UNIVERSIDADE FEDERAL DE SANTA CATARINA

Análise e Comparação de Frameworks para Desenvolvimento Web em Java

Thiago Roberto dos Santos

Florianópolis – SC

2007 / 2

UNIVERSIDADE FEDERAL DE SANTA CATARINA

DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

CURSO DE SISTEMAS DE INFORMAÇÃO

Análise e Comparação de Frameworks para Desenvolvimento Web em Java

Thiago Roberto dos Santos

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Sistemas de Informação

Florianópolis – SC

2007 / 2

Análise e Comparação de Frameworks para Desenvolvimento Web em Java

Thiago Roberto dos Santos

Trabalho de conclusão de curso apresentado como parte dos requisitos

para obtenção do grau de Bacharel em Sistemas de Informação

Orientador: Vitório Bruno Mazzola

Banca examinadora

Leandro José Komosinski
Rosvelter Coelho da Costa

RESUMO

Este trabalho visa definir critérios para avaliar e comparar tecnologias para a camada de apresentação de aplicações Web desenvolvidas em Java. O ambiente web vem sendo muito utilizado para o desenvolvimento de aplicações. É interessante analisar ferramentas que auxiliem na resolução de problemas na camada de apresentação que, em geral, já foram resolvidos nas aplicações convencionais.

Para realizar o levantamento dos critérios e avaliar algumas soluções disponíveis no mercado, foram construídos protótipos com os frameworks Struts, Spring MVC e JBoss Seam.

A construção de protótipos facilitou a identificação de características comuns e particularidades de cada tecnologia. Foram definidos critérios aos quais foram submetidos os frameworks avaliados. Foi gerado um quadro comparativo para os frameworks estudados para facilitar a visualização dos resultados.

O estabelecimento de critérios, embora subjetivos, deve auxiliar futuras avaliações de frameworks, permitindo que o analista investigue diretamente a classificação do artefato nos critérios pré-estabelecidos.

Com a devida consideração de características especificas da aplicação e da equipe em questão, o trabalho auxilia uma organização no processo de escolha de um framework para desenvolvimento Web em Java.

ÍNDICE DE FIGURAS

SUMÁRIO

INTRODUÇÃO

1.1. Problema

Tem se tornado cada vez mais comum a escolha do ambiente Web para o desenvolvimento de aplicações corporativas. Em decorrência disso, a complexidade das aplicações tem crescido muito rapidamente.

Na plataforma Java, o modelo de objetos de negócio e o mapeamento objeto relacional vêm sendo bem atendidos pela especificação EJB3. Contudo, existe muito trabalho repetitivo na construção de interfaces com o usuário e no tratamento das entradas do sistema, o que prolonga o tempo de desenvolvimento e dificulta a manutenção.

Visando facilitar o desenvolvimento da camada de apresentação, surgiram vários frameworks para diminuir a codificação repetitiva. A grande oferta de soluções dificulta a escolha e a identificação da mais adequada para uma determinada aplicação ou conjunto de aplicações.

1.2. Objetivos

Este trabalho tem como principal objetivo definir critérios para análise e comparação das tecnologias para camada de apresentação mais adequadas para sistemas gerenciais de pequeno e médio porte que utilizem a tecnologia EJB para a camada de negócio e persistência de dados.

De maneira complementar, o trabalho também objetiva:

- Avaliar os frameworks Struts 2, Spring MVC e Jboss Seam

- Elaborar um quadro comparativo dos frameworks avaliados
- Identificar características comuns aos frameworks.

1.3. Justificativa

Devido ao grande número de organizações que disponibilizam serviços para Internet e Intranet, o desenvolvimento de aplicações Web tornou-se um ramo muito importante na indústria de software.

A mudança de ambiente trouxe de volta uma série de problemas já resolvidos pelas aplicações tradicionais. Várias soluções já estão disponíveis para resolver tais problemas, mas, devido a grande oferta, é difícil escolher uma solução perene para a organização. Até mesmo identificar quais são as opções disponíveis e estáveis é uma tarefa complexa. Um estudo para identificação e comparação de frameworks seria bastante útil para auxiliar as organizações nessas tarefas.

CONCEITOS FUNDAMENTAIS

1.1 Java

Java é uma linguagem de programação orientada a objeto desenvolvida na década de noventa por James Gosling, na SUN Microsystems. Java tem sintaxe similar às linguagens C e C++. Ela é compilada em bytecodes, que podem ser executados por uma máquina virtual ou por hardware específico (Deitel; Deitel, 2004).

1.2 Plataforma Java

Plataforma Java é uma plataforma para desenvolvimento e execução de programas escritos na linguagem Java. A plataforma não é específica para um hardware ou sistema operacional. As aplicações da plataforma rodam sobre uma máquina virtual, que possuí implementações para diversas arquiteturas. A plataforma é distribuída em três edições diferentes:

- Java Standart Edition (Java SE), para máquinas desktop.
- Java Micro Edition (Java ME), para dispositivos portáteis.
- Java Enterprise Edition (Java EE), para aplicações corporativas.

1.3 Java EE

Java EE é uma edição da plataforma Java voltada para aplicações corporativas, distribuídas, multicamadas.

A plataforma Java EE é definida por uma especificação, considerada informalmente um padrão, pois os fornecedores de servidores de aplicação têm que cumprir certos requisitos para declarar seus produtos compatíveis com Java EE. Ela inclui várias especificações de API como JDBC, RMI, e-mail, JMS entre outras.

1.4 Servlets

"Um Servlet estende a funcionalidade de um servidor, como um servidor Web que serve páginas da Web para um navegador do usuário com o protocolo Http" (DEITEL; DEITEL, 2004, p. 929).

O servlet trabalha com o conceito de requisição e resposta, através do protocolo Http. Um Servlet recebe uma requisição e, baseado nela, fornece uma resposta. A resposta pode ser um documento HTML, XML, entre outros.

O contêiner WEB é responsável pelo ciclo de vida do Servlet, pelo tratamento das múltiplas chamadas que podem ser feitas pelos clientes e também pela tradução de uma chamada http para uma requisição do Servlet.

1.5 Java ServerPages (JSP)

"A tecnologia das Java ServerPages, um extensão da tecnologia dos servlets, simplifica o processo de criação de páginas separando a apresentação do conteúdo. Normalmente, as JSP são utilizadas quando a maior parte do conteúdo enviado ao cliente é texto estático e quando uma

pequena parte do conteúdo é gerada dinamicamente com código Java" (DEITEL; DEITEL, 2004, p. 929).

As JSP combinam o uso de HTML e linguagem Java. Em relação aos Servlets, elas facilitam a divisão de trabalho entre os designers e os programadores, pois permitem a escrita das páginas em HTML intercaladas com código Java.

As JSP são compiladas para Servlets, passando a responder requisições de maneira semelhante a eles. A compilação pode ser feita no momento da instalação da aplicação no servidor ou no momento em que a JSP é requisitada pela primeira vez.

1.6 Bibliotecas de Tags (Tag Libs)

As bibliotecas de tags definem funcionalidades que podem ser usadas em qualquer JSP, em forma semelhante às tags HTML. As tags são processadas no servidor no momento da requisição, podendo gerar código HTML.

As bibliotecas mantêm o código mais legível, pois evitam a mistura de código Java com código HTML. As tags costumam ser mais amigáveis também para os designers, que não estão acostumados em ver trechos de código no meio das páginas.

