força ao longo dessa caminhada.

À minhas irmãs, por partilharem vários momentos de alegria até hoje.

À Jandir e Graciema, meus Avôs, que compartilharam sua morada comigo nos primeiros anos de Faculdade.

A todos os meus colegas de graduação, principalmente Cassiano C. Casagrande, Rafael Voltolini, Adelar da Silva Queiróz, Claudir Galesky e Thiago da Silva Sodré, incluindo os antigos colegas André L. Brun, Alan, Cassiano Vargas, Carlos Moratelli, e outros não citados aqui. Por todos os momentos juntos nessa luta.

Ao Anibal Mantovani Diniz, meu orientador, por me incentivar e me auxiliar neste trabalho ao longo do ano. A professora Claudia e ao professor Ivonei por auxiliarem com correções e instruções para este trabalho. Aos demais professores do Colegiado de Informática por sua dedicação aos longo desses anos.

Lista de Figuras

Lista de Quadros

Lista de Tabelas

Lista de Abreviaturas e Siglas

API	*Application Programming Interface*
Db4o	*Database for Objects*
HTML	*HyperText Markup Language*
HTTP	*HyperText Transfer Protocol*
IDE	*Integrated Development Environment*
IoC	*Inversão de Controle*
JCP	*Java Community Process*
JDK	*Java Development Kit*
JPA	*Java Persistence API*
JSF	*JavaServer Faces*
JSP	*JavaServer Pages*
MVC	*Model - View - Controller*
UML	*Unified Modeling Language*
XML	*Extensible Markup Language*

Sumário

Resumo

Devido à complexidade envolvida no desenvolvimento de sistemas para web, torna-se cada vez mais importante a adoção de *frameworks* visando aumentar a simplicidade e a produtividade das aplicações. *Frameworks* para web são ferramentas que auxiliam nas principais atividades no desenvolvimento dessas aplicações, tais como processamento de requisições e separação da aplicação em camadas. Além disso, disponibilizam componentes reutilizáveis para serem utilizados em diferentes aplicações. Atualmente, existe uma grande quantidade de *frameworks open-source* disponíveis no mercado, e a equipe desenvolvedora durante a fase de elaboração dos sistemas deverá fazer a escolha daquele que mais atenda às necessidades de projeto. Portanto, é necessário que se investigue algumas daquelas que aumentam a produtividade para sistemas web. Este trabalho realizou um estudo apresentando pontos positivos e negativos de três dentre os considerados melhores *frameworks* java para web segundo a comunidade java: JavaServer Faces, Struts2 e SpringMVC. São apresentados os resultados de uma avaliação dos *frameworks* envolvendo critérios como: Qualidade da documentação, facilidade de configuração, demanda e disponibilidade de profissionais com conhecimento nos *frameworks*. Ao final, verificou-se quais frameworks seriam mais indicados para cada tipo de projeto, levando em conta sua complexidade e se a empresa desenvolvedora já utiliza outros *frameworks* para desenvolvimento web.

Palavras-chave: Struts2, JSF, SpringMVC, Avaliação de *frameworks* java para web.

Capítulo 1

Introdução

Aplicações web são programas armazenados em um servidor. Dependendo do tipo da aplicação, ele poderá executá-lo neste ambiente e retornar artefatos de software para serem interpretados pelo navegador do cliente. Entre estes artefatos, podem ser encontrados diversos formatos, tais como: flash, javascript, applets, XML e ainda HTML. O desenvolvimento de tais aplicações obriga-nos a programar com conceitos que não estamos acostumados quando desenvolvemos aplicações *desktop*. O principal deles é a navegabilidade das informações via protocolo HTTP.

No processo de desenvolvimento de aplicações web, encontramos um grande número de tarefas básicas repetitivas, tal como o controle de fluxo de páginas. Assim como acontece na programação convencional, essas tarefas repetitivas podem ser quebradas em métodos ou procedimentos. Semelhantemente, podemos generalizar estas tarefas de forma a reutilizá-las em outros projetos. Este processo de generalização e reuso, segundo Nash [22], é o motivo que leva a criação de *frameworks*.

Segundo Schmidt et al. [25], um *framework* é um conjunto integrado de artefatos de software (classes, objetos, e componentes) que colaboram para fornecer uma arquitetura reutilizável para uma família de aplicações relacionadas.

Muitas empresas investem especificamente em *frameworks*, tanto desenvolvendo o seu próprio, quanto adquirindo ou adotando um existente. De seu ponto de vista, é um investimento para o futuro; é um investimento que reduz o tempo de desenvolvimento e melhora a qualidade do produto final. Seus desenvolvedores conseguem dar mais atenção aos requisitos de negócio nos seus projetos, ao invés da infra-estrutura de projeto e desenvolvimento.

Os *frameworks* trazem um grande número de benefícios para o processo de desenvolvi-

mento. Segundo Nash [22], o principal benefício é a velocidade de desenvolvimento. Uma vez que a curva de aprendizado é superada, o tempo necessário para se desenvolver uma aplicação web pode se tornar bem menor comparado ao tempo necessário para se desenvolver a mesma aplicação sem o uso de um *framework*.

Segundo Markiewicz e Lucena [19], *frameworks* devem ser considerados como uma solução para desenvolvimento de software quando os requisitos sofrem mudanças rapidamente, e segundo Wong et al. [33], um *framework* pode ajudar a padronizar o processo de desenvolvimento, aumentando a produtividade e a manutenabilidade do código. Além disso, os *frameworks* possibilitam a separação entre as camadas de interface e a camada da lógica de negócio dentro de uma aplicação, permitindo que equipes possam se especializar em uma das camadas.

Nota-se hoje que existem vários *frameworks* para aplicações web no mercado, tais como JavaServer Faces, Tapestry, Spring, Grails, Wicket, Jboss Seam, Oracle ADF e Struts, utilizando a tecnologia java. Utilizando a linguagem PHP temos CakePHP, Symfony, Zend Framework, CodeIgniter e Prado. Em .NET temos o Microsoft UIP e Maverick.NET, e utilizando a linguagem Ruby temos o Rails [21][6].

Para este trabalho, escolheu-se *frameworks* java por ser esta tecnologia a mais utilizada pelas grandes empresas de desenvolvimento. Com apenas algumas pesquisas em fóruns e comunidades de desenvolvimento para web, é possível verificar que a tecnologia java é a mais utilizada pelos desenvolvedores, principalmente quando as aplicações são complexas, exigindo escalabilidade e desempenho. Outro fator considerado, é que aplicações java rodam em diversos dispositivos além de computadores, como celulares e PDA's.

Para este trabalho, foram escolhidos apenas três deles, já que o processo de aprendizado dos mesmos é bastante complexo [22]. Optou-se, então, por JavaServer Faces, Struts2 e SpringMVC por estarem entre os mais indicados pelas comunidades de desenvolvimento web.

1.1 Objetivos

1.1.1 Objetivos Gerais

Este trabalho de conclusão de curso tem por objetivo geral analisar e comparar os *frameworks* JSF (JavaServer Faces), Struts2 e SpringMVC, apresentando pontos positivos e negativos dos mesmos no desenvolvimento de aplicações web.

1.1.2 Objetivos Específicos

Os seguintes tópicos fazem parte dos objetivos específicos deste trabalho:

a) Levantamento de *frameworks* disponíveis para desenvolvimento de software em Java para a WEB;

b) Fazer um estudo sobre como cada *framework* implementa o padrão MVC (Modelo - Visão - Controlador);

c) Realizar uma discussão sobre os critérios que devem ser analisados pela equipe de desenvolvimento para se adotar um *framework*, levando em conta o domínio da aplicação e a complexidade do projeto;

d) Descobrir quais aplicações seriam beneficiadas por *frameworks* levando em conta seu porte e complexidade;

e) Verificar as possibilidades e as vantagens de se trabalhar com mais de um *framework* em conjunto, e quais são as dificuldades de integração.

1.2 Motivação

Os desenvolvedores encontram muitas dificuldades na hora de escolher um *framework* para o domínio da aplicação. Isto ocorre devido à baixa curva de aprendizado dos *frameworks* e por haver dúvida de qual *framework* irá melhor atender os requisitos de determinada aplicação [22]. Assim, mesmo existindo vários *frameworks* para desenvolvimento de aplicações web no mercado, algumas empresas ainda continuam construindo seus projetos sem sua utilização, pelo fato de ainda existirem muitas dúvidas sobre suas vantagens e desvantagens. Um estudo sobre estes *frameworks* focado em auxiliar na decisão da escolha poderá contribuir para a produtividade no desenvolvimento web.

1.3 Estrutura do Trabalho

Este trabalho está estruturado da seguinte forma:

No capítulo 1 são apresentados a introdução, motivação, objetivos e a estrutura deste trabalho de conclusão de curso.

No capítulo 2 são apresentados conceitos importantes para que se possa prosseguir com o estudo sobre *frameworks*. É apresentado como funciona a navegabilidade das informações em aplicações web, assim como tecnologias java para construção de páginas e padrões de projeto.

No capítulo 3 é feito um estudo sobre cada um dos *frameworks* escolhidos: JSF, Struts2 e SpringMVC, apresentando suas arquiteturas e características que são importantes para a construção de aplicações web.

O capítulo 4 descreve um estudo de caso que será utilizado para se avaliar os *frameworks*. São apresentados os requisitos desse sistema e também os casos de uso. O objetivos desse estudo de caso é poder extrair características desses *frameworks* que somente a prática pode fornecer.

No capítulo 5 são apresentadas as avaliações dos *frameworks*. São analisados critérios como disponibilidade de documentação, disponibilidade de profissionais, demanda de profissionais, disponibilidade de suporte por parte da comunidade ou de instituição oficial e facilidade de configuração.

Já no capítulo 6 são feitas as conclusões do trabalho, são descritas as dificuldades encontradas e sugestões para trabalhos futuros.

Capítulo 2

Tecnologias Java Disponíveis para o Desenvolvimento Web

As primeiras aplicações web apresentavam conteúdo estático, ou seja, apenas textos, imagens e outros elementos que já estavam no servidor e que eram apenas enviados ao *browser* do usuário. Com o passar do tempo verificou-se que este modelo podia ser melhorado. Poderia haver mais interação entre o usuário e o servidor. Com isto foram surgindo tecnologias que rodavam no servidor para tornar essa interatividade maior através de geração de páginas dinâmicas e de acesso ao banco de dados. Mas tais tecnologias tornaram muito mais complexo o desenvolvimento desses sistemas.

Neste contexto, surgiram então, tecnologias java para se trabalhar com a web. Servlet e JSP são até hoje as tecnologias java que formam a base para essas aplicações. São estes elementos que processam as requisições, invocam processamentos complexos e geram as páginas dinamicamente.

Outros elementos importantes também, são os padrões de projeto. Eles surgiram através da documentação dos problemas comuns que os desenvolvedores encontravam no dia-a-dia junto com suas respectivas soluções [22]. Um destes padrões, o MVC, fornece também uma arquitetura para a aplicação. Ela é dividida em camadas facilitando a manutenção e aumentando o reuso. Os *frameworks* para web existentes hoje no mercado, unem o MVC com outros padrões de projeto fornecendo vários componentes da arquitetura da aplicação. Esses conceitos e tecnologias são melhor esclarecidos da Seção 2.1 até a Seção 2.5 deste capítulo.

Um dos grandes desafios ainda existentes para os *frameworks* é tornar as aplicações mais reutilizáveis e mais fáceis de configurar. Algumas técnicas que estão sendo utilizadas para se

alcançar esses objetivos são o uso de Inversão de Controle, anotações e XML, que são apresentados nas Seções 2.6, 2.7 e 2.8 deste capítulo. Na Seção 2.9 é apresentada a validação, um processamento que deve existir em aplicações web para garantia de consistência dos dados no servidor.

2.1 O Modelo de Funcionamento da Navegabilidade em Aplicações Web

A navegabilidade das informações em aplicações web é bastante complexa. Enquanto que em aplicações *desktop* o acesso aos dados ou aos serviços pode ocorrer de forma direta, em aplicações web toda troca de informações entre o navegador e o servidor precisa ser feita sobre o protocolo HTTP.

Este modelo é composto por dois elementos principais: o lado cliente, onde está o usuário utilizando um navegador e o lado do servidor, que contém as informações e o conteúdo que interessa para o usuário. Neste modelo, o usuário interage clicando em links ou em botões que fazem com que uma requisição seja gerada e enviada ao servidor. Este analisa esta requisição e devolve como resposta uma outra página para o usuário.

As primeiras aplicações web construídas disponibilizavam apenas páginas estáticas. Neste modelo o servidor web responde a todas as requisições com páginas já existentes com conteúdo pré-definido, o que torna essas aplicações muito limitadas. Quando se tem páginas dinâmicas, a requisição depois de recebida, é processada pelo servidor web que vai criar dinamicamente o conteúdo que depois será enviado para o cliente. As páginas dinâmicas devem ser programadas utilizando alguma linguagem de programação, que dependendo do servidor web pode ser PHP, Java, Ruby ou .NET por exemplo. Assim, podemos criar componentes web que rodam no servidor, eventualmente acessando a bases de dados e produzindo conteúdo dinamicamente. A Figura 2.1 apresenta esses dois modelos, em (a) utilizando páginas estáticas e em (b) utilizando componentes web para geração de páginas dinâmicas.

Para ficar mais claro, podemos citar um exemplo no qual um usuário deseja consultar quais produtos estão disponíveis em uma loja. Quando o servidor web recebe a requisição, ele faz acesso ao banco de dados e gera a página de resposta de acordo com os produtos que estão disponíveis neste banco.

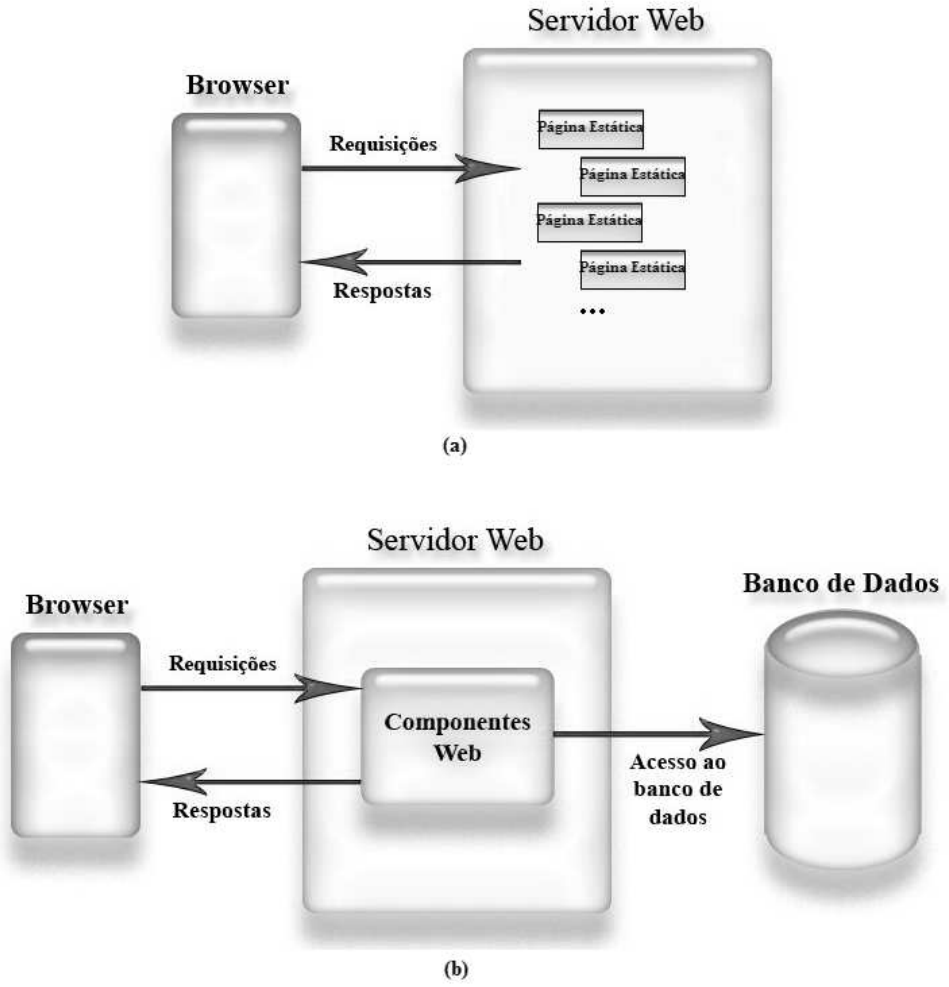

Figura 2.1: Modelo de aplicações web com páginas estáticas (a) e dinâmicas (b).