1.6.1 Java Stantard Tag Library (JSTL)

Java Standard Tag Library encapsula em tags simples as funcionalidades centrais comuns à maioria das aplicações Web. Ela traz tags para iterações, operações condicionais, manipulação de XML, tags para internacionalização e acesso a banco de dados via SQL.

1.7 EJB

Segundo Sriganesh et al. (2006, p. 10) EJB é um padrão para desenvolvimento e instalação de componentes de servidor distribuídos em Java. Ele define um acordo (contrato) entre componentes e servidores de aplicação que possibilita qualquer componente ser executado em qualquer servidor de aplicação compatível.

EJB é sigla de Enterprise Java Beans, especificação de componentes para construção modular de aplicações. A especificação padroniza a implementação da regra de negócio das aplicações.

EJB especifica como um servidor de aplicação prove:

- Persistência de Dados
- Processamento de transações/Controle de Concorrência
- Segurança
- Serviço de troca de mensagens (JMS)
- Invocação remota de procedimentos (RMI/IIOP)
- Disponibilização de Métodos de Negócio como Web Services

1.7.1 API de Persistência Java (JPA)

A API de Persistência Java (JPA) provê mapeamento objeto relacional para JAVA. O mapeamento facilita o desenvolvimento de aplicações orientadas a objeto com dados persistidos em bancos de dados relacionais.

1.7.2 Entidades

Entidades são classes Java que, tipicamente, representam uma tabela uma base de dados relacional. Cada instância da classe, por sua vez, representa uma linha na tabela. As entidades podem se relacionar entre si nas multiplicidades um para um (One-to-One), um para muitos (One-to-Many), muitos para um (Many-to-One) e muitos para muitos (Many-to-Many).

O mapeamento entre as tabela e as entidades são feitas através de simples anotações. As anotações permitem o mapeamento das relações e os ajustes de nome da tabela e das colunas.

1.7.3 Java Persistence Query Language

Java Persistence Query Language é uma linguagem semelhante a SQL, mas pode fazer consultas contra as entidades mapeadas. Dessa forma é possível abstrair a forma de armazenamento dos dados.

1.7.4 Beans de Sessão (*EJB Session Beans*)

Para Sriganesh et al. (2006, p. 91), um Bean de Sessão executa uma operação independente do código cliente que o chama. Beans de Sessão são componentes reusáveis que contêm lógica para processos de negócio.

Beans de Sessão são tipicamente classes de negócio que podem ser publicadas em um contêiner J2EE.

1.7.4.1 Beans de Sessão Sem Estado (Stateless Session Beans)

Um Bean de Sessão sem Estado é um bean que executa conversações que duram apenas uma chamada de método. Eles são chamados de “sem estado” porque não guardam estado entre chamadas. Após cada chamada, o contêiner pode decidir destruir o bean ou recria-lo, limpando toda a informação pertencente a chamadas anteriores. Ele também pode decidir manter a instancia, talvez usando para todos os clientes que querem usar o mesmo bean de seção. O algoritmo exato é específico de cada contêiner (SRIGANESH et al. , 2006, p. 92).

1.7.4.2 Beans de Sessão Com Estado (Stateful Session Beans)

Alguns processos de negócio são naturalmente projetados para conversações de diversas requisições. Um exemplo é o procedimento de compra numa loja virtual. Durante a vista de um usuário ele pode adicionar produtos ao seu carrinho de compras. Cada vez que o usuário adiciona um produto é feita uma nova requisição. Consequentemente os componentes

precisam manter o estado do usuário (como o estado do carrinho de compras) de requisição em requisição (SRIGANESH et al. , 2006, p. 94).

Um bean de sessão com estado mantém o estado dos atributos entre as requisições de um cliente, portanto é interessante usa-lo quando temos transações ou processos de negócio que necessitem de mais de uma chamada do cliente para serem completados.

1.8 Padrões de projeto

Os padrões de projeto são soluções já testadas para resolver problemas recorrentes. Os padrões são documentados e, à medida que se mostrem eficientes, devem ser reutilizados. Os padrões de projeto precisam ser adaptados ao contexto da aplicação. Eles não se propõem a determinar a maneira como o código deve ser escrito, mas como os elementos estão estruturados.

1.9 Padrão MVC

O padrão MVC teve origem no SmallTalk, onde era usado para separar a interface com o usuário dos dados. Dessa forma, mudanças as mudanças de interface causavam menos impacto na camada de dados e vice-versa.

1.9.1 Modelo

O modelo é responsável pelos dados e pelas regras de negócio da aplicação. Ele que interage com os repositórios de dados e executa as operações da aplicação.

1.9.2 Visão

A visão é responsável por apresentar os dados para o usuário. Os mesmos dados podem ser apresentados de diversas formas diferentes, como tabelas, formulários ou gráficos, sem que seja necessário reescrever o código de acesso aos dados para cada forma de apresentação.

1.9.3 Controlador

O controlador é responsável pela comunicação entre as camadas visão e modelo. Ele trata entradas dos usuários, invoca as regras de negócio pertinentes na camada modelo e apresenta os dados através da camada de visão.

1.10 Arquitetura JSP Modelo 2

A arquitetura do modelo 2 é aderente ao padrão MVC trabalhando com uma abordagem para aplicações Web que combina JSP e Servlets. Ele foca nos pontos fortes de cada tecnologia, usando as JSP para a camada de visão e os Servlets como controladores, tratando as entradas dos usuários e se comunicando com a camada modelo. As JSP não contêm nenhuma regra de

negócio, apenas apresentando dados de objetos previamente criados pelos Servlets e pelos objetos de negócio.

1.11 Framework

Segundo Silva (2000, p. 21) “Frameworks são estruturas de classes que constituem implementações incompletas que, estendidas, permitem produzir diferentes artefatos de software. A grande vantagem desta abordagem é a produção de reuso de código e projeto, que pode diminuir o tempo e o esforço exigidos na produção de software”.

Vários frameworks aderentes aos padrões MVC e JSP Modelo 2 serão estudados nos próximos capítulos.

1.12 IoC (Inversão de Controle)

Meu primeiro contato com inversão de controle foi com o controle principal de uma interface com o usuário. As primeiras interfaces eram controladas pela aplicação. Você tinha uma seqüência de comandos como "Digite o nome", "Digite o endereço": seu programa ia requisitando entradas e tratando as respostas para cada um. Com as interfaces gráficas (ou baseadas em tela) o framework de interface com o usuário cotinha o controle e seu programa provia tratadores de eventos para os vários campos na tela. O controle principal do programa foi invertido, passou do seu programa para o framework (FOWLER, 2004).

Inversão de controle é uma característica dos frameworks, porém a variedade de casos em que o termo se aplica às vezes o torna confuso.

1.13 Injeção de Dependência

Segundo Ladd (2006, p. 11) uma especialização de Inversão de Controle, Injeção de Dependência é uma técnica usada por frameworks para unir a aplicação. O framework faz o trabalho de conectar as dependências da aplicação, removendo o código específico para criação de objetos da aplicação.

FRAMEWORKS AVALIADOS

Neste trabalho foram avaliados os frameworks Struts2, Spring MVC e JBoss Seam. Struts 2 foi escolhido por sua popularidade no mercado. A versão 2 trouxe avanços sobre a anterior, especialmente na simplificação dos arquivos de configuração. Spring MVC foi selecionado pela quantidade de artigos escritos sobre o framework. JBoss Seam foi selecionado pois o conceito do framework é semelhante ao do Web Beans, que é um modelo de componentes que integra JSF e EJB3 e pode vir a ser incorporado à especificação Java EE.

Para padronizar a utilização dos frameworks, em todos eles foi utilizada a tecnologia JSP para as visões. Os frameworks em geral deixam em aberto a escolha da tecnologia de visões. Foi escolhida JSP por ser bastante conhecida e compatível com todos os frameworks estudados.

1.1 Struts 2

O Struts é um framework de código aberto desenvolvido pela fundação apache para desenvolvimento de aplicações Web com Java EE que encoraja a programação no modelo MVC(Model-View-Controller). Ele provê um tratador de requisições e de respostas, que auxilia o tratamento das interações entre cliente e servidor. Também é disponibilizado um conjunto de bibliotecas de tags para facilitar a construção de aplicações baseadas em forms HTML.

O Struts oferece duas versões de seu framework, a versão 1, que é reconhecida como o framework mais popular para aplicações Web em Java, e a versão 2.0, que surgiu da união da equipe do projeto Struts com a equipe do

projeto WERBWORK. Anteriormente a versão 2 era conhecida como WEBWORK 2. Este trabalho avaliou a versão 2 do framework, por trazer uma série de recursos novos e pelo fato de que ela deve substituir gradativamente a versão 1.

1.1.1 Configuração

Para configuração de uma aplicação básica feita com Struts são usados pelo menos dois arquivos de configuração, web.xml e struts.xml. Pode ser usado também o arquivo struts.properties, mas ele não será tratado neste trabalho pois é opcional e suas configurações também podem ser feitas nos dois arquivos citados anteriormente.

1.1.1.1 Web.xml

No arquivo web.xml precisamos configurar um filtro para que as requisições passem pelo framework. O filtro deve ser configurado apontando para a classe FilterDispatcher, do framework, conforme o exemplo abaixo.

```
<filter>
    <filter-name>meuFiltro</filter-name>
    <filter-class>org.apache.struts2.dispatcher.FilterDispatcher
      </filter-class>
</filter>

<filter-mapping>
    <filter-name>meuFiltro</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

O nome do filtro (<filter-name>) pode ser definido pelo usuário. Os recursos interceptados podem ser definidos pelo usuário na tag <url-pattern> para que apenas um conjunto de recursos seja interceptado pelo framework. No exemplo acima todos os recursos serão interceptados.

1.1.1.2 Struts.xml

O struts.xml assim como o struts.properties também pode ser pode ser removido da aplicação. Ele pode ser substituído parcialmente por anotações, parâmetros no web.xml, entre outras possibilidades. Para aplicações grandes, podemos fragmentar o arquivo de configuração usando a *tag* <include>, que deve apontar para outro arquivo xml que deve seguir a mesma estrutura do struts.xml.

A *tag* <package> é usada para agrupar configurações que tenham atributos em comum, como interceptores, por exemplo. Usualmente se tratam de configurações de ações, mas podem conter qualquer tipo de configuração (ROUGJLEY, 2006, p. 18).

O arquivo também contém outras configurações que são adicionadas com o decorrer do desenvolvimento da aplicação, como configurações de ações, por exemplo. Esse tipo de uso será relatado nas seções posteriores.

1.1.2 Arquitetura Básica

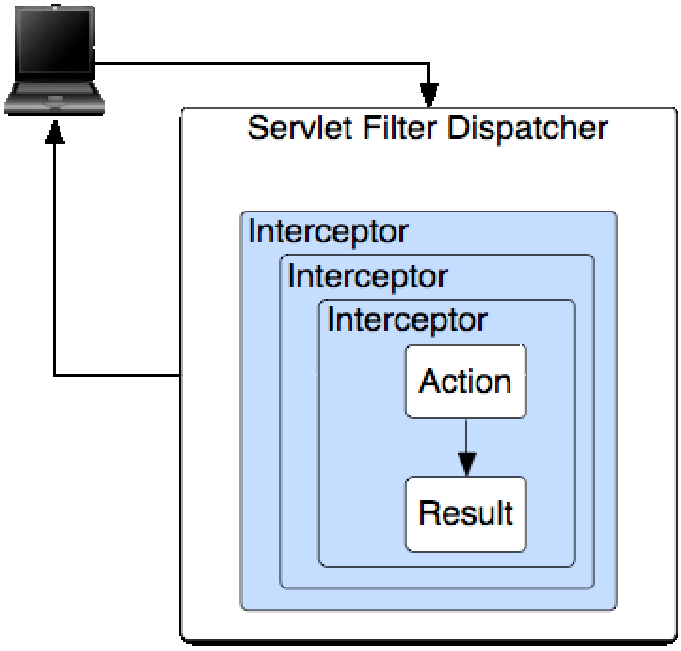

Figura 1 - Arquitetura do Framework Struts

Fonte: Husted (2007)

As requisições aos recursos são captadas pelos filtros de servlet que de acordo com a requisição determina a Ação (*Action*) a ser executada. Interceptores (*Interceptors*) aplicam funcionalidades comuns como validação, tratamento de upload, e fluxo de trabalho (*workflow*). Após os tratamentos, a Ação é executada e os resultados são renderizados no cliente na forma de HTML, PDF, imagens ou outra saída.

A operação de negócio é executada na Ação invocada pelo *framework* e os interceptores fazem tratamentos prévios da requisição, de acordo com as configurações e com o conteúdo enviado pelo cliente.

1.1.3 Ações

O Struts é um framework baseado em ações, portanto as classes de ação são parte central do framework. As ações podem ser usadas de diferentes formas. Na sua forma mais simples, temos uma classe que implementa o método execute (o método pode ter outro nome, desde que esteja indicado em arquivos de configuração), que retorna uma String indicando o resultado da ação, conforme o código abaixo. Opcionalmente, é possível estender classes do framework que auxiliam no desenvolvimento das ações.

```
public class TipoContratoAction extends ActionSupport {
      public String execute() throws Exception {
            if (tipos == null) tipos =
            getTipoContratoDAO().getAllTipoContrato();
            tipoContrato = null;
            return SUCCESS;
      }
}
```

Trecho de Código 1 - Exemplo de Ação do Struts

Alguns conjuntos de ações podem ter resultados comuns, podendo ser criados mais de um método na classe de ação. Nesse caso, quando a ação for invocada dever passado como entrada o método que será executado. Caso não seja informado um método o será executado o método execute.

1.1.3.1 Configuração das ações

A configuração da ação pode ser feita de várias maneiras. A maneira mais comum é a configuração no arquivo struts.xml. Ela é familiar aos usuários de versões anteriores do framework. Segue abaixo a configuração da ação apresentada mostrada no código anterior.