2.2 Servlet e JSP

Servlet e JSP (Java Server Pages) são duas tecnologias Java para a construção de aplicações web para geração de conteúdo dinâmico, similares às tecnologias PHP e ASP. Servlets são classes Java, desenvolvidas de acordo com uma estrutura bem definida, e que, quando instaladas junto a um Servidor que implemente um Servlet Container (um servidor que permite a execução de Servlets, como o Tomcat por exemplo), podem tratar requisições recebidas de clientes [31].

Podemos dizer que um JSP consiste de uma página HTML com alguns elementos especiais, que conferem o caráter dinâmico da página. Esses elementos podem tanto realizar um processamento por si, como podem recuperar o resultado do processamento realizado em um Servlet

por exemplo, e apresentar esse conteúdo dinâmico junto a página JSP. Enquanto em um Servlet o processamento de requisições se mistura com a geração de conteúdo e formatação HTML, em um JSP essas tarefas se encontram separadas, facilitando modificações na aplicação.

Para se beneficiar das boas características de cada uma delas, podemos trabalhar com as duas em conjunto. Podemos utilizar JSP para criação e manutenção de HTML, e utilizar Servlets para invocar métodos da lógica de negócios e executar outras tarefas mais complexas [13]. Assim, podemos dizer que JSP é mais indicado para tarefas orientadas para apresentação e que Servlets é mais indicado para tarefas orientadas ao processamento.

Mesmo sendo elementos diferentes, todo JSP é convertido em um Servlet e compilado. Quando o navegador requisita a página JSP ao servidor web pela primeira vez, o *Servlet Engine* primeiramente converte o JSP em um servlet e o executa retornado o resultado para o cliente. Nas próximas requisições a página JSP, o servlet que foi criado na primeira requisição será executado diretamente, sem necessidade de conversão [7].

2.3 Padrões de Projeto

Padrões de projeto podem ser vistos como soluções de eficiência já comprovadas e amplamente utilizadas para a resolução de problemas comuns em projeto de software. Estas soluções são desenvolvidas e conhecidas por especialistas e tornam-se padrões por serem reutilizadas várias vezes em vários projetos e por terem eficácia comprovada [5].

Um padrão descreve um problema que ocorre várias vezes em determinado contexto, e descreve ainda a solução para esse problema, de modo que essa solução possa ser utilizada sistematicamente em situações diferentes [11].

Padrões de projeto são utilizados no projeto de software orientado a objetos e geralmente são descritos utilizando-se UML. A seguir são apresentados alguns dos principais padrões de projeto utilizados tanto na API Java quanto nos *frameworks*.

2.3.1 Composite

As interfaces gráficas geralmente são compostas por dois tipos de objetos gráficos: os primitivos e os compostos. Um objeto composto é aquele que contém ou é formado por outros objetos. Por exemplo, um formulário é considerado um objeto composto quando contém in-

ternamente um label, um campo de texto e um botão. Quando se trabalha com uma complexa hierarquia de componentes, se torna difícil para o programador executar operações nesses objetos sem saber seu tipo. O composite é um padrão que permite que clientes lidem com objetos primitivos e compostos uniformemente. Todos os objetos compostos ou primitivos devem implementar uma classe abstrata, a qual possui as operações para se trabalhar de forma uniforme [15][11].

2.3.2 Observer

Existe um problema quando se tem vários objetos interessados (chamados *Observers*) em receber notificações de eventos ou mudanças de estado de um outro objeto (chamado *Subject*), principalmente quando:

a) O número e tipos dos *Observers* não é conhecido em tempo de compilação;

b) Os vários *Observers* poderão não fazer parte de uma mesma hierarquia de objetos;

c) Não queremos criar um acoplamento forte entre *Subject* e *Observers*.

O objetivo deste padrão é automaticamente notificar os objetos interessados pelos eventos que ocorrem no *Subject*. Os *Observers* quando querem receber eventos de um *Subject* se cadastram a ele, de forma que o *Subject* mantém uma lista de *Observers* interessados. Assim, quando o *Subject* gera um evento, ele apenas notifica os interessados desta lista [11].

2.3.3 Facade

O problema que o Facade tenta resolver é quando se tem um subsistema com muitas funcionalidades e estas são muito complexas para um objeto interessado (cliente) utilizar. O propósito deste padrão é oferecer uma interface única para um conjunto de interfaces de um subsistema, tornando o subsistema mais fácil de usar [11].

2.3.4 Singleton

Existem momentos em que se torna necessário que exista apenas uma instância de uma certa classe. Por exemplo, se existe uma classe que gerencia a conexão com um banco de dados, então deve existir apenas uma instância dessa classe para que se tenha apenas uma conexão

com o banco na mesma aplicação. O padrão Singleton tem por objetivo garantir essa restrição [15][11].

2.3.5 Command

Os objetivos do padrão Command são encapsular uma requisição como um objeto, permitindo que clientes parametrizem diferentes requisições, filas ou requisições de log, e também suportar operações reversíveis [11]. O padrão consiste em usar polimorfismo para construir objetos que encapsulam um comando oferecendo um único método *execute*() com a implementação do comando a ser executado. Desta forma, consegue-se encapsular as operações e reduzir o acoplamento entre as requisições e os objetos que as executam, facilitando a implantação de novas operações e tornando mais simples sua manutenção. Este padrão é muito utilizado nos *frameworks*, e sua utilização é bem clara no Struts2 que é apresentado no próximo capítulo.

2.4 Arquitetura MVC

MVC (Modelo - Visão - Controlador) é uma arquitetura e também um padrão de projeto utilizado nas aplicações para separação dos dados ou da lógica de negócio (Modelo) da interface com o usuário (Visão) e do fluxo de aplicação (Controlador). Esta separação em camadas é de extrema importância, principalmente para as aplicações web onde as tecnologias de apresentação de conteúdo mudam com muita freqüência [22].

Na camada Visão, um componente de visualização renderiza o conteúdo de uma parte particular do modelo e encaminha para o controlador as ações do usuário. Acessa também os dados do modelo, via controlador, e define como esses dados devem ser apresentados.

O Modelo representa os dados e as regras de negócio que governam o acesso e a modificação dos dados. Mantém o estado persistente do negócio e fornece ao controlador a capacidade de acessar as funcionalidades da aplicação encapsuladas pelo próprio modelo.

O Controlador define o comportamento da aplicação. É ele que interpreta as ações do usuário a as mapeia para chamadas ao modelo. Em um cliente de aplicações web, essas ações do usuário poderiam ser cliques de botões ou seleções de menus. As ações realizadas pelo modelo incluem ativar processos de negócio ou alterar o seu próprio estado.

2.4.1 Modelo MVC 1

O modelo 1 foi uma das primeiras arquiteturas do MVC. Neste modelo, o tratamento das requisições e das respostas no servidor é realizado pelo mesmo componente, que pode ser tanto um JSP quanto um Servlet. Assim, apenas a lógica de negócios se encontra em componentes separados [22].

2.4.2 Modelo MVC 2

O modelo MVC 2 surgiu na tentativa de solucionar os problemas encontrados no modelo 1 no qual o mesmo componente realizava várias tarefas. No novo modelo, a requisição é primeiro interceptada por um controlador. Esse controlador chama a lógica de negócio necessária na camada Modelo e então determina qual página deve ser enviada para o usuário. Nesta arquitetura, o Controlador, o Modelo e a Visão se encontram em componentes distintos. Desta forma, com um baixo acoplamento, a manutenção e reutilização das aplicações são favorecidos [22][8]. Na Figura 2.2 pode ser visto a arquitetura do Modelo 1 em (a), e do Modelo 2 em (b).

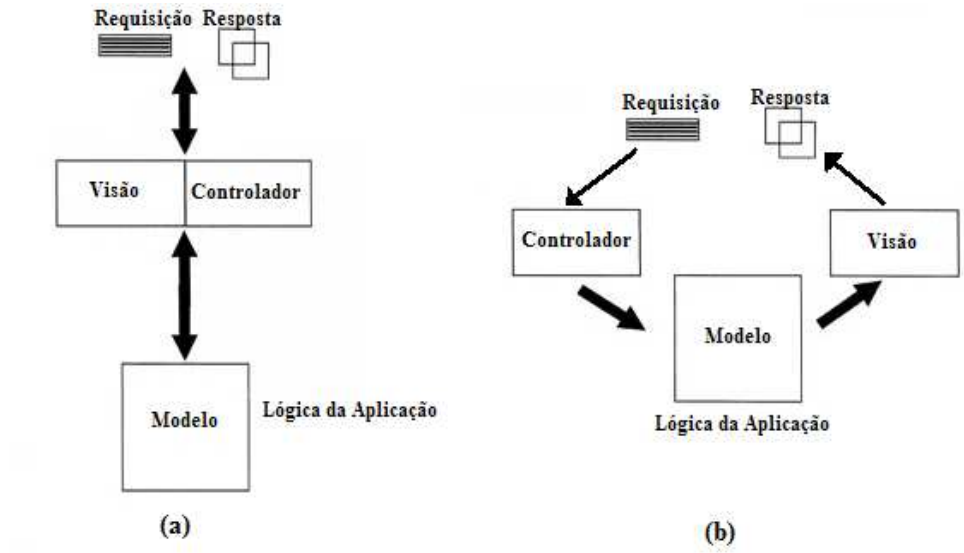

Figura 2.2: Arquitetura MVC Modelo 1 (a) e Modelo 2 (b). (Adaptado de [22])

2.5 Frameworks

Os *frameworks* surgiram da idéia de unir as características de padrões de projeto, do MVC, e componentes reutilizáveis para as tarefas mais comuns dos programadores junto a uma estrutura para a aplicação, de forma a tornar mais rápido e mais fácil o desenvolvimento de aplicações web [22]. Um bom *framework* deveria fornecer um ambiente genérico onde diferentes tipos de aplicações poderiam utilizá-lo.

Um *framework* deve possuir as seguintes características [4]:

a) Englobar várias classes ou componentes, cada um podendo fornecer uma abstração de algum conceito em particular;

b) O *framework* define como essas abstrações trabalham juntas para resolver um problema;

c) Os componentes devem ser reutilizáveis;

d) O *framework* deve organizar padrões de projeto em alto-nível.

É importante ressaltar que *framework* e biblioteca são diferentes. Enquanto este último possui rotinas ou funções que uma aplicação pode chamar, um *framework* fornece componentes genéricos e relacionados que uma aplicação pode estender para prover um conjunto particular de funções [4].

Como o foco deste trabalho de conclusão de curso é o estudo de *frameworks*, foi necessário escolher alguns dentre os vários *frameworks* java para web existentes. Optou-se então por JavaServer Faces, SpringMVC e Struts2 devido a grande popularidade e aceitação dos mesmos na comunidade java.

2.6 Inversão de Controle (IoC)

A inversão de controle é o nome dado a um padrão de desenvolvimento que é disponibilizado em alguns *frameworks* que consiste em mudar o fluxo de controle de um programa, onde ao invés de o programador determinar quando um método será executado, ele apenas determina qual método deve ser executado quando um determinado evento ocorrer [14].

Quadro 2.6.1: Código sem utilização de Inversão de Controle

```
1.    Context ctx = new InitialContext();
2.    DataSource ds = (DataSource) ctx.lookup("jdbc/WroxJDBCDS");
3.    Connection conn = ds.getConnection("user", "pass");
4.    Statement stmt = conn.createStatement();
5.    ResultSet rs = stmt.executeQuery( "SELECT * FROM Cust" );
6.    while( rs.next() )
7.        System.out.println( rs.getString(1)) ;
8.    rs.close() ;
9.    stmt.close() ;
10.   conn.close() ;
11.   con.close();
```

O trecho de código do Quadro 2.6.1 é um exemplo onde não é utilizado Inversão de Controle. Neste código, fica a cargo do programador obter um DataSource através de um lookup JNDI.

Os serviços de Inversão de Controle podem ser disponibilizados por *frameworks* como o Spring por exemplo. O objetivo da Inversão de Controle neste exemplo é de deixar a cargo do *framework* criar ou localizar um DataSource para a aplicação. Quando se utiliza IoC para o atributo **ds**, não se faz mais necessário a linha 2 do código do Quadro 2.6.1. Para isto, o desenvolvedor deve informar através de arquivos de configuração, alguns parâmetros para que o *framework* possa criar uma instância adequada para a aplicação [32]. O método SetDs localizado na linha 2 do do Quadro 2.6.2 deve existir para que o *framework* possa "injetar" a instância para dentro da aplicação. Este método é chamado automaticamente assim que algum trecho da aplicação for utilizar o atributo **ds**. O Quadro 2.6.2 mostra a utilização de inversão de controle onde se obtem as mesmas funcionalidades do Código 2.6.1.

Quadro 2.6.2: Código mostrando a utilização de Inversão de Controle

```
1.    private DataSource ds;
2.    public void setDs(DataSource datasource) {
3.        ds = datasource;
4.    }
5.    ...
6.    Connection conn = ds.getConnection("user", "pass");
7.    Statement stmt = conn.createStatement();
8.    ResultSet rs = stmt.executeQuery( "SELECT * FROM Cust" );
9.    while( rs.next() )
10.       System.out.println( rs.getString(1)) ;
11.   rs.close() ;
12.   stmt.close() ;
13.   conn.close() ;
```

Com a disponibilização deste padrão pelos *frameworks*, consegue-se diminuir o acoplamento entre as camadas [14][10], e com isso:

a) Aumentar a reutilização de classes e componentes;

Quadro 2.7.1: Código com exemplo de uso de Anotações

```
1.    @Entity
2.    @Table(name = "User" )
3.    public class UserBean implements java.io.Serializable {
4.        private int id;
5.        ...
6.        @Id (generate = GeneratorType.AUTO)
7.        public int getId()
8.        {
9.            return this.id;
10.       }
11.       ...
12.   }
```

b) Construir classes mais fáceis de testar;

c) Construir aplicações mais fáceis de configurar.