```
<action name="TipoContrato"
            class="prototipo.struts.web.TipoContratoAction">
      <result>/TipoContrato.jsp</result>
</action>
```

Trecho de Código 2 - Configuração da Ação do Struts

No trecho de código acima foi dado um nome para a ação, TipoContrato, que será usado nas páginas que a referenciarem. É definida a classe que implementa a ação no atributo class. Por fim, configuramos a visão a ser apresentada caso a ação seja executada com sucesso. Para ações mais complexas podemos configurar vários tipos de resultados que são apresentados de acordo com o retorno do método execute.

Além da configuração no arquivo, as ações podem ser configuradas através das anotações @Result e @Results para resultados simples e compostos respectivamente. A String retornada pelo método execute deve ser igual a uma das configuradas nas anotações.

O método execute pode, opcionalmente, retornar um objeto da classe Result do framework. A instancia dessa classe define qual visão deve ser apresentada ao usuário.

Existem também plugins que permitem fazer o direcionamento automático usando convenções. Um exemplo é o code-behind plugin. Quando ele é utilizado, as visões são invocadas através da concatenação do nome da ação e do retorno do método execute. Por exemplo, se o nome da ação for “adduser” e o retorno do método execute for “sucess” a visão apresentada será “adduser-sucess.jsp” (ROUGJLEY, 2006, p. 22).

1.1.4 Acesso aos dados da requisição

Para tomar decisões de como a ação irá trabalhar e prover dados para persistência de objetos em bancos de dados, a ação pode precisar ter acesso aos valores da requisição ou de dados de formulários (ROUGJLEY, 2006, 23).

Os dados resultantes das ações usualmente também são necessários para a renderização da visão. Tanto os dados de entrada (valores da requisição ou de formulários) quanto os dados de saída podem ser acessados por métodos de acesso padrão dos JavaBeans (*getters* e *setters*).

Os dados de entrada são convertidos automaticamente de *String* para qualquer tipo primitivo. Podem ser criadas extensões para converter as entradas em outros objetos complexos. A construção de visões que utilizam e alimentam dados das ações será tratada em seções posteriores.

1.1.5 Interceptores

Interceptores funcionam de maneira semelhante aos filtros dos Servlets, podendo assim ser ordenados e acessar dados da atual requisição. Para a seleção da ação a ser executada, extração dos parâmetros de entrada e tratamento de exceção, por exemplo, são utilizados interceptores. Alguns interceptores são pré-configurados pela configuração padrão do Struts.

O usuário pode configurar outros interceptores se desejar ou pode criar seus próprios interceptores. Para criar seu próprio interceptor, basta implementar a interface Interceptor do *framework*. Existem também classes que provêm algumas funcionalidades básicas que podem ser estendidas.

Nesse trabalho os interceptores não foram estudados a fundo, foram usados apenas os interceptores disponíveis no *framework*.

1.1.6 Dados de formulários

O Struts possibilita o uso de várias tecnologias para o desenvolvimento das visões. Foi utilizada a tecnologia JSP pelos motivos descritos no inicio deste capítulo.

O Struts provê uma série de tag libs para criação de formulários. A tag <s:form> define um formulário e a ação que será executada quando ele for submetido ao servidor. Os campos de texto podem ser adicionados pela tag `<s:textfield>` e os botões pela tag `<s:submit>`. Os botões podem definir métodos específicos da ação que devem ser executados. Segue um exemplo abaixo.

```
<s:form action="TipoContrato">
      <s:textfield label="ID" name="tipoContrato.id"/>
      <s:textfield label="Descrição" name="tipoContrato.descricao" />

      <s:submit value="limpar"/>
      <s:submit method="salvar" value="salvar"/>
</s:form>
```

Trecho de Código 3 - Tags do Struts

O código acima é renderizado da seguinte forma:

Figura 2 - Renderização das tags do Struts

Caso o objeto tipoContrato na ação que invocou a jsp seja diferente de nulo os campos virão preenchidos automaticamente com os dados "id" e "descricao", declarados no atributo *name* da *tag* `<s:textfield>`.

O *framework* possui também *tags* para interação em coleções de objetos. Aplicações gerenciais muitas vezes precisam acessar dados em coleções. Exemplos comuns são itens de pedidos, itens de contratos ou uma lista de dados guardados em banco.

A *tag* `<s:iterator>` pode varrer objetos do tipo *java.util.Collection* ou *java.util.Iterator*. O exemplo abaixo mostra o uso para listar as entidades numa lista.

```
<table>
<tr>
<td>ID</td>
<td>Descrição</td>
</tr>
<s:iterator value="tipos">
    <tr>
      <td>
      <s:url id="url" action="TipoContrato" method="selecionar">
            <s:param name="idSelecionado" value="%{id}" />
      </s:url>
      <s:a href="%{url}" >
            <s:property value="id" />
```

```
        </s:a>
        </td>

              <td><s:property value="descricao" /> </td>
        <td>
        <s:url id="url" action="TipoContrato" method="remover">
              <s:param name="idSelecionado" value="%{id}" />
        </s:url>
        <s:a href="%{url}" >Excluir</s:a>
        </td>
        </tr>
</s:iterator>
</table>
```

Trecho de Código 4 - Tags do Struts para Interação

No trecho de código acima, foi criada uma tabela para organizar a lista a ser apresentada. Os dados dos itens da lista podem ser acessados diretamente, conforme feito nas *tags* `<s:property>` e `<s:param>` no exemplo acima. No exemplo a lista é carregada chamando o método "getTipos", conforme configurado no atributo "value" da *tag* `<s:iterator>`.

O código apresentado acima é renderizado como apresentado na figura abaixo.

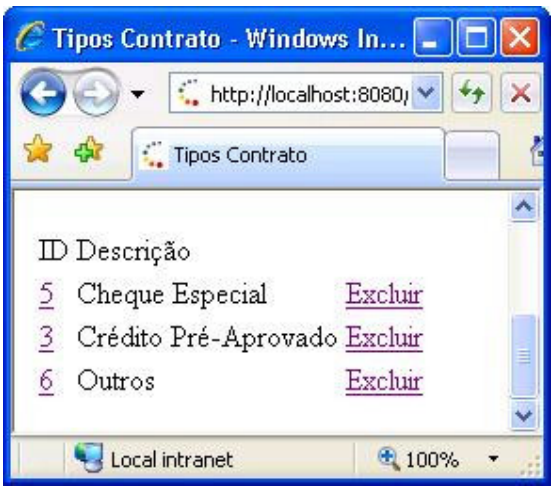

Figura 3 - Renderização das tags de Interação do Struts

1.1.7 Validação

Há duas maneiras de prover validação em uma aplicação feita com Struts2: de maneira programática e de maneira declarativa (ROUGHLEY, 2006, p. 66). A implementação programática consiste em implementar nas ações interfaces específicas do framework. Na maneira declarativa, a validação é feita usando anotações ou arquivos XML associados às ações.

No caso de validação declarativa em XML, deve ser seguido o modelo definido pelo DTD do framework. Todas as validações oferecidas por padrão no XML de validação possuem uma anotação correspondente, que pode ser aplicada diretamente aos atributos da ação, eliminando a necessidade de um arquivo XML adicional.

1.2 Spring Web MVC

O Spring Web MVC Framework é um framework robusto, flexível e bem projetado para o desenvolvimento rápido de aplicações Web usando o padrão de projeto MVC (HEMRAJANI, 2006, p.128). Ele é parte do Spring Framework, que provê diversas soluções para persistência de dados, programação orientada a aspectos, entre outras.

1.2.1 Arquitetura Básica

O Spring MVC é projetado ao redor do DispatcherServlet, que é um Servlet que distribui as requisições para as classes configuradas para trata-las, com visão, localização e temas configuráveis (Spring Reference).

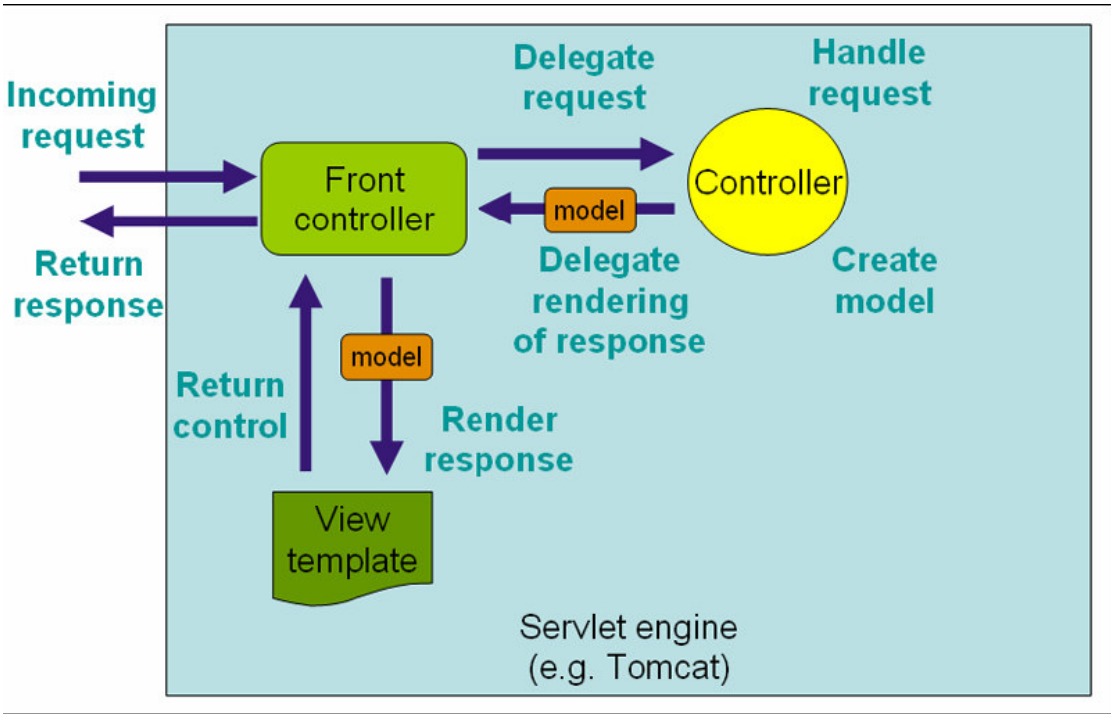

Figura 4 - Arquitetura do Spring MVC (Alto Nível)

Fonte: Spring Reference (2007)

O modelo MVC do Spring é muito similar ao do Struts, apesar de não ser derivado dele. Um *Controller* Spring é semelhante a uma *Action* do Struts, sendo um objeto de serviço multi-thread, com uma única instancia respondendo para todos os clientes (JOHNSON, 2005).

Para Ladd(2006, p. 52), *Controllers* são responsáveis pelo processamento das requisições HTTP, pelo execução das regras de negócio, pela composição dos objetos de resposta e por passar o controle de volta ao fluxo de tratamento principal. O *Controller* não trata a renderização da visão, focando no tratamento de requisições e respostas e delegando ações à camada de serviço.

O framework oferece vários *Controllers* que podem ser estendidos de acordo com a necessidade da aplicação. São *controllers* para tratamentos desde funções simples de processamento e redirecionamento até classes para tratar formulários de dados do usuário e *upload* de arquivos.

Para construir a resposta ao usuário, o método invocado no Controller pode retornar um objeto da classe ModelAndView. O objeto dessa classe mantém tanto a visão que será apresentada ao cliente quanto o modelo usado para construir a visão (JOHNSON, 2005).

1.2.2 Validação

O Spring traz uma interface chamada *Validator*, que pode ser usada para validar objetos. Ela trabalha usando a classe *Errors*, também do *framework*, para receber as mensagens de falhas durante o processo de validação (Spring Reference).

A classe que implementa a interface Validator deve implementar os métodos supports, que deve verifica se o objeto informado pode ser validado por essa classe, e validate, que é responsável pela validação dos campos.

1.3 JBoss Seam

JSF e EJB 3.0 são duas das melhores novidades do Java EE 5. EJB3 é um novo modelo de componentes para lógica de negócio e persistência de dados no lado do servidor. JSF é um excelente modelo de componentes para a camada de apresentação. Infelizmente, nenhum modelo de componentes é

capaz de resolver todos os problemas da computação sozinho. Certamente, JSF e EJB3 trabalham melhor juntos. Mas a especificação Java EE 5 não traz uma maneira padrão para integrar os dois modelos de componentes. Felizmente, os criadores dos dois modelos previram essa situação e proveram pontos de extensão padrão para extensão e integração de outras soluções. O JBoss Seam unifica o modelo de componentes do JSF e do EJB3, eliminando o código de integração, deixando o desenvolvedor pensar na regra de negocio (Seam Reference, 2007).

Segundo Yuan e Heute (2007, p. 5) o Seam foi projetado para aplicações Web com estado (*Stateful*). Aplicações Web são inerentemente multi-usuário e aplicações de comércio eletrônico são inerentemente com estado. Contudo, a maioria dos *frameworks* para aplicações Web são direcionados para aplicações sem estado. Você precisa armazenar os objetos na sessão HTTP para gerenciar o estado do usuário. Isso não apenas complica sua aplicação com código não relacionado ao negócio, mas também traz uma série de problemas de performance.

A figura abaixo mostra como o Seam se encaixa no padrão na arquitetura da especificação Java EE.

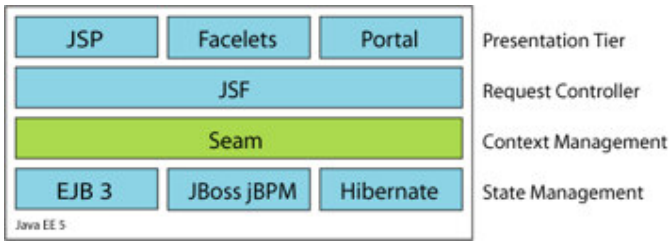

Figura 5 - Seam na arquitetura Java EE

Fonte: Seam Reference (2007)

1.3.1 Arquitetura Básica

O Seam trabalha basicamente com contextos e componentes. Os componentes são criados e associados a um contexto. Os contextos são gerenciados pelo contêiner, de acordo com as transações executadas pelo usuário.

1.3.1.1 Contextos

Os contextos do Seam são: Stateless (sem estado), Event (evento), Page (Página), Conversation, Session, Business Process e Application.

Alguns contextos são semelhantes aos da especificação dos Servlets ou de especificações relacionadas, mas alguns são novos no Seam, como o Conversation e o Business Process (Seam Reference, 2007, p. 52).

Entre os contextos mais utilizados estão o Event e o Conversation. O ciclo de vida do contexto Event dura até o fim da requisição. No caso de uma requisição JSF, o contexto é finalizado após a renderização da página. O contexto Conversation é um dos mais elaborados do framework. Uma conversação é uma unidade de trabalho, um processo de negócio. O estado dos componentes é mantido durante as várias requisições do processo. Várias conversações do mesmo cliente podem ser executadas em paralelo.