2.7 Anotações

Anotações são meta-informações que "anotam"ou marcam elementos tais como atributos ou métodos com informações, para serem processadas na compilação, implantação ou execução dos programas [16]. Entre os vários usos de anotações, podemos citar [28]:

a) Passar informações ao compilador para detecção de erros ou para omissão de warnings;

b) Ferramentas podem utilizar as anotações para gerar código e arquivos XML em tempo de compilação ou de implantação;

c) Ferramentas podem utilizá-las para checagem em tempo de execução.

Um exemplo de uso de anotações pode ser visto no código do Quadro 2.7.1. A anotação @Entity determina que a classe UserBean deve ser gerenciada como uma entidade de persistência, ou seja, será mapeada para uma tabela no banco de dados relacional. A anotação @Table(name = “User”) determina a tabela para qual a classe será mapeada, que no caso é “User”. E por fim, a anotação @Id (generate = GeneratorType.AUTO) determina que o atributo “id” será chave primária e que seu valor será gerenciado pelo provedor de persistência, ou seja, pelo componente responsável pelo mapeamento objeto-relacional da aplicação, que pode ser tanto o Hibernate quanto o JPA.

2.8 XML para Configuração de Frameworks

XML significa Extensible Markup Language. É uma linguagem de marcação semelhante ao HTML capaz de descrever diversos tipos de dados. No XML ao contrário do HTML, é possível definir as tags que forem necessárias para representar a informação. O propósito principal dessa linguagem é permitir a troca de informações de forma estruturada atráves da Internet, permitindo que os programadores transportem dados de um servidor para outro da rede de forma transparente e organizada [20].

A utilização de XML nas aplicações construídas com *frameworks* é muito comum. O objetivo é tentar retirar algumas configurações ou regras de dentro do código, separando-os da lógica de negócio. Quando se trata de aplicações web, utiliza-se muito o XML para definir as regras de navegação, as quais também não necessitam mais estar dentro da lógica do programa [18]. Isto torna as aplicações mais reutilizáveis e mais fáceis de alterar.

A utilização de XML também facilita a aplicação de testes, pois as dependências dos objetos podem ser encontrados mais facilmente quando as informações estão centralizadas em arquivos XML [18].

2.9 Validação

Em uma aplicação web, existe um componente da camada de controle que recebe as requisições. Cabe a ele extrair os parâmetros desta requisição, que são do tipo String, e convertê-los para tipos mais específicos da aplicação, que podem ser tipos primitivos mais restritos como inteiros e boleanos ou objetos do Modelo, e validá-los. Essa validação consiste em verificar se os parâmetros atendem os requisitos dos tipos de dados do Modelo. A validação verifica por exemplo, se o usuário digitou apenas números em um campo onde deve ser informado a idade do usuário. Em caso de erro de validação, o controlador exibe novamente a visão que causou o erro. Os *frameworks* disponibilizam componentes que são utilizados na visão, para repovoar os campos preenchidos e exibir mensagens sobre os erros ocorridos [7].

A tarefa de validação dos dados digitados pelos usuários, é de extrema importância para tornar as aplicações confiáveis. Dessa maneira, ela deve ser executada de acordo com padrões bem especificados. Se o usuário digitar valores inválidos e a aplicação aceitá-los, um comportamento anômalo pode ocorrer. A camada de Controle, em conjunto com a camada Modelo, é

responsável por essa tarefa, fazendo-se necessário que ela realize um processamento específico.

Os processamentos mais comuns para a verificação dos dados incluem: validação de endereço de e-mail, validação de CEP, verificação de campos não preenchidos, etc. A fim de facilitar o desenvolvimento, os *frameworks*, normalmente, fornecem processamentos para tais tarefas.

Capítulo 3

Frameworks JSF, Struts2 e SpringMVC

Para que se possa fazer uma avaliação dos *frameworks* JSF, Struts2 e SpringMVC, escolhidos para este estudo, se faz necessário um estudo de cada um deles. Com isto, pretende-se apresentar, ao final deste trabalho, pontos positivos e negativos dos mesmos para o desenvolvimento de aplicações web.

Nas próximas seções será apresentada a forma com que os três *frameworks* implementam a arquitetura MVC, descrevendo os principais componentes responsáveis pelo tratamento das requisições, pela construção de interfaces com o usuário e pela execução da lógica de negócios da aplicação.

3.1 Framework Struts2

O Struts2 é um *Framework* desenvolvido pela Apache Software Foundation que inicialmente era conhecido como WebWork [9]. Após trabalhar independentemente por vários anos, as comunidades do WebWork e do Struts se uniram para a criação do Struts2, que é mais simples de usar e mais próximo de seu propósito inicial [9]. Dentre os objetivos da nova versão do Struts, podemos citar a diminuição do acoplamento entre os módulos do *framework*, aumentando assim a reutilização dos componentes e conseguindo-se também uma melhor manutenabilidade do sistema. A seguir é apresentada a arquitetura MVC implementada no Struts2.

3.1.1 Arquitetura MVC no Struts2

Os componentes principais da arquitetura MVC do Struts2 são o FilterDispatcher que representa o Controlador, as actions que representam o modelo, e os results que representam a visão. O fluxo de trabalho e as relações entre os módulos da arquitetura podem ser vistos na Figura

3.1.

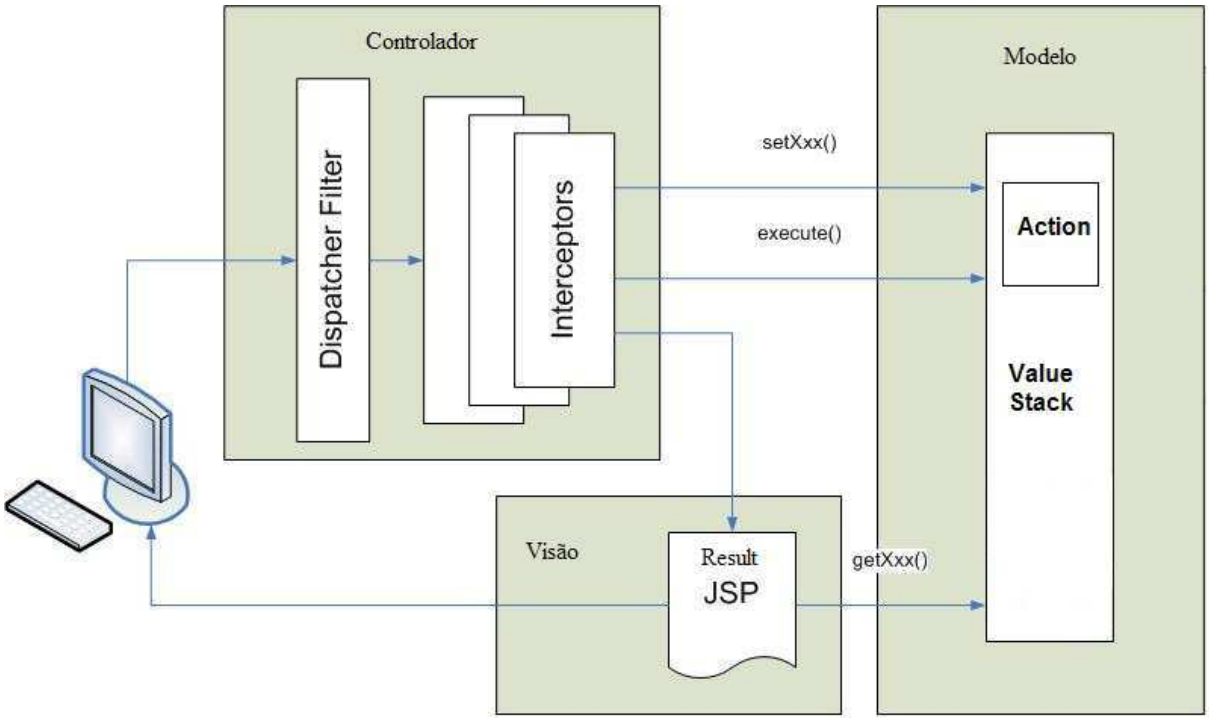

Figura 3.1: Arquitetura MVC do Struts2. (Adaptado de [24])

A seguir é apresentada uma descrição dos principais componentes dessa arquitetura:

Controlador - FilterDispatcher

A principal tarefa deste componente é de receber as requisições HTTP vindas do usuário e determinar, para cada requisição, qual action deverá processá-la. Esse mapeamento no Struts2 pode ser feito através de um arquivo XML chamado “web.xml“ ou através de anotações.

Modelo - Action

Cada action é representada por uma classe java tendo dois papéis importantes:

1. Todas as actions devem possuir um método execute(), o qual pode conter várias chamadas à lógica de negócios. Uma vez identificada a action que processará a requisição do usuário pelo controlador, este método é invocado automaticamente pelo *framework*.

2. Servir como um local para transferência de dados. Uma vez realizado o tratamento da requisição pela action, uma certa página JSP deve ser selecionada e enviada ao usuário. Se

esta página necessitar de dados para serem exibidos ao usuário, estes devem ser fornecidos pela action.

Visão - Result

Results são as páginas que serão enviadas para o usuário. Uma action escolhe qual página deve ser enviada de acordo com a lógica de negócio e dos dados que foram recebidos na requisição.

Interceptors

Além do FilterDispatcher na camada Controle, existem também outros componentes chamados Interceptors. São componentes que executam antes e depois que uma requisição seja executada por uma action. As principais tarefas que ele executa são validação de dados, conversão de tipos e upload de arquivos. O objetivo é tanto separar quanto reutilizar essas tarefas básicas que são muito comuns no tratamento de requisições. Além de disponibilizar vários interceptors padrões para a maioria das tarefas, o Struts2 possibilita que o programador construa interceptors que atendam as necessidades das aplicações [3].

ValueStack

ValueStack é um componente gerenciado pelo *framework* que serve como área de armazenamento dos dados associados ao processamento das requisições. Esses dados são manipulados enquanto as actions processam a requisição, e são consultados, quando as results renderizam as páginas de resposta. O objetivo deste elemento na arquitetura é de tirar a necessidade de transferência de dados por toda a aplicação, mas sim, centralizá-los em um local.

3.1.2 Exemplo de Aplicação

Este simples exemplo de aplicação utilizando o *framework* Struts2 tem por objetivo mostrar em um nível um pouco mais baixo, que arquivos devem ser criados e como devem ser configurados. A aplicação consiste de uma página onde o usuário entra com seu nome e após ser clicado um botão submit, a aplicação devolve uma outra página com uma mensagem de recepção customizada. Para isto, serão mostrados dois arquivos de configuração: ”web.xml“ e ”struts.xml“, a action ”HelloWorld.java“, e duas páginas (*Results*): ”home.jsp” e ”helloWorld.jsp”.

No Quadro 3.1.1 é exibido o arquivo ”web.xml” onde deve ser definido o FilterDispatcher como entrada para as requisições e quais url’s devem ser tratadas por ele. A linha 9 do arquivo indica que o FilterDispatcher irá tratar todas as requisições de entrada. Já as linhas de 11 à 13, indicam qual será a página inicial da aplicação.

Quadro 3.1.1: Arquivo “web.xml” - Struts2

```
1.    <web-app>
2.    <filter>
3.        <filter-name>struts2</filter-name>
4.        <filter-class>org.apache.struts2.dispatcher.FilterDispatcher
5.        </filter-class>
6.    </filter>
7.    <filter-mapping>
8.        <filter-name>struts2</filter-name>
9.        <url-pattern>/*</url-pattern>
10.   </filter-mapping>
11.   <welcome-file-list>
12.       <welcome-file>home.jsp</welcome-file>
13.   </welcome-file-list>
14.   </web-app>
```

No arquivo ”struts.xml“ apresentado no Quadro 3.1.2, devem ser feitas as declarações de actions que serão utilizadas, assim como quais páginas devem ser apresentadas em cada caso. No nosso exemplo, após ser executada a action ”HelloWorld”, será analisado a string de retorno do método *execute*, caso seja ”success“ será renderizada a página ”HelloWorld.jsp”.

Quadro 3.1.2: Arquivo “struts.xml” - Struts2

```
1.    <action name="HelloWorld" class="HelloWorld">
2.        <result name=’’success’’>HelloWorld.jsp</result>
3.    </action>
```

O arquivo ”home.jsp“ apresentado no Quadro 3.1.3 é a primeira página enviada para o usuário. São apresentados uma caixa de texto e um botão para submeter o formulário.

Como o atributo name do textfield da página ”home.jsp“ tem o mesmo valor *name* de um atributo da action *HelloWorld*, o valor enviado pelo usuário é automaticamente atribuído a esta variável dentro da action. É invocado então o método execute que retorna a string ”success”. O Quadro 3.1.4 apresenta o conteúdo do arquivo “HelloWorld.java”.

Como está configurado no ”struts.xml“, a página ”HelloWorld.jsp“ é enviada para o usuário já que o retorno do método execute da action foi ”success”. No Quadro 3.1.5 é exibido o conteúdo do arquivo “HelloWorld.jsp”.

Quadro 3.1.3: Arquivo “home.jsp” - Struts2

```
1.    <%@ page contentType="text/html; charset=UTF-8" %>
2.    <%@ taglib prefix="s" uri="/struts-tags" %>
3.    <html>
4.        <head>
5.            <title>Name Collector</title>
6.        </head>
7.        <body>
8.            <h4>Enter your name </h4>
9.            <s:form action=’’HelloWorld’’>
10.               <s:textfield name="name" label="Your name"/>
11.               <s:submit/>
12.           </s:form>
13.       </body>
14.    </html>
```

Quadro 3.1.4: Arquivo “HelloWorld.java” - Struts2

```
1.    public class HelloWorld {
2.       private static final String GREETING = "Hello ";
3.       public String execute() {
4.           setMensagem( GREETING + getName() );
5.           return "success";
6.       }
7.       private String name;
8.       private String mensagem;
9.       public String getName() {
10.          return name;
11.      }
12.      public void setName(String name) {
13.          this.name = name;
14.      }
15.      public String getMensagem(){
16.          return mensagem;
17.      }
18.      public void setMensagem( String msg){
19.          this.mensagem = msg;
20.      }
21.   }
```

Quadro 3.1.5: Arquivo “HelloWorld.jsp” - Struts2

```
1.    <%@ page contentType="text/html; charset=UTF-8" %>
2.    <%@ taglib prefix="s" uri="/struts-tags" %>
3.    <html>
4.       <head>
5.           <title>HelloWorld</title>
6.       </head>
7.       <body>
8.           <h3>Custom Greeting Page</h3>
9.           <h4><s:property value="mensagem"/></h4>
10.      </body>
11.   </html>
```

3.2 Framework JavaServer Faces (JSF)

O JSF é um *framework* desenvolvido pela Sun Microsystems junto com a Java Community Process, projetado para ser flexível e fácil de utilizar [29]. O *framework* disponibiliza vários componentes de interface que encapsulam diversas funcionalidades comuns em aplicações para

web. Entre as características do JSF, podemos citar:

- Permite ao desenvolvedor a criação de interfaces gráficas através de um conjunto de componentes de interface pré-definidos;
- Fornece um conjunto de tags JSP para acessar os componentes;
- Permite a reutilização de componentes das páginas;
- Associa os eventos do lado cliente com os manipuladores dos eventos do lado servidor;

3.2.1 Arquitetura MVC no JSF

O JSF assim como o Struts2, tem uma estrutura MVC bem definida. Uma das principais diferenças do JSF está na visão. Os componentes da visão no JSF são organizados em uma árvore hierárquica chamada *Component Tree*. A seguir são apresentadas as camadas da arquitetura MVC no JSF que podem ser vistos na Figura 3.2.

Controlador

O controlador do JSF é constituído de um servlet chamado FacesServlet, de um ou mais arquivos de configuração, de backing beans e de um conjunto de manipuladores de ações e eventos. O FacesServlet é responsável por tratar as requisições que chegam do cliente e executar uma série de passos até que seja enviada a resposta. Esses passos são:

1. Criar um component tree que represente a requisição do usuário, caso já não exista um;
2. Aplicar os valores recebidos na requisição no componente criado anteriormente;
3. Processar validação correspondente ao componente sobre os dados recebidos;
4. Atualizar os backing beans de modelo associados ao componente;
5. Caso necessário, objetos de negócio são invocados para realização de tarefas como verificar autenticidade ou consultar produto por exemplo;
6. Ao final, baseado nos resultados retornados pela lógica de negócio, uma página deve ser escolhida para ser renderizada e enviada para o usuário.

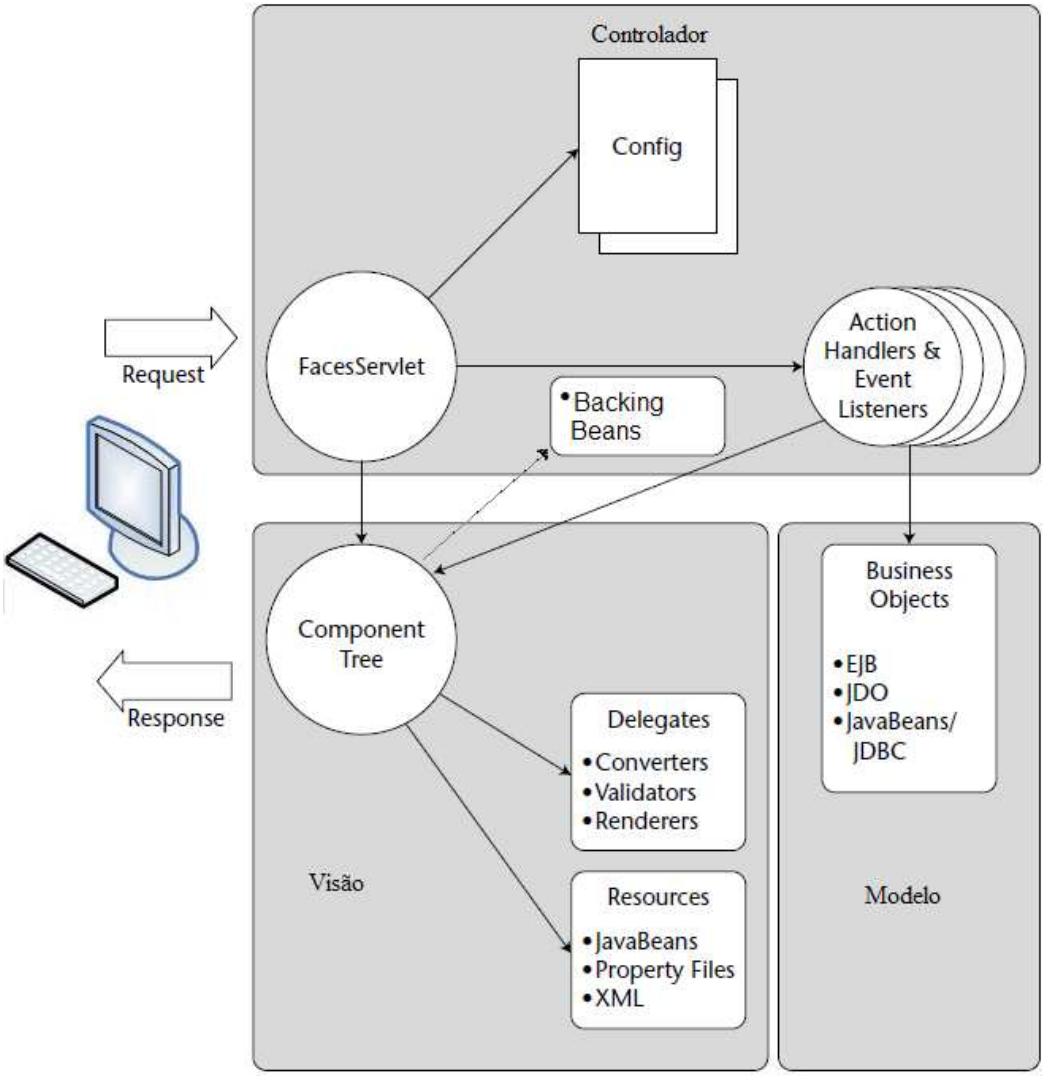

Figura 3.2: Arquitetura MVC no JSF. (Adaptado de [8])

Modelo

A camada Modelo do JSF utiliza recursos como EJB (Enterprise JavaBeans) ou o *framework* Spring, que são mais indicados para a implementação da lógica de negócios. Quem geralmente invoca a execução de tarefas da camada Modelo são os backing beans associados aos formulários.

Visão

Cada componente gráfico da *component tree* possui seus respectivos conversores, validadores, renderizadores, arquivos de configuração e JavaBeans. As tarefas principais de um

component tree são:

a) Renderizar o componente, geralmente com a geração de linguagem de marcação;

b) Manipular os eventos do componente;

c) Validar os valores recebidos na requisição.

Geralmente a Visão de uma aplicação JSF é composta por JSP's, as quais comtêm uma ou mais component trees para seu conteúdo. A Figura 3.3 apresenta a forma em que são vistos os componentes gráficos tanto no lado cliente quanto no lado servidor.

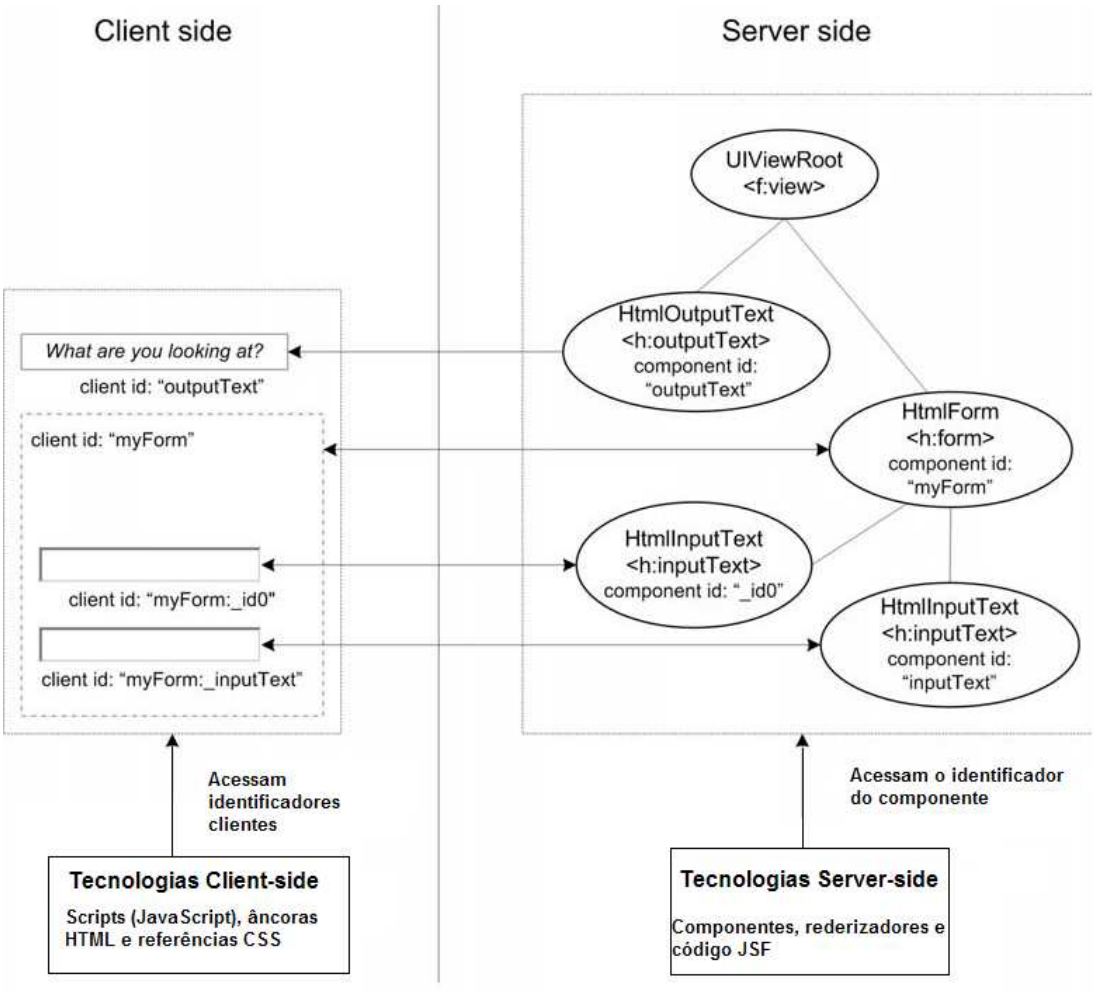

Figura 3.3: Component Tree no JSF. [12]

3.2.2 Exemplo de Aplicação

Nesta seção será mostrado um exemplo simples para que seja melhor entendido que componentes e arquivos de configuração são necessários e como é a relação entre eles. O exemplo é o mesmo da subseção 3.1.2.

O arquivo ”web.xml” apresentado no Quadro 3.2.1 é chamado de descritor da aplicação. Nele é configurado o servlet que processará as requisições que no caso é o FacesServlet, e é informado que todas as requisições que chegarem com url iniciando com “/faces/” deverão ser mapeadas para o servlet.

Quadro 3.2.1: Arquivo “web.xml” - JSF

```
1.    <?xml version="1.0" encoding="UTF-8"?>
2.    <web-app>
3.       <display-name>Hello, World!</display-name>
4.       <description>Welcome to JavaServer Faces</description>
5.       <servlet>
6.          <servlet-name>Faces Servlet</servlet-name>
7.          <servlet-class>
8.                javax.faces.webapp.FacesServlet
9.          </servlet-class>
10.          <load-on-startup>1</load-on-startup>
11.       </servlet>
12.      <servlet-mapping>
13.         <servlet-name>Faces Servlet</servlet-name>
14.         <url-pattern>/faces/*</url-pattern>
15.     </servlet-mapping>
16.   </web-app>
```

Para executar a tarefa de controlador no JSF, é necessário que se crie um *backing bean*, que nada mais é que uma classe java responsável por armazenar alguns dados do usuário, e possuir métodos para acessar, se necessário, a camada de negócio do sistema e então retornar uma string que será utilizada para escolher qual página da visão deve ser enviada para o usuário em seguida. O Quadro 3.2.2 apresenta a classe User utilizada no exemplo.

Para a camada de visão serão necessários dois arquivos JSP, um onde o usuário entra com seu nome, e outro onde é apresentado uma mensagem. O Quadro 3.2.3 apresenta o conteúdo do arquivo ”home.jsp“ que contém uma caixa de texto para a entrada do nome e um botão para submissão do formulário. No arquivo também está configurado para que seja executado o método criaMensagem do *backing bean* quando o botão for clicado.

O arquivo ”welcome.jsp“ descrito no Quadro 3.2.4 faz acesso ao backing bean *User* e exibe a mensagem ao usuário.

Para que todos esses componentes possam funcionar, é necessário que sejam corretamente

Quadro 3.2.2: Arquivo “User.java” - JSF

```
1.    public class User{
2.       String nome;
3.       String mensagem;
4.       public void setNome(String nome){
5.          this.nome=nome;
6.       }
7.       public String getNome(){
8.          return nome;
9.       }
10.      public void setMensagem(String mensagem){
11.         this.mensagem=mensagem;
12.      }
13.      public String getMensagem(){
14.         return mensagem;
15.      }
16.      public String criaMensagem(){
17.         mensagem=''Bem vindo ''+nome;
18.        return ''welcome'';
19.      }
20.   }
```

Quadro 3.2.3: Arquivo “home.jsp” - JSF

```
1.    <%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
2.    <%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
3.    <html>
4.       <body>
5.          <f:view>
6.             <h:form>
7.                Digite o nome:
8.                <h:inputText id="nome" value="#{user.nome}"/>
9.                <h:commandButton value="OK"
10.                action="#{user.criaMensagem}"
11.               />
12.            </h:form>
13.         </f:view>
14.      </body>
15.   </html>
```

Quadro 3.2.4: Arquivo “welcome.jsp” - JSF

```
1.   <%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
2.   <%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
3.   <html>
4.      <body>
5.          <f:view>
6.             <h:form>
7.                 <h:outputText  value="#{user.mensagem}">
8.             </h:form>
9.          </f:view>
10.      </body>
11.   </html>
```

configurados. No JSF essa configuração deve ser feita no arquivo ”faces-config.xml“. Nele devem ser declarados os backing beans, regras de navegação, e outros elementos que forem utilizados como validadores e conversores. No nosso exemplo, foi adicionada uma regra de navegação para que quando for processada a requisição da página ”home.jsp“, e o método do

backing bean retornar a string ”welcome“, seja apresentada em seguida a página ”welcome.jsp”. O arquivo “faces-config.xml” é apresentado no Quadro 3.2.5.

Quadro 3.2.5: Arquivo “faces-config.xml” - JSF

```
1.   <managed-bean>
2.      <managed-bean-name>user</managed-bean-name>
3.      <managed-bean-class>User</managed-bean-class>
4.      <managed-bean-scope>session</managed-bean-scope>
5.   </managed-bean>
6.   <navigation-rule>
7.      <from-view-id>/home.jsp</from-view-id>
8.      <navigation-case>
9.         <from-outcome>welcome</from-outcome>
10.        <to-view-id>/welcome.jsp</to-view-id>
11.     </navigation-case>
12.  </navigation-rule>
```

3.3 Framework SpringMVC

SpringMVC é um módulo do *Framework* Spring correspondente a camada de apresentação, responsável pela interface com o usuário e pelo tratamento das requisições. Os desenvolvedores do Spring decidiram escrever seu próprio web *framework* por acharem que tanto o popular *framework* Jakarta Struts como também outros *frameworks*, tinham uma separação insuficiente entre a camada de manipulação de requisições, a camada modelo, e a camada de apresentação [17].

3.3.1 Arquitetura do SpringMVC

A arquitetura do SpringMVC é bem definida. Seus principais componentes são: *DispatcherServlet*, *HandlerMapping*, *Controller*, *ViewResolver*, *Model*, *ModelAndView* e *View*, que podem ser visualizados na Figura 3.4. A descrição desses componentes junto a um exemplo serão mostrados na próxima subseção.

3.3.2 Exemplo de Aplicação e Descrição dos Componentes

Nesta subseção será feita uma descrição dos principais componentes da arquitetura do SpringMVC, assim como a descrição dos arquivos necessários, acompanhado de um exemplo de aplicação semelhante aos das subseções anteriores, que apresenta uma mensagem para o usuário.