Os contextos do Seam se propõem a limitar o uso desnecessário de memória, uma vez que o desenvolvedor não precisa se preocupar com a gerência dos componentes na sessão do usuário. O framework garante, por

exemplo, que os componentes de uma conversação serão excluídos da sessão do usuário se ela for finalizada ou expirar.

1.3.1.2 Componentes

Os componentes do Seam podem ser Beans de Sessão EJB, Beans Dirigidos por Mensagem (*Message Driven Beans*), Beans de Entidade ou simples JavaBeans. Cada componente é armazenado em um Contexto.

Todo componente do Seam tem um nome. Ele é definido pela anotação @Name. Esse nome não está relacionado a nenhum outro nome presente no Java EE, ele é específico para o Seam. Nenhuma das outras anotações do framework funcionará se o componente envolvido não tiver um nome.

O contexto do componente pode ser definido pela anotação @Scope. Caso não seja definido explicitamente o contexto do componente é atribuído de acordo com o tipo de componente. Os Beans de Sessão com Estado, por exemplo, por padrão são associado ao contexto Conversation. Como os Beans de Sessão Sem Estado não são capazes de armazenar estado, eles são sempre armazenados no contexto Stateless.

1.3.1.3 Bibliotecas de Tags

Embora boa parte das JSP possam ser construídas com tags do JSF, o Seam, assim como os outros frameworks estudados, fornece uma série de tags complementares para facilitar o desenvolvimento. Entre as tags fornecidas estão as tags de validação e de criação de links relacionados a ações de componentes.

1.3.2 Validação

Para facilitar a validação das entradas do usuário o Seam usa o hibernate validator. As entidades são anotadas com instruções de validação padrão. O Seam provê tags para serem usadas com JSF que fazem com que as entradas do usuário sejam validadas e que a página seja recarregada em caso de erro de validação.

A validação das entradas do usuário nos *forms* do Seam é feita no lado do servidor. Você deve sempre validar seus dados no lado do servidor porque um usuário mal intencionado pode desabilitar qualquer mecanismo de validação no cliente, como JavaScripts, por exemplo (YUAN e HEUTE, 2007, p. 139).

METODOLOGIA

Para avaliação e levantamento de critérios de comparação entre os frameworks foi desenvolvido um protótipo com cada ferramenta estudada. A camada de negócio da aplicação foi desenvolvida anteriormente e foi usada da mesma forma para todos os frameworks.

O protótipo é uma simplificação de uma aplicação de cálculo de contratos de empréstimo para fins de renegociação. A aplicação foi escolhida por conter alguns cadastros simples, sem regras de negócio complexas, e uma rotina de cálculo, específica da aplicação. Dessa forma foi possível avaliar o desempenho dos frameworks tanto nas tarefas simples, de simplesmente manter uma tabela num banco de dados, quanto para tarefas mais complexas, como uma rotina de cálculo.

1.1 Modelo de entidades

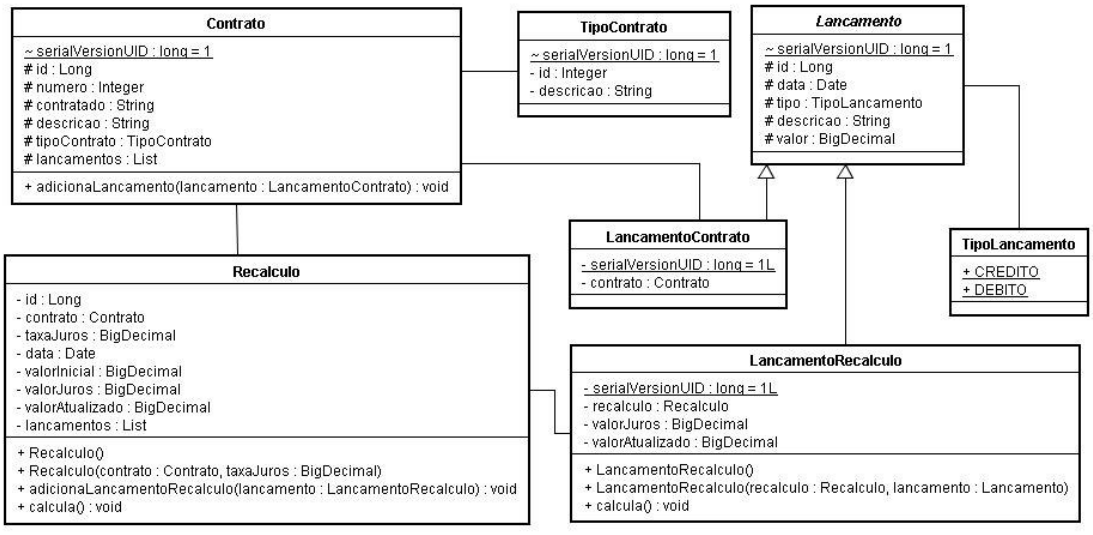

Figura 6 - Diagrama de Classes das Entidades dos Protótipos

A entidade central do modelo é a classe Contrato. Ela possui um tipo de contrato, representado por outra entidade, e uma lista de lançamentos de contrato. Sobre um contrato podem ser efetuados recálculos, que por sua vez tem lançamentos de recalculo. Pela semelhança entre lançamentos de contrato e recálculo, foi criada uma generalização.

Para o protótipo desenvolvido com o framework JBoss Seam foram adicionadas anotações específicas nas entidades. As anotações adicionais não foram utilizadas nos protótipos dos outros frameworks.

1.2 Casos de Uso

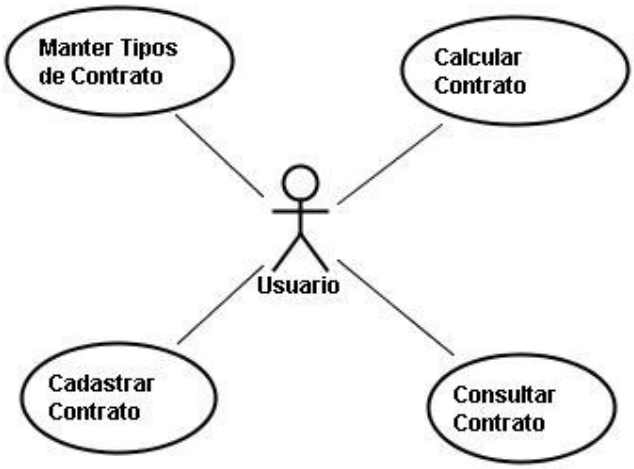

Figura 7 - Casos de Uso dos Protótipos

- Manter Tipos de Contrato

Manutenção da tabela de tipos de contrato que serão utilizados posteriormente no cadastro de contratos. Esse caso de uso foi incluído para

avaliar os *frameworks* em operações simples de manutenção de tabelas em banco de dados.

- Cadastrar Contrato

Cadastro dos contratos e lançamentos do protótipo. Esse caso de uso foi incluído por se tratar de uma inclusão envolvendo várias entidades. Existe um relacionamento muitos para um, entre Contrato e TipoContrato, e um relacionamento um para muitos, entre Contrato e LancamentoContrato.

- Consultar Contrato

Consulta dos contratos cadastrados através de critérios simples, como número e nome do contratado. Esse caso de uso foi incluído por ser uma função comum em aplicações gerenciais e por seu formulário não conter exatamente os mesmo campos das entidades.

- Calcular Contratos

Calcula um contrato e seus lançamentos, gerando um recalculo. O cálculo é feito em cada lançamento usando juro composto. O saldo final do recalculo é a diferença entre o somatório do saldo final dos lançamentos do tipo Crédito e o somatório do saldo final dos lançamentos do tipo Débito.

Esse caso de uso foi incluído por implementar uma regra de negócio e gerar com resultado um novo registro de outra entidade.

1.3 Classes para acesso a dados

Foram criados três EJB três Beans de Sessão sem Estado para fazer buscas e alterações nos bancos de dados. As classes construídas com os *frameworks* acessarão esses métodos para persistir os dados da aplicação.

1.4 Camada Visão e Camada Controle

Foram criadas quatro páginas JSP, uma para a manutenção de tipos de contrato, uma para consulta de contrato, uma para cadastro de contratos e uma para mostrar o resultado de um cálculo. Para cada protótipo foram usadas as bibliotecas de *tags* específicas do *framework*.

Foram criados quatro controladores para tratar os eventos da camada visão. Os controladores e a forma como a camada visão se comunica com eles é específica de cada *framework*.

1.5 Empacotamento e distribuição

Cada protótipo foi empacotado em pacotes ear, contendo os arquivos de configuração do Java EE e os específicos do *framework*.

Os protótipos foram testados no servidor JBoss, por ser o servidor JEE de código aberto mais utilizado no mercado e por já ser de conhecimento do desenvolvedor do trabalho.

CRITÉRIOS

5.1. Integração com EJB3

Foi verificado se a solução é capaz de interagir com a camada de negócio escrita com EJB3, se são necessárias configurações ou bibliotecas adicionais. A camada desenvolvida com o *framework* deve ser capaz de utilizar a implementação prévia das regras de negócio.

Com todos os frameworks estudados a utilização das regras de negócio escritas anteriormente foi bastante simples. Contudo foi possível perceber algumas diferenças.

Para tirar proveito do modelo de componentes do JBoss Seam, foi necessário adicionar anotações em algumas entidades das regras de negócio. Dessa forma, o framework facilitava as operações de inserção.

No JBoss Seam, como para construção dos componentes foram utilizados Beans de Sessão com Estado, não foi necessário acessar diretamente o JNDI para acessar os Beans de Sessão sem Estado das regras de negócio. Foi possível usar a anotação @EJB, que faz com que o contêiner seja responsável pela injeção do bean de sessão.

Para avaliação dos frameworks, foram criadas as categorias **Ótima**, **Boa** e **Ruim**. O JBoss Seam ficou classificado na categoria Ótima integração com EJB3, por oferecer facilidades além das disponíveis pelos outros frameworks. Struts e Spring MVC foram classificados na categoria Boa integração com EJB3, por não apresentarem nenhuma dificuldade em trabalhar com a tecnologia. A categoria Ruim foi mantida para o caso de, em futuras análises,

for observado que em algum framework o uso de EJB3 acarrete dificuldades no desenvolvimento.

5.2. Controle de estados

O controle de estados é um dos desafios da programação Web. Em algumas aplicações é necessário que a ferramenta permita um controle da maneira adaptável a diversos cenários e de maneira simples, para facilitar o desenvolvimento.

O Struts e o Spring MVC trabalham em geral sem estado. Para armazenar informações do usuário entre várias requisições é necessário que a aplicação utilize algum recurso para manter o estado, como por exemplo a sessão http. O Seam foi projetado especialmente para aplicações complexas e com estado, por isso provê o modelo de componentes contextual, que oferece vários níveis (contextos) de manutenção de estado.

Tendo em vista os frameworks estudados, foram criadas as categorias **Com controle de estado próprio** e **Sem controle de estado próprio**. Não existe hierarquia entre as categorias pois o fato de o framework possuir um controle de estados específico pode acarretar um aumento de complexidade que, dependendo da aplicação, pode não ser desejável. Dessa forma, o Spring MVC e o Struts foram classificados na primeira categoria e o JBoss Seam na segunda.

5.3. Curva de aprendizado e Documentação

O custo de difusão do conhecimento na equipe de desenvolvimento depende em parte da complexidade das novas tecnologias a serem adotadas bem como da documentação disponível.

O tempo de aprendizado em um novo framework depende muito do atual conhecimento da equipe. Para esse trabalho, foi avaliada a dificuldade para desenvolvedores que tenham conhecimento intermediário de EJB, JSP e Servlets, bem como conhecimento básico de http, html e java script.

Como documentação foram considerados a documentação oficial, oferecida pelo desenvolvedor do framework, livros publicados na área, exemplos e tutoriais disponíveis no site do desenvolvedor e na Internet.

Foram criados dois critérios separados para documentação e curva de aprendizado, pois, embora relacionados, o segundo depende muito do conhecimento da equipe que irá utilizar o framework. Tratando os itens separadamente, é possível modificar a classificação de acordo com o conhecimento de uma equipe específica.

Foram criadas quatro categorias para classificar a documentação. **Ruim**, **regular**, **boa** e **muito boa**. Para curva de aprendizado foram criadas também quatro categorias: **curta**, **média**, **longa** e **muito longa**.

O Struts, por sua maturidade e popularidade, possui vasta documentação na Internet e vários livros publicados a respeito. Seu manual de referência é bastante completo. Por esses motivos teve sua documentação classificada como **muito boa**.

O Spring vem adquirindo muitos adeptos e há vários livros publicados sobre o framework, contudo, como o framework MVC é apenas uma pequena parte do Spring, ele acaba tendo uma documentação menos detalhada tanto

nos livros quanto na documentação oficial. Sua documentação foi classificada como **boa**.

O JBoss Seam possui um bom manual de referência e alguns livros publicados, porém por ser um framework relativamente novo ainda não possui muitos desenvolvedores e exemplos disponíveis na Internet. Sua documentação foi classificada como **boa**.

A curva de aprendizado dos frameworks Struts e Spring MVC para desenvolvedores Web experientes não deve ser muito longa, pois o tratamento de requisições com pouco controle de estados é comum na programação com JSP e Servlets. Já o JBoss Seam, por seu modelo de componentes com estado, requer mais esforço para compreensão. O Seam também requer conhecimentos no framework JSF.

Pelos motivos acima, a curva de aprendizado dos frameworks Struts e Spring foram classificadas como **médias** e o a do JBoss Seam como **longa**.

5.4. Facilidade de Efetuar Testes Automatizados

Boa parte do desenvolvimento de software nos dias de hoje necessita de testes automatizados. Os artefatos produzidos com o frameworks devem poder ser facilmente testados fora do ambiente de produção.

Para avaliar a facilidade de testar automaticamente o código feito para os frameworks foram criadas três categorias, **baixa**, **alta** e **muito alta**.

Os três frameworks trabalham com POJOs ou beans de sessão, que podem rodar independentes do contêiner e, portanto, podem ser facilmente testados usando JUnit ou outro framework de testes.

Os frameworks Struts e Spring possuem classes para auxiliar o desenvolvimento de testes automatizados e foram classificados na categoria alta. O JBoss Seam vai além, implementando seu próprio framework de testes, baseado no framework testNG, sendo portando enquadrado na categoria muito alta.

5.5. Utilização no mercado

Em alguns casos, a utilização no mercado é muito importante para seleção de um framework. Quanto mais difundida a tecnologia, maior a possibilidade de se encontrar profissionais já experientes e maior a facilidade de encontrar exemplos e discussões na Web. Além disso, soluções bem aceitas pelo mercado costumam ser levadas em conta nas especificações de novos padrões.

O Struts, na sua versão 1, tornou-se extremamente popular. Segundo muitos especialistas e segundo a própria página do produto, é o framework para desenvolvimento Web mais utilizado no mercado. Há uma infinidade de código e discussões a respeito na Web. O Spring MVC, também bastante popular, embora não tanto quanto o Struts. O Seam é o mais recente dos três frameworks estudados e, portanto, menos utilizado até o momento.