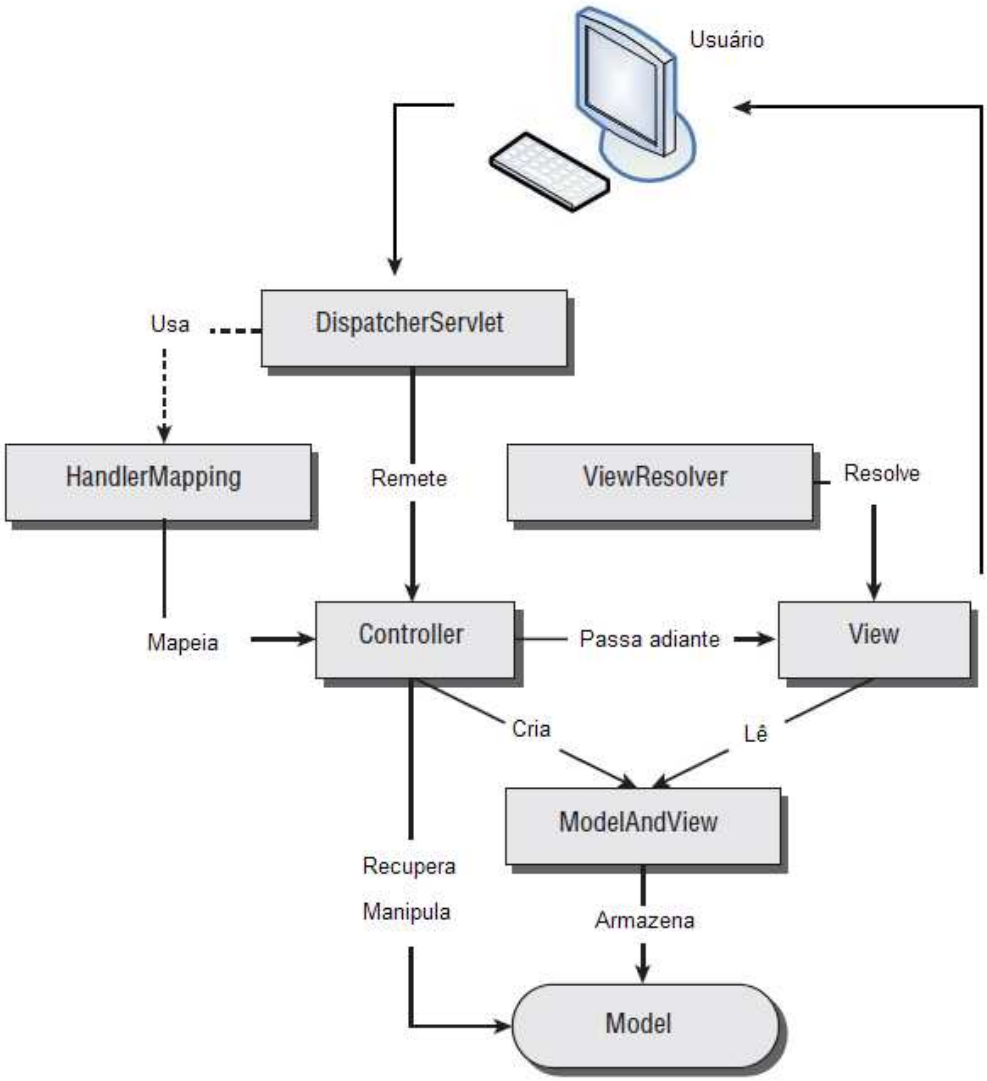

Figura 3.4: Arquitetura do SpringMVC. (Adaptado de [32])

DispatcherServlet e HandlerMapping

O *DispatcherServlet* é o controlador principal de uma aplicação SpringMVC. Ele é o servlet que recebe todas as requisições que chegam e gerencia todos os elementos necessários para processá-las. O *HandlerMapping* é o componente utilizado pelo *DispatcherServlet* que mapeia uma determinada URL de entrada para um determinado *Controller*. Esses dois componentes já são implementados pelo *framework*. O que se faz necessário, é estar declarado o DispatcherServlet no arquivo ”web.xml“, assim como todos os mapeamentos necessários de URL’s para *Controllers*.

O Quadro 3.3.1 apresenta o conteúdo do arquivo ”web.xml“ com a declaração de um *DispatcherServlet*, identificando que todas as URL’s terminadas em ”.form“ devem ser processadas por ele. Também é configurado a utilização do arquivo ”example-servlet.xml“.

Quadro 3.3.1: Arquivo “web.xml” - SpringMVC

```
1.   <web-app>
2.      <servlet>
3.         <servlet-name>example</servlet-name>
4.         <servlet-class>
5.           org.springframework.web.servlet.DispatcherServlet
6.         </servlet-class>
7.         <load-on-startup>1</load-on-startup>
8.      </servlet>
9.      <servlet-mapping>
10.         <servlet-name>example</servlet-name>
11.        <url-pattern>*.form</url-pattern>
12.     </servlet-mapping>
13.  </web-app>
```

O arquivo ”example-servlet.xml“ apresentado no Quadro 3.3.2, deve armazenar o contexto da aplicação. Existe uma convensão para o seu nome, que deve ser sempre o nome do DispatcherServlet, no caso, ”example“ seguido de ”-servlet.xml“. Este arquivo deve armazenar todas as informações relativas aos *Controllers*, *HandlerMappings* e outros componentes da arquitetura. O mapeamento das URL’s para os controladores específicos podem ser inseridos nesse arquivo dentro de uma declaração de um bean *SimpleUrlHandlerMapping*. O mapeamento utilizado no arquivo ”example-servlet.xml“ indica que as URL’s referenciadas a ”/home“, serão tratadas pelo *Controller* ”homeController“ e as referenciadas para ”/search“, serão tratadas pelo *Controller* ”searchController”.

É importante ressaltar que o *framework* disponibiliza várias outras formas de mapeamento além da utilização do *SimpleUrlHandlerMapping*.

Controllers

São os componentes responsáveis por tratar as requisições de páginas específicas. Ao receber a requisição do *DispatcherServlet*, um controller transfere-a para outras classes responsáveis por fazer o processamento e ao fim, coleta os dados do processamento e seleciona uma *view* para ser enviada ao usuário como resposta [26]. Todo *Controller* deve implementar alguma interface Controladora. Neste exemplo, será utilizado a interface AbstractController. O Quadro 3.3.3 apresenta o conteúdo do arquivo ”HomeController.java”.

Quadro 3.3.2: Arquivo “example-servlet.xml” - SpringMVC

```
1.   <bean id="homeController" class="HomeController">
2.      <property name="cacheSeconds" value="120"/>
3.   </bean>
4.   <bean class="org.springframework.web.servlet.
5.                handler.SimpleUrlHandlerMapping">
6.      <property name="urlMap">
7.         <map>
8.            <entry key="/home" value-ref="homeController" />
9.            <entry key="/search" value-ref="searchController" />
10.        </map>
11.        </property>
12.   </bean>
13.   <bean id="viewResolver"
14.      class="org.springframework.web.servlet.view.
15.               InternalResourceViewResolver">
16.      <property name="prefix">
17.         <value>/WEB-INF/jsp/</value>
18.      </property>
19.      <property name="suffix">
20.         <value>.jsp</value>
21.      </property>
22.   </bean>
```

Quadro 3.3.3: Arquivo “HomeController.java” - SpringMVC

```
1.   public class HomeController extends AbstractController {
2.      public ModelAndView handleRequestInternal(
3.      HttpServletRequest request,
4.      HttpServletResponse response) throws Exception {
5.         ModelAndView mav = new ModelAndView("hello");
6.         mav.addObject("message", "Hello World!");
7.         return mav;
8.      }
9.   }
```

Model

O modelo é constituído por uma coleção de objetos que representam os dados que serão renderizados por uma View. Esses objetos podem armazenar resultados de operações realizadas por componentes da lógica de negócio da aplicação. O *framework* combina também as funcionalidades do Model e da View em uma única classe chamada ModelAndView. Isto se deve ao fato de que um *Controller* deve rotornar ambos os objetos para o *DispatcherServlet*.

Considerando a classe HomeController apresentada anteriormente, o *Model* que é adicionado no objeto *ModelAndView* é apenas um objeto com nome “message” e com valor “Hello World!”. Em uma aplicação real, o *Model* poderá conter outros objetos mais complexos criados através de chamadas a lógica de negócios. Por exemplo, o *Model* poderia ser um Array de produtos disponíveis, ou qualquer outra classe que seja necessária para exibir informações para o usuário.

View e ViewResolver

Assim como nos outros *frameworks*, a View representa os componentes gráficos de interface com o usuário. Após ser selecionada por um *Controller*, uma view renderiza uma página obtendo dados do Model e então é enviada para o navegador do cliente. O SpringMVC traz já várias implementações de Views para se utilizar. Entre elas podemos citar JSP, FreeMarker, PDF, Excel e JasperReports [26].

Uma vez retornado um objeto *ModelAndView* por um *Controller* para o *DispatcherServlet*, se faz necessário identificar uma View apropriada para que seja enviada ao usuário. Para isto, o *DispatcherServlet* utiliza um componente do *framework* chamado ViewResolver, que no nosso caso, analisa o arquivo "example-servlet.xml" e identifica as regras declaradas. Como se deseja uma classe chamada "hello" (declarado na criação do ModelAndView no Quadro 3.3.3, linha 5), o *ViewResolver* de acordo com os prefixos e sufixos configurados, retorna o caminho: "/WEB-INF/jsp/hello.jsp".

Agora com a página JSP já definida, ela é então renderizada e enviada para o usuário. Podemos notar que a página utiliza o objeto "message" representando o *Model* que foi criado no *Controller*. Neste exemplo, é exibida a seguinte mensagem no navegador do usuário: "Welcome! Hello World!". O conteúdo do arquivo "hello.jsp" é apresentado no Quadro 3.3.4.

Quadro 3.3.4: Arquivo "hello.jsp" - SpringMVC

```
1.   <%@ page contentType="text/html" %>
2.   <%@ taglib prefix="c" uri="http://java.sun.com/jstl/core" %>
3.   <html>
4.      <head><title>Rantz</title></head>
5.      <body>
6.         <h2>Welcome!</h2>
7.         <c:out value="${message}"/>
8.      </body>
9.   </html>
```

Capítulo 4
A implementação

Para que realmente fosse possível avaliar os *frameworks* escolhidos para este estudo, fez-se necessário a realização de um estudo de caso. A estratégia adotada foi a de desenvolver uma mesma aplicação web em cada um destes *frameworks*. O objetivo desta implementação, foi de descobrir que características ou soluções um *framework* oferece que os outros não oferecem ou que oferecem de uma forma menos adequada. A implementação permite que sejam avaliados tópicos como facilidade de configuração, facilidade de aprendizado e se os *frameworks* oferecem suporte para utilização de Inversão de Controle.

A aplicação a ser desenvolvida consiste em um gerenciador online de notas, de presença e de conteúdo das aulas, a qual poderá ser utilizada no Curso de Informática da Universidade Estadual do Oeste do Paraná (UNIOESTE). Para isto, o sistema deve também gerenciar o cadastro de disciplinas, turmas, professores, alunos e anos letivos.

Foram utilizados para a implementação a IDE NetBeans versão 6.1 e JDK versão 1.6.0. Para a persistência dos dados foi utilizado o banco orientado a objetos Db4o versão 6.4. Todos rodando em sistema Linux Ubuntu versão 8.04. A versão dos *frameworks* utilizados foram:

- JSF versão 1.2;
- Struts2 versão 2.0.11;
- SpringMVC versão 2.5;

4.1 Requisitos Funcionais

Para um melhor entendimento da aplicação a ser desenvolvida, são apresentados nos Quadros a seguir os requisitos funcionais da aplicação.

Quadro 4.1.1: Requisito Funcional **Cadastrar Ano Letivo**.

RF-01 Cadastrar Ano Letivo	
Descrição	O sistema deve possibilitar o cadastramento de um novo ano letivo sempre que as atividades de um novo ano se inicia.
Pré-condições e Entradas	Um secretário deve estar logado no sistema. Entrada: Ano corrente.
Saída	Um novo registro relativo ao ano em questão no sistema.
Passos para obter sucesso	i.Clicar na opção Iniciar Ano Letivo; ii.Entrar com o ano; iii.Clicar no botão Salvar.

Quadro 4.1.2: Requisito Funcional **Cadastrar Professor**.

RF-02 Cadastrar Professor	
Descrição	O sistema deve possibilitar o cadastramento de professores.
Pré-condições e Entradas	Um secretário deve estar logado no sistema. O secretário entra com vários dados relativos a um professor, como: nome, idade, endereço, etc.
Saída	Um novo registro de professor no sistema.
Passos para obter sucesso	i.Clicar na opção Cadastrar Professor; ii.Entrar com os dados relativos ao professor; iii.Clicar no botão Salvar.

Quadro 4.1.3: Requisito Funcional **Cadastrar Aluno**.

RF-03 Cadastrar Aluno	
Descrição	O sistema deve possibilitar o cadastramento de alunos.
Pré-condições e Entradas	Um secretário deve estar logado no sistema. O sistema recebe como entrada vários dados relativos a um aluno, como: nome, idade, endereço, etc.
Saída	Um novo registro de aluno no sistema.
Passos para obter sucesso	i.Clicar na opção Cadastrar Aluno; ii.Entrar com os dados relativos ao aluno; iii.Clicar no botão Salvar.

Quadro 4.1.4: Requisito Funcional **Cadastrar Disciplina**.

RF-04 Cadastrar Disciplina	
Descrição	O sistema deve possibilitar o cadastramento de disciplinas.
Pré-condições e Entradas	Um secretário deve estar logado no sistema. O sistema recebe como entrada vários dados relativos a uma disciplina, como: código, nome, série, carga horária, etc.
Saída	Um novo registro de disciplina no sistema.
Passos para obter sucesso	i.Clicar na opção Cadastrar Disciplina; ii.Entrar com os dados relativos à disciplina; iii.Clicar no botão Salvar.

Quadro 4.1.5: Requisito Funcional **Cadastrar Turma**.

RF-05 Cadastrar Turma	
Descrição	O sistema deve possibilitar o cadastramento de turmas.
Pré-condições e Entradas	Um secretário deve estar logado no sistema. O sistema recebe como entrada vários dados relativos a uma turma, como: disciplina, se é prática ou teórica, horário, etc.
Saída	Um novo registro de turma no sistema.
Passos para obter sucesso	i.Clicar na opção Cadastrar Turma; ii.Entrar com os dados relativos à turma; iii.Clicar no botão Salvar.

Quadro 4.1.6: Requisito Funcional **Importar Matrícula**.

RF-06 Importar Matrícula	
Descrição	O sistema deve possibilitar a importação de matrículas que podem vir da secretaria acadêmica sempre que as atividades de um novo ano se inicia.
Pré-condições e Entradas	Um secretário deve estar logado no sistema. Entrada: Arquivo em formato específico com as matrículas
Saída	Confirmação dos alunos matriculados nas respectivas disciplinas
Passos para obter sucesso	i.Clicar na opção Importar Matrícula; ii.Escolher o arquivo adequado; iii.Clicar em Importar.

Quadro 4.1.7: Requisito Funcional **Matricular Aluno**.

RF-07 Matricular Aluno	
Descrição	O sistema deve possibilitar a matricula de alunos nas turmas.
Pré-condições e Entradas	Um secretário deve estar logado no sistema. O sistema recebe como entrada o aluno e a turma a qual ele esta se matriculando.
Saída	Confirmação do aluno matriculado na turma em questão.
Passos para obter sucesso	i.Clicar na opção Matricular Aluno; ii.Escolher o aluno; iii.Escolher a turma; iv.Clicar no botão Salvar.

Quadro 4.1.8: Requisito Funcional **Consultar Notas**.

RF-08 Consultar Notas	
Descrição	O sistema deve possibilitar que alunos, professores e secretários possam fazer consultas de notas.
Pré-condições e Entradas	Um secretário, aluno, ou professor, deve estar logado no sistema. O sistema recebe como entrada uma turma ou um aluno.
Saída	As notas do aluno ou dos alunos da turma escolhida.
Passos para obter sucesso	i.Clicar na opção Consultar Notas; ii.Se for um aluno que estiver consultando, será selecionado automaticamente todas as disciplinas sendo cursadas por ele naquele ano; Caso seja um professor, deve-se escolher uma turma; Caso seja um secretário, deve-se escolher um aluno. iii.Clicar no botão Consultar.

Quadro 4.1.9: Requisito Funcional **Consultar Faltas**.

RF-09 Consultar Faltas	
Descrição	O sistema deve possibilitar que alunos, professores e secretários possam fazer consultas das faltas dos alunos.
Pré-condições e Entradas	Um secretário, aluno, ou professor, deve estar logado no sistema. O sistema recebe como entrada uma turma ou um aluno.
Saída	As faltas do aluno ou dos alunos da turma escolhida.
Passos para obter sucesso	i.Clicar na opção Consultar Faltas; ii.Se for um aluno que estiver consultando, será selecionado automaticamente todas as disciplinas sendo cursadas por ele naquele ano; Caso seja um professor, deve-se escolher uma turma; Caso seja um secretário, deve-se escolher um aluno. iii.Clicar no botão Consultar.

4.2 Casos de Uso Implementados

Inicialmente, pretendia-se implementar todas as funcionalidades da aplicação. Porém, devido a complexidade envolvida no aprendizado de vários recursos que os *frameworks* disponibi-

Quadro 4.1.10: Requisito Funcional **Lançar Notas**.

RF-10 Lançar Notas	
Descrição	O sistema deve possibilitar que os professores possam lançar as notas dos alunos.
Pré-condições e Entradas	Um professor deve estar logado no sistema. O sistema recebe como entrada uma turma e as notas dos alunos.
Saída	As notas ficam registradas no sistema.
Passos para obter sucesso	i.Clicar na opção Lançar Notas; ii.Escolher a turma; iii.Registrar a nota de cada aluno; iv.Clicar no botão Salvar.

Quadro 4.1.11: Requisito Funcional **Lançar Conteúdo das Aulas**.

RF-11 Lançar Conteúdo das Aulas	
Descrição	O sistema deve possibilitar que os professores possam registrar o conteúdo das aulas em suas respectivas datas.
Pré-condições e Entradas	Um professor deve estar logado no sistema. O sistema recebe como entrada a turma, o conteúdo e a data da aula.
Saída	O conteúdo fica registrado no sistema.
Passos para obter sucesso	i.Clicar na opção Lançar Conteúdo das Aulas; ii.Escolher a turma; iii.Escolher a data; iv.Inserir o conteúdo da aula; v.Clicar no botão Salvar.

lizam, foi necessário implementar apenas algumas destas funcionalidades, permitindo assim que se pudesse avaliar os três *frameworks*. As funcionalidades implementadas foram as seguintes:

- Gerenciamento de Professores, Alunos e Disciplinas por parte da secretaria; - Controle das contas de acesso ao sistema; - Usuários podem se logar e deslogar no sistema.

Os casos de uso implementados são apresentados na Figura 4.1.

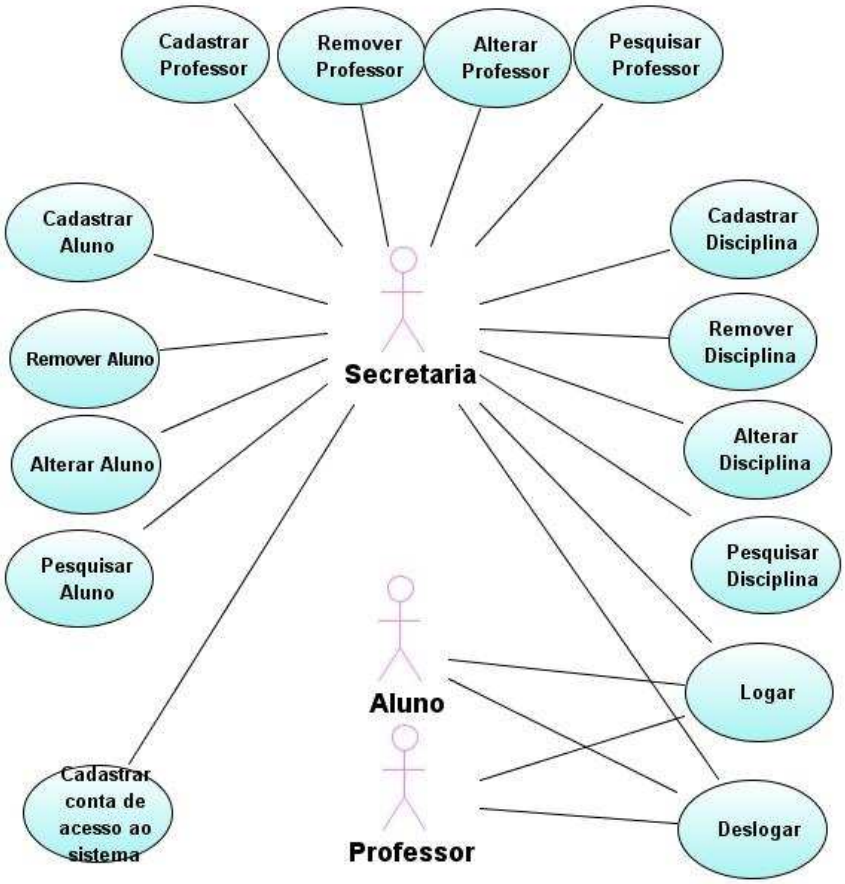

Figura 4.1: Casos de uso implementados.

Capítulo 5

Análise e Avaliação dos Frameworks

Este capítulo tem por objetivo apresentar uma avaliação sobre os *frameworks*, utilizando para isto vários critérios. Foram avaliados tanto critérios não-técnicos como documentação e treinamentos, assim como critérios técnicos como configuração e mecânismos de validação dos dados. Ao final é apresentada uma avaliação geral dos *frameworks*, apresentando sugestões de quais projetos seriam mais beneficiados por cada ferramenta. A Tabela 5.1 apresenta todos os critérios utilizados para a avaliação dos *frameworks* utilizados neste trabalho.

Seção	**Critério avaliado**
Seção 5.1	Documentação
Seção 5.2	Treinamentos
Seção 5.3	Evolução da ferramenta
Seção 5.4	Demanda de profissionais
Seção 5.5	Disponibilidade de profissionais
Seção 5.6	Configuração
Seção 5.7	Curva de aprendizado
Seção 5.8	Decoração de páginas
Seção 5.9	Inversão de Controle
Seção 5.10	Mecanismos de Validação

Tabela 5.1: Critérios utilizados para avaliação dos *frameworks*.

5.1 Documentação

Avaliou-se em um primeiro momento, a qualidade e a facilidade de compreensão da documentação disponibilizada pela empresa desenvolvedora do *framework* ou pela comunidade. Esta documentação foi analisada avaliando-se a organização, abrangência, objetividade e clareza.

O JSF se destacou neste critério. É o que possui a maior quantidade de documentação e é a que está melhor organizada. Os exemplos utilizados são simples, facilitando o aprendizado.

O Struts2 não disponibiliza uma boa documentação, talvez por ser este um *framework* bas-

tante novo. Isto pode prejudicar e atrasar o desenvolvimento, pois algumas dúvidas somente podem ser esclarecidas com o auxílio da comunidade do *framework* ou dos mantenedores do projeto.

Dentre os *frameworks* analisados, o SpringMVC é o que possui a documentação de maior dificuldade de entendimento. Em certas partes, é difícil distinguir a documentação do *framework* web com a do projeto Spring como um todo. Seus exemplos são de difícil compreensão por serem extremamente técnicos. Passa a impressão de que este *framework* não é recomendado para projetos de pequeno porte.

Em um segundo momento, foi feita uma pesquisa para verificar o número de livros publicados sobre cada *framework*. A pesquisa foi realizada utilizando-se o site Amazon.com e os resultados obtidos são mostrados na Figura 5.1. Analisando os comentários dos leitores no mesmo site, pode-se afirmar que os livros apresentam um bom conteúdo e abordam bem as características dos *frameworks*. Contudo, o JSF se destacou por ter um número maior de livros publicados.

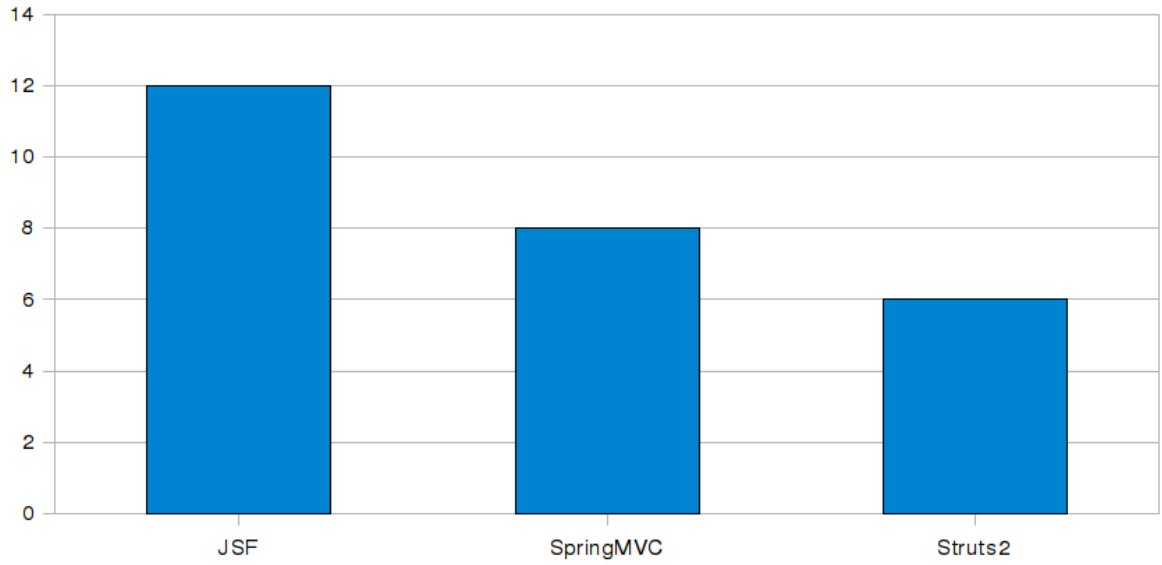

Figura 5.1: Número de livros publicados. Fonte: Amazon.com, Novembro de 2008.

5.2 Treinamentos

Foi analisado se as empresas mantenedoras dos *frameworks* disponibilizam treinamentos e certificações. Este é um fator importante para os profissionais que buscam conhecer e aprofundar seus conhecimentos, possibilitando um bom aproveitamento dos recursos disponibilizados pelos *frameworks*.

A empresa SpringSouce, mantenedora do projeto Spring, oferece treinamento e certificação para profissional em desenvolvimento com o *framework* Spring. Mas não existe nenhum treinamento específico para o módulo SpringMVC, obrigando o profissional interessado a ter que fazer o treinamento completo. Esses treinamentos ocorrem apenas nos Estados Unidos, Europa, e na Austrália [27]. A empresa Sun, que mantém o projeto do JSF, é dentre as três empresas, a que oferece os melhores treinamentos e certificações. São oferecidos vários tipos de treinamento, incluindo desde o básico em aplicações web, até aqueles que cobrem os recursos do *framework* JSF. Quanto a certificação, é oferecido para desenvolvedor de componentes web. Além disso, os treinamentos ocorrem em vários países, incluindo o Brasil [30]. Já a empresa Apache, mantenedora do projeto Struts2, não oferece qualquer tipo de treinamento para o *framework*.

Neste quesito o *framework* JSF se destacou, pois é o mais fácil de se conseguir treinamento oficial e sem sair do Brasil. Mesmo existindo treinamento para o SpringMVC, este é oferecido apenas em Países distantes do Brasil, o que pode tornar esta prática inviável em relação ao custo de deslocamento. Já o Struts2 falha bastante neste quesito, pois não se tem qualquer treinamento ou certificação oficial para o mesmo.

5.3 Evolução da Ferramenta

Este tópico tem por objetivo avaliar o quanto os *frameworks* estão evoluindo. Esta evolução passa uma certa confiança de que a ferramenta não vai deixar de ter melhorias e correções de *bugs* das versões atuais. Foi realizada uma análise das versões que foram liberadas no sites das empresas mantenedoras para os anos de 2007 e de 2008. Como pode ser visto na Figura 5.2, os resultados obtidos mostram que todos os *frameworks* deste estudo estão recebendo atenção por parte das empresas mantenedoras e que em média a cada poucos meses se tem uma nova versão das ferramentas.

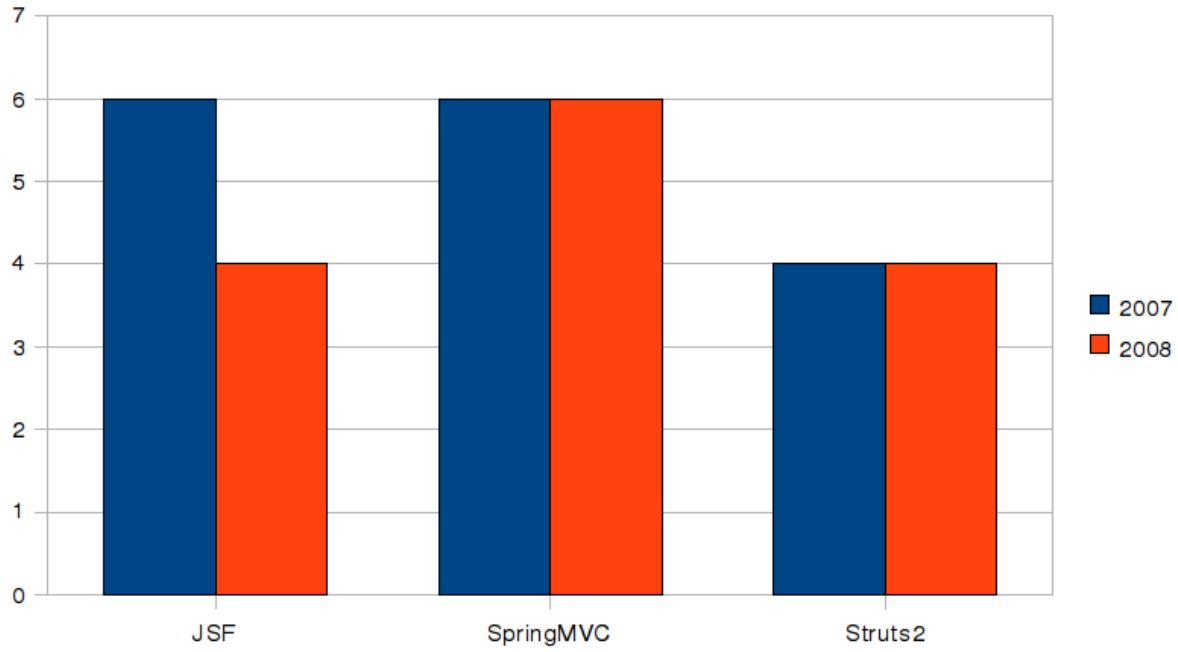

Figura 5.2: Número de releases dos *frameworks* disponibilizados em 2007 e 2008.

5.4 Demanda de Profissionais

Para verificar como está a busca por profissionais com conhecimento nos *frameworks*, realizou-se uma pesquisa em agências de emprego virtuais nacionais e internacionais. A partir desta pesquisa, pode-se analisar o quanto o conhecimento de cada *framework* pode contribuir para integrar um desenvolvedor ao mercado de trabalho e quais *frameworks* possuem maior ou menor procura por parte das empresas.

Para se realizar essa pesquisa foi necessário encontrar empresas virtuais que disponibilizassem o serviço de pesquisa de empregos. Foi encontrada uma lista de sites de emprego nacionais no site “http://br.geocities.com/ilseinfo” onde foram escolhidos o Catho.com e o InfoJobs.com.br por retornarem um maior número de vagas para algumas pesquisas. Já para os empregos internacionais, foram encontradas indicações dos melhores sites em “http://www.consumersearch.com/job-sites“. O site Dice.com foi considerado pelas pesquisas como o melhor para a área de tecnologia. Já o SimplyHired.com é um motor de pesquisa que une resultados de vários outros sites. Foram analisados além dos três *frameworks* o Struts, que é a versão a partir da qual o Struts2 foi construído. Uma contagem do número de vagas ofere-

cidas para desenvolverdores com conhecimento em cada um dos *frameworks* foi realizada e os resultados são mostrados nas Figuras 5.3, 5.4, 5.5 e 5.6.

Através da análise dos resultados, verifica-se que existe uma grande demanda de profissionais com conhecimento no *framework* JSF, tanto a nível nacional quanto internacional. Isto significa que as empresas estão apostando mais neste *framework* do que nos outros para a utilização em novos projetos de aplicações web. Verifica-se também que as empresas ainda estão utilizando bastante a versão mais antiga do Struts, contudo, os valores baixos obtidos para o Struts2 podem ser consequência da utilização do termo Struts para projetos que já utilizem Struts2.

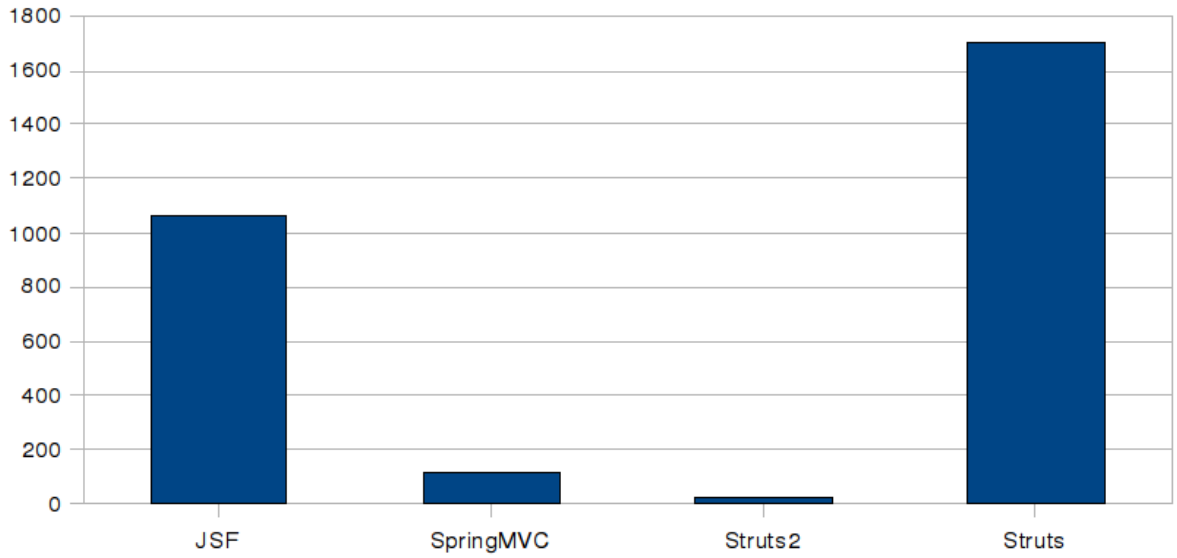

Figura 5.3: Número de ofertas de emprego encontradas no site Dice.com, Novembro de 2008.

5.5 Disponibilidade de Profissionais

Antes de escolher um web *framework* é importante conhecer a disponibilidade de mão-de-obra no mercado de trabalho que esteja preparada para trabalhar com o *framework*. Escolher um que seja pouco conhecido, implicará em custos de treinamento de pessoal. Para avaliar a disponibilidade de profissionais habilitados a trabalhar com estes *frameworks* foi feito uma pesquisa em sites que hospedam currículos.

Na busca por dados internacionais, encontrou-se muitos sites pagos. Nos sites gratuitos, os dados obtidos revelaram um número muito pequeno de resultados e por isso, foram desconsid-

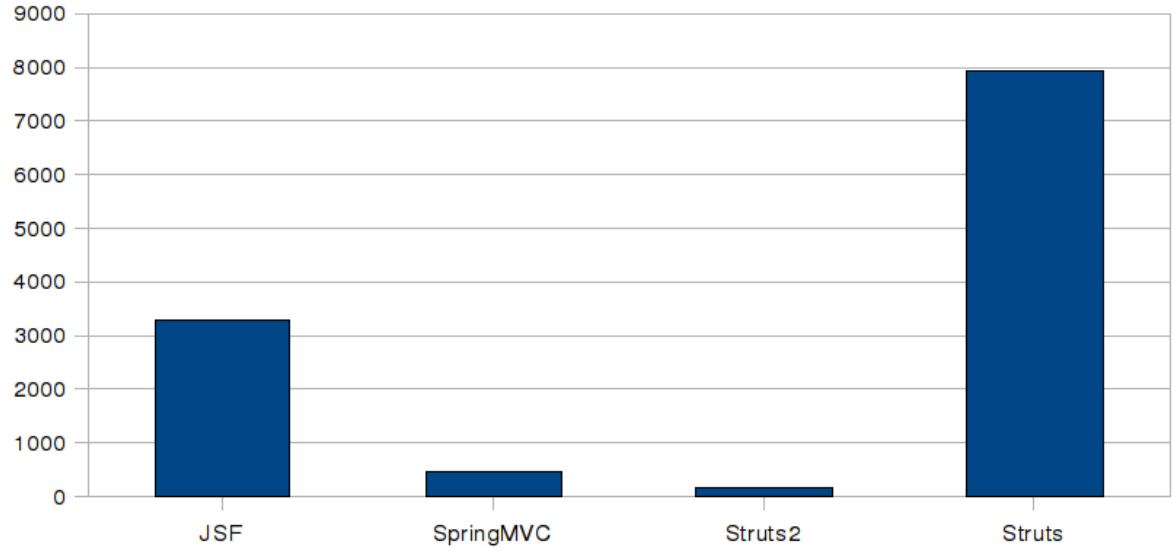

Figura 5.4: Número de ofertas de emprego encontradas no site SimplyHired.com, Novembro de 2008.

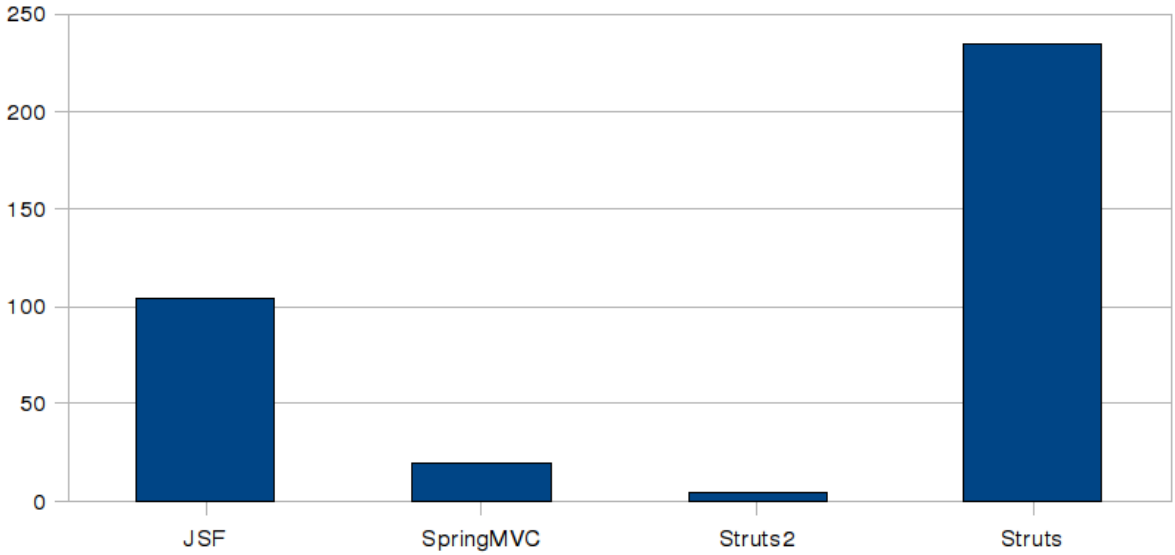

Figura 5.5: Número de ofertas de emprego encontradas no site Catho.com, Novembro de 2008.

erados. Já para a pesquisa nacional foi encontrado um ótimo site, o APInfo.com. A Figura 5.7 mostra os resultados da pesquisa realizada no site em novembro de 2008, revelando a disponibilidade de profissionais com habilidades nos web *frameworks* no mercado nacional.

O JSF foi o *framework* que apresentou mais profissionais preparados, com um número bem maior comparado aos outros dois. Isto significa que em novos projetos utilizando o *framework*,

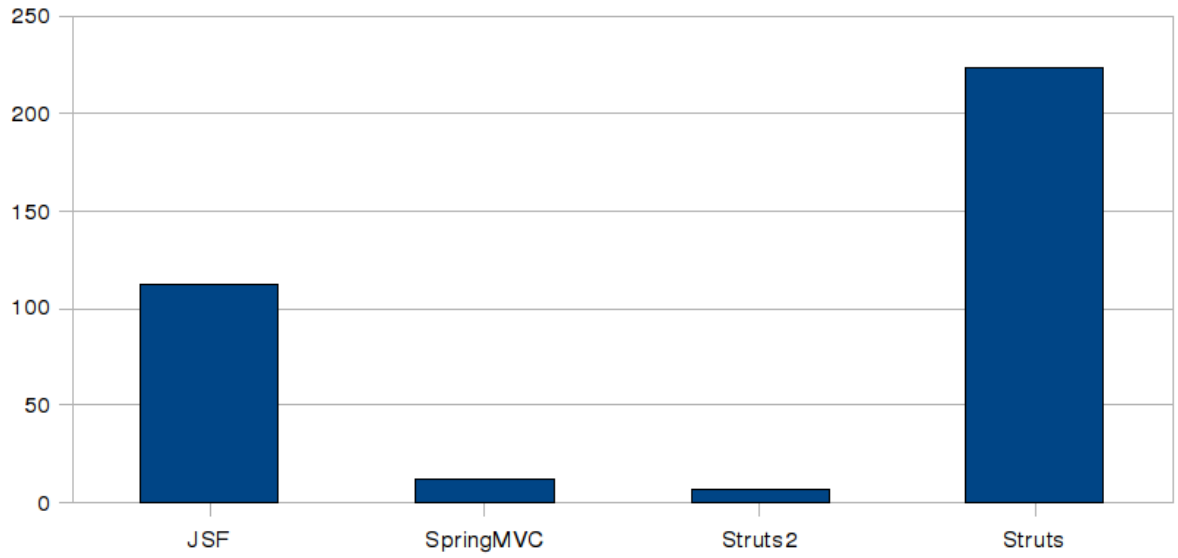

Figura 5.6: Número de ofertas de emprego encontradas no site InfoJobs.com.br, Novembro de 2008.

as chances de as empresas encontrarem mão-de-obra especializada será maior.

5.6 Configuração

Neste quesito, os *frameworks* Struts2 e JSF mostraram-se bastante flexíveis, pois apresentam maior configuração à medida que são utilizados recursos mais avançados. Possuem herança de configuração entre os controladores, o que centraliza a configuração em um único ponto, facilitando a manutenção do mesmo.

O SpringMVC é o *framework* onde a configuração exige mais linhas de código nos arquivos XML. Sua configuração inicial é mais complexa pois obriga o desenvolvedor à declarar uma série de objetos para gerenciar os recursos de internacionalização, fluxo de navegação e seus controladores. Contudo, após esta etapa, não é necessário configurar o mapeamento de páginas JSP, o que pode ser considerado uma vantagem em relação ao Struts2 e ao JSF quando se está desenvolvendo aplicações onde há bastante mudanças no fluxo entre as páginas.

5.7 Curva de Aprendizado

Esta seção tem por objetivo avaliar a facilidade de aprendizagem na utilização dos recursos disponibilizados pelos *frameworks*. O JSF é o único *framework* dentre os três que utiliza

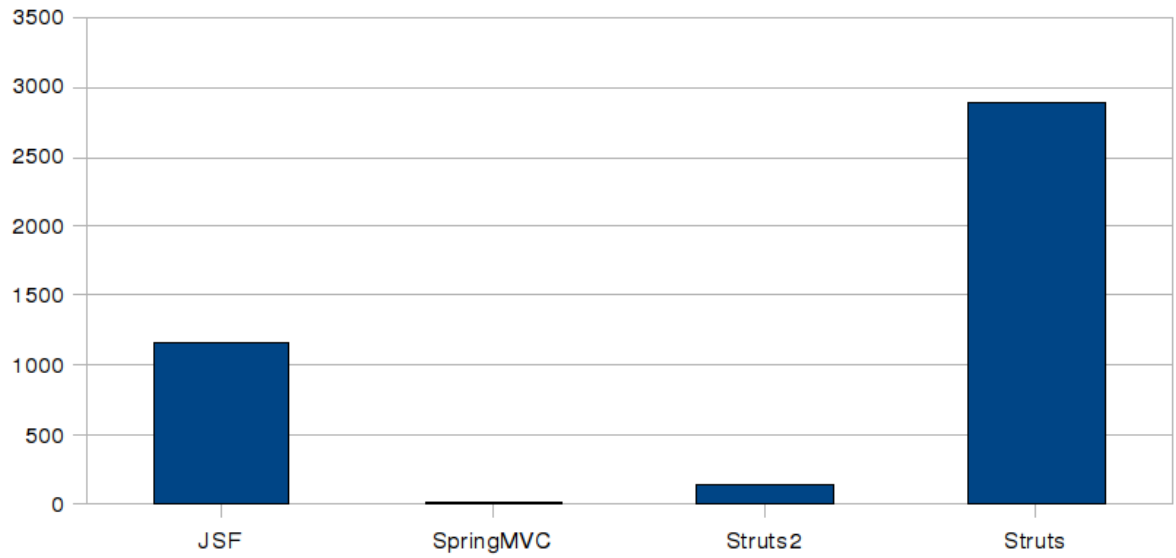

Figura 5.7: Número de currículos de profissionais com conhecimento nos *frameworks* encontrados no site APInfo.com, Novembro de 2008.

o esquema *drag-and-drop*, ou seja, um esquema em que os componentes gráficos podem ser arrastados e soltos diretamente na posição desejada na tela. O código-fonte necessário para os componenetes também são todos criados pela ferramenta. Por disponibilizar este recurso, verificou-se que a criação de componentes visuais para as páginas é de fácil aprendizagem e de rápido tempo de desenvolvimento. No JSF, é possível criar aplicações simples sem precisar conhecer muitos de seus recursos, exigindo que sejam conhecidos a medida que a complexidade das aplicações aumentem.

O Struts2 possui a curva de aprendizagem um pouco mais complexa que a do JSF. Uma vez que exige que o desenvolvedor conheça um maior número de recursos disponibilizados como interceptadores, filtros e de como manipular objetos na pilha ValueStack. Estes recursos precisam ser conhecidos quando se desenvolve aplicações de qualquer complexidade.

O SpringMVC também possui uma curva de aprendizado maior que a do JSF, já que sua configuração é mais complexa. Além disso, em aplicações mais complexas são utilizados vários recursos que estão no *framework* Spring, tornando-se necessário ter um mínimo de conhecimento em Spring.

5.8 Decoração de páginas

Um problema encontrado utilizando os *frameworks* foi no desenvolvimento de telas para interface com o usuário que seguissem um esquema de *layout* semelhante em todas as páginas e também na criação de menus de navegação que fossem facilmente utilizados por toda a aplicação. Desta forma, procurou-se por ferramentas que pudessem resolver este problema. Foram encontradas então diversas soluções que se diziam resolver o problema. Optou-se então por testar a utilização de dois *frameworks* de decoração e *layout* de páginas web que pudessem ser utilizados em conjunto com os *frameworks* JSF, Struts2 e SpringMVC: Tiles e Sitemesh.

Ambas as soluções trabalham interceptando as requisições para páginas estáticas ou dinâmicas da aplicação, verificam as propriedades e o conteúdo, e por fim geram a página resultante de acordo com esquemas de *layout* que devem ser configurados em arquivos XML [2][23].

O JSF funcionou de forma adequada com o Sitemesh mas já com o Tiles, foram encontrados problemas. Mesmo após feitas todas as configurações e incluídas todas as bibliotecas necessárias para sua execução, não conseguiu-se integrar as ferramentas. Os componentes gráficos do JSF não foram renderizados na tela, a qual ficava sem conteúdo.

Já com o Struts2 e com o SpringMVC os *frameworks* para decoração e *layout* funcionaram de forma adequada não apresentando problemas nos testes realizados.

5.9 Inversão de Controle

O Struts2 não oferece recursos de Inversão de Controle. Para utilizá-los, o Struts2 precisa ser integrado ao *framework* Spring utilizado muitas vezes para a camada de negócios nas aplicações. Porém, esta integração não é muito complexa, bastando adicionar algumas linhas de configuração no arquivo “struts.xml”.

O SpringMVC por fazer parte do projeto do *framework* Spring, também utiliza os recursos do mesmo. Para isto, não é necessário nenhum tipo de configuração já que possui integração nativa.

O JSF é o único *framework* que traz consigo suporte completo para Inversão de Controle. Para sua utilização é necessário a configuração das propriedades dos beans gerenciados no arquivo de configuração XML. Apenas deve-se ter cuidado com a compatibilidade do escopo dos beans na aplicação, para que o *framework* consiga injetar as propriedades no escopo correto.

Quando se planeja utilizar o *framework* Spring para a camada de negócios da aplicação, os três *frameworks* podem utilizar perfeitamente a Inversão de Controle. Mas para pequenas aplicações onde geralmente não se utiliza o Spring, o JSF se torna a melhor opção, oferecendo suporte nativo para Inversão de Controle.

5.10 Mecanismos de Validação

Uma das soluções encontradas que se destacam para a validação dos dados é o *Commons Validator*. Esse *framework* é um projeto desenvolvido pela fundação Apache e é largamente utilizado nas aplicações em geral [1]. Os *frameworks* que dão suporte a esta solução de acordo com os testes realizados são o SpringMVC e o Struts2.

Os três *frameworks* oferecem seus próprios validadores padrão e também suporte para que o desenvolvedor crie validadores de acordo com suas necessidades. Contudo, o SpringMVC obriga o desenvolvedor a utilizador também o *framework* Spring para a camada de negócios, pois os módulos de validação fazem parte do mesmo.

Com o Struts2, os validadores são construídos através de *interceptors*, mas desta forma, a validação ocorre na camada de Controle, separada da lógica de negócios (Modelo), o que pode não ser interessante para determinadas aplicações, pois permite que dados relacionados possam localizar-se em locais diferentes.

O JSF fornece a validação através de componentes e foi considerada a mais fácil de utilizar. É possível especificar *tags* e atributos de validação dentro do código JSP. Já quando é necessário uma validação mais robusta, é possível criar os validadores integradados com a camada Modelo e utilizá-los com a mesma facilidade.

5.11 Avaliação Geral dos Frameworks

O *framework* que mais teve pontos positivos nos critérios avaliados foi o JSF. Seu aprendizado é favorecido por causa da boa documentação existente. Ele se torna bastante indicado para projetos onde os componentes visuais sofrem muitas alterações, pois o mesmo tem um ótimo gerenciamento destes além de oferecer suporte para *drag-and-drop* (Arrastar e soltar). O *framework* também se mostra indicado para aplicações que vão desde pequeno até grande porte. Isto porque os recursos precisam ser conhecidos a medida que a aplicação vai se tor-

nando mais complexa. O número de profissionais disponíveis com conhecimento no *framework* é outro fator positivo, que torna-o uma boa opção para novos projetos onde há a necessidade de contratação de novos desenvolvedores.

O SpringMVC através da análise dos critérios avaliados, não seria muito indicado para aplicações simples que não envolvem muita complexidade. O *framework* oferece ótimos recursos para o desenvolvimento, mas geralmente se faz necessário utilizar o *framework* Spring completo, que é um ótimo *framework*, mas para a camada de negócios. Desta forma, o SpringMVC seria indicado para empresas que já vem desenvolvendo aplicações complexas com o Spring e que agora precisam de módulos web, ou para novos projetos onde é notável a complexidade da camada de negócios e não se tenha muitas restrições de tempo para o desenvolvimento.

O Struts2 assim como os JSF e o SpringMVC mostra-se ter muitos recursos importantes que facilitam o desenvolvimento, mas seria mais indicado para projetos que já estão sendo desenvolvidos em Struts, a versão mais antiga do *framework*. Isto porque a facilidade para o aprendizado do Struts2 seria beneficiado já que possui alguns recursos semelhantes. Para novos projetos é importante que as empresas pensem em um bom treinamento no *framework* já que não existem muitos profissionais preparados. Vale considerar também, que o Struts2 possui ótima integração com o *framework* Spring, sendo também uma boa opção para projetos mais complexos que utilizem o Spring para a camada de negócios.

Capítulo 6

Conclusões

Este trabalho teve como objetivo fazer um estudo de três *frameworks* java para desenvolvimento web, e apresentar pontos positivos e negativos dos mesmos de forma a auxiliar os desenvolvedores na hora de optar por um deles.

Inicialmente o Capítulo 1 fez uma introdução sobre as aplicações web, sobre os problemas encontrados no desenvolvimento dessas aplicações e sobre os *frameworks*. Foram apresentados também os objetivos gerais e específicos, assim como a motivação e a estrutura do trabalho. O Capítulo 2 apresentou uma revisão sobre tecnologias para desenvolvimento de aplicações web e sobre tecnologias java. Foi apresentado entre outros o MVC, um padrão de projeto que é muito utilizado e que vem implementado pelos *frameworks*.

Já no Capítulo 3 foi feito um estudo mostrando os principais detalhes sobre os *frameworks* escolhidos. Foi mostrado quais são os principais componentes da arquitetura e como eles estão organizados para implementar o padrão MVC. O Capítulo 3 também mostra para cada *framework* um exemplo simples para seu melhor entendimento. O Capítulo 4 descreve os detalhes de implementação e o estudo de caso que foi utilizado para se utilizar os *frameworks* de forma mais prática.

No Capítulo 5 foram apresentados de acordo com vários critérios uma análise e avaliação dos *frameworks*. Entre os critérios utilizados temos: Disponibilidade de profissionais com conhecimento nos *frameworks*, evolução da ferramenta, curva de aprendizado e configuração. Ao final, foram apresentados alguns tipos de projeto que seriam mais beneficiados por determinado *framework*. Além disso, foram apresentados possibilidade de integração entre *frameworks*, uma vez que estes já podem estar sendo utilizados pelas empresas.

Na implementação do estudo de caso descrito no Capítulo 4, foram encontradas muitas

dificuldades em várias etapas e com todos os *frameworks* escolhidos. Algumas delas foram:

- A documentação muitas vezes não cobria adequadamente os recursos disponibilizados pelos *frameworks*, fazendo com que se perdesse muito tempo na procura de documentação correta. As dicas e soluções muitas vezes foram encontradas em fóruns e comunidades de desenvolvimento de aplicações;

- Os *frameworks* disponibilizam muitos componentes e muitas formas de se resolver o mesmo problema. Contudo, para cada situação existe uma escolha que é a mais adequada. Como nem tudo está documentado, isto se tornou um fator negativo para o trabalho, pois em várias situações foi escolhido aquela que demandaria de um tempo maior para sua configuração e utilização.

Mesmo com essas dificuldades, e em específico para a aplicação descrita no Capítulo 4, o Struts2 foi o *framework* em que foram encontrados menos problemas na implementação. Desta forma, ele seria o mais indicado pelo autor deste texto, para a construção de pequenas e médias aplicações e similares a realizada neste trabalho. O motivo foi que os elementos da arquitetura do Struts2 necessários para a aplicação, foram mais rapidamente assimilidos comparado aos outros dois *frameworks*.

Como existem muitas questões importantes relativas aos *frameworks* que podem auxiliar as empresas de desenvolvimento, e que a implementação citada no Capítulo 4 não pode ser completamente implementada, ficam como sugestões para trabalhos futuros:

- Finalizar o trabalho de implementação descrito no Capítulo 4 utilizando o *framework* Struts2. Como este é um *framework* bastante novo, seria importante analisar mais a fundo sua arquitetura e seus poderosos componentes;

- Um estudo sobre o *framework* Spring para a camada de negócios, onde poderão ser avaliados os recursos de Inversão de Controle e de programação orientada à aspectos;

- Utilizar o estudo de caso para fazer um estudo de tecnologias ou *frameworks* para persistência dos dados, tais como o Hibernate e JPA. Poderão ser avaliadas questões como desempenho e mapeamento objeto-relacional.

Referências Bibliográficas

[1] APACHE. **Commons Validator**. Disponível em <http://commons.apache.org/validator>. Acessado em Novembro de 2008.

[2] APACHE. **Tiles Overview**. Disponível em <http://tiles.apache.org>. Acessado em Novembro de 2008.

[3] BROWN, D.; DAVIS, C. M. S. S. **Struts2 in Action**. Manning, 2008.

[4] CAVANESS, C. **Programming Jakarta Struts**. O'Reilly, 2002.

[5] COOPER, J. W. Using design patterns. **CACM**, [S.l.], v.41, n.6, p.65–69, Junho, 1998.

[6] DESIGNS, R. **Web Frameworks**. Disponível em <http://raibledesigns.com/rd/tags/webframeworks>. Acessado em Junho de 2008.

[7] DOWNEY, T. **Web Development with Java Using Hibernate, JSPs and Servlets**. Miami, FL, USA: Springer, 2007.

[8] DUDNEY, B.;LEHR, J. B. R. **Mastering JavaServer Faces**. Wiley Publishing, Inc., 2004.

[9] FOUNDATION, A. S. **Apache Struts2**. Disponível em <http://struts.apache.org/2.x/>. Acessado em Julho de 2008.

[10] FOWLER, M. **Containers de Inversão de Controle e o padrão Dependency Injection**. Disponível em <http://www.javafree.org/content/view.jf?idContent=1>. Acessado em Junho de 2008.

[11] GAMMA, E.; HELM, R. J. R. V. J. **Design Patterns: Elements of Reusable Object-Oriented Software**. Reading, MA: Addison Wesley, 1995.

[12] GEARY, D.;HORSTMANN, C. **Core JavaServer Faces**. 2. ed. Prentice Hall, 2007.

[13] HALL, M.; BROWN, L. **Core Servlets and JavaServer Pages**. 2. ed. Prentice Hall PTR, 2003.

[14] HAMMANT, P. **Inversion of Control**. Disponível em <http://www.picocontainer.org/Inversion+of+Control>. Acessado em Julho de 2008.

[15] IBM. **Design with the JSF architecture: Exploring design patterns used in the JavaServer Faces framework**. Disponível em <http://www.ibm.com/developerworks/web/library/wa-dsgnpatjsf.html>. Acessado em Agosto de 2008.

[16] JAMES GOSLING, BILL JOY, G. S. G. B. **The Java Language Specification**. 3. ed. ADDISON-WESLEY, 2005.

[17] JOHNSON, R. **Expert One-on-One J2EE Design and Development**. Wrox Press, 2003.

[18] KURT A. GABRICK, D. B. W. **J2EE and XML Development**. Manning, 2002.

[19] MARKIEWICZ, M. E.; LUCENA, C. J. Object oriented framework development. **ACM Crossroads Student Magazine**, [S.l.], 2001.

[20] MCKINNON, L. **Beginning XML**. 4. ed. Wrox Press, 2007.

[21] MINETTO, E. L. **Frameworks para Desenvolvimento em PHP**. Editora Novatec, 2007.

[22] NASH, M. **Java Frameworks and Components: Accelerate Your Web Application Development**. Cambridge Universty Press, 2003.

[23] OPENSYMPHONY. **SiteMesh Overview**. Disponível em <http://opensymphony.com/sitemesh>. Acessado em Novembro de 2008.

[24] ROUGHLEY, I. **Starting Struts 2**. C4Media, 2006.

[25] SCHMIDT, D. C.; GOKHALE, A. N. B. Frameworks: Why they are important and how to apply them effectively. Nashville.

[26] SMEETS, B.;LADD, S. **Building Spring 2 Enterprise Applications**. Apress, 2007.

[27] SPRINGSOURCE. **Spring Home**. Disponível em <http://www.springsource.com>. Acessado em Outubro de 2008.

[28] SUN. **Java Annotations**. Disponível em <http://java.sun.com/docs/books/tutorial/java/javaOO/annotations.html>. Acessado em Julho de 2008.

[29] SUN. **JSF Overview**. Disponível em <http://java.sun.com/javaee/javaserverfaces/overview.html>. Acessado em Junho de 2008.

[30] SUN. **Treinamentos na Sun**. Disponível em <http://www.sun.com/training>. Acessado em Outubro de 2008.

[31] TEMPLE, A.; MELLO, R. F. C. D. T. S. M. **Programação Web com Jsp, Servlets e J2EE**. 2004.

[32] VAN DE VELDE, T.; SNYDER, B. C. L. S. H. A. B. N. **Beginning Spring Framework 2**. Indianapolis, IN: Wiley Publishing, Inc., 2008.

[33] WONG, W. C.; EYADAT, M. N. S. Degree of freedom - experience of applying software framework. In: PROCEEDINGS OF THE THIRD INTERNATIONAL CONFERENCE ON INFORMATION TECHNOLOGY: NEW GENERATIONS (ITNG'06), 2006. [s.n.], 2006.