Foram criadas três categorias, **muito utilizado**, **pouco utilizado** e **novo**. A categoria novo foi criada, pois embora com pouco uso, um framework novo pode vir a ser mais utilizado em um curto espaço de tempo caso se destaque e, portanto pode ser analisado de maneira diferente. Struts e Spring MVC foram

classificados como muito utilizados, enquanto o JBoss Seam foi classificado como novo.

5.6. Quadro comparativo dos frameworks avaliados

Critério/Frameworks	Struts	Spring MVC	Seam
Integração com EJB3	Boa	Boa	Ótima
Controle de estados	Sem	Sem	Controle Próprio
Documentação	Muito Boa	Boa	Boa
Curva de Aprendizado	Média	Média	Longa
Facilidade de efetuar testes	Alta	Alta	Muito Alta
Utilização no Mercado	Muito Utilizado	Muito Utilizado	Novo

CONCLUSÃO

Há uma grande variedade de frameworks para o desenvolvimento Web em Java, o que torna muito difícil a sua avaliação. O levantamento de critérios auxilia a escolha de um framework para uma determinada situação, pois permite a tabulação das características de cada artefato estudado, facilitando assim a análise.

É difícil levantar critérios objetivos na comparação de tecnologias. Critérios como velocidade de desenvolvimento ou linhas de código necessárias para desenvolver uma aplicação não seriam avaliadas adequadamente apenas com a construção de protótipos.

O estabelecimento de critérios, embora subjetivos, deve auxiliar futuras avaliações de frameworks, permitindo que o analista investigue diretamente a classificação do artefato nos critérios pré-estabelecidos.

A elaboração de um quadro comparativo facilita as análises por sintetizar boa parte do estudo elaborado. Ele pode ser facilmente atualizado, no caso de uma nova versão do framework, ou expandido, no caso da avaliação de um novo framework.

A escolha do melhor framework deve também levar em consideração especificidades das aplicações e da equipe de desenvolvedores da organização. No entanto, o estudo feito deve auxiliar na análise numa situação específica, uma vez que os dados estão tabulados e seu embasamento está contido no trabalho.

REFERÊNCIAS

DEITEL H. M., DEITEL P.J. **Java Como Programar**. Sexta Edição, Prentice Hall. 2004.

FORD, N. **Art of Java Web Development**. Greemwich: Manning, 2003.

FOWLER, M. **Inversion of Control Containers and the Dependency Injection Pattern**, 23 Jan. 2004. Disponível em http://martinfowler.com/articles/injection.html Acesso em: 08 out. 2007.

HEMRAJANI, A. **Agile Java Development With Spring, Hibernate and Eclipse**, SAMS, 2006

Hunter, J. **Java Servlet Programming.** Primeira Edição. Sebastopol: O'Reilly, 1998.

HUSTED, T.; LUPPENS, P. **Nutshell**, Disponível em http://struts.apache.org/2.0.9/docs/nutshell.html Acesso em: 08 out. 2007.

JOHNSON, R. **Introduction to Spring Framework**, Disponível em http://www.theserverside.com/tt/articles/article.tss?l=SpringFramework Acesso em 08 out. 2007.

JOHNSON, R. et al. **Professional Java Development with the Spring Framework**, Indianápois: Wiley, 2005

LADD, S. **Expert Spring MVC and Web Flows**, Primeira Edição, Nova Iorque: Apress, 2006

SILVA, R. P. **Suporte ao desenvolvimento e uso de frameworks e componentes**, Porto Alegre, 2000, 262 p. Tese (Doutorado em Ciência da Computação) – Instituto de Informática, Universidade Federal do Rio Grande do Sul.

ROUGHLEY, I. **Starting Struts2**, C4Media, 2006

Spring Reference Manual, Disponível em http://static.springframework.org/spring/docs/2.1.x/spring-reference.pdf Acesso em: 22 out. 2007.

SRIGANESH, P. et al. **Mastering Enterprise Java Beans 3.0**, Quarta Edição, Indianápolis: Wiley, 2006.

YUAN, M. J.; HEUTE, T. **JBoss Seam Simplicity and Power Beyond Java EE**, Primeira Edição, Upper Saddle River: Prentice Hall, 2007

ANEXO 1 – ARTIGO

ANÁLISE E COMPARAÇÃO DE FRAMEWORKS PARA DESENVOLVIMENTO WEB EM JAVA

Thiago Roberto dos Santos

Departamento de Informática e Estatística, Universidade Federal de Santa Catarina
Campus Trindade, Caixa Postal 476, CEP 88.040-900, Florianópolis-SC, BRAZIL

RESUMO

Este trabalho visa definir critérios para avaliar e comparar tecnologias para a camada de apresentação de aplicações Web desenvolvidas em Java. O ambiente web vem sendo muito utilizado para o desenvolvimento de aplicações. É interessante analisar ferramentas que auxiliem na resolução de problemas na camada de apresentação que, em geral, já foram resolvidos nas aplicações convencionais.

Para realizar o levantamento dos critérios e avaliar algumas soluções disponíveis no mercado, foram construídos protótipos com os frameworks Struts, Spring MVC e JBoss Seam.

Foram definidos critérios aos quais foram submetidos os frameworks avaliados. Foi gerado um quadro comparativo para os frameworks estudados para facilitar a visualização dos resultados.

1- INTRODUÇÃO

Tem se tornado cada vez mais comum a escolha do ambiente Web para o desenvolvimento de aplicações corporativas. Em decorrência disso, a complexidade das aplicações tem crescido muito rapidamente.

A mudança de ambiente trouxe de volta uma série de problemas já resolvidos pelas aplicações tradicionais. Várias soluções já estão disponíveis para resolver tais problemas, mas, devido a grande oferta, é difícil escolher uma solução perene para a organização.

Este trabalho tem como principal objetivo definir critérios para análise e comparação das tecnologias para camada de apresentação mais adequadas para sistemas gerenciais de pequeno e médio porte que utilizem a tecnologia EJB para a camada de negócio e persistência de dados.

2- CONCEITOS FUNDAMENTAIS

Java

Java é uma linguagem de programação orientada a objeto desenvolvida na década de noventa por James Gosling, na SUN Microsystems. Java tem sintaxe similar às linguagens C e C++. Ela é compilada em bytecodes, que podem ser executados por uma máquina virtual ou por hardware específico (Deitel; Deitel, 2004).

Plataforma Java

Plataforma Java é uma plataforma para desenvolvimento e execução de programas escritos na linguagem Java. A plataforma não é específica para um hardware ou sistema operacional. As aplicações da plataforma rodam sobre uma

máquina virtual, que possuí implementações para diversas arquiteturas. A plataforma é distribuída em três edições diferentes:

- Java Standart Edition (Java SE), para máquinas desktop.
- Java Micro Edition (Java ME), para dispositivos portáteis.
- Java Enterprise Edition (Java EE), para aplicações corporativas.

Java EE

Java EE é uma edição da plataforma Java voltada para aplicações corporativas, distribuídas, multi-camadas.

A plataforma Java EE é definida por uma especificação, considerada informalmente um padrão, pois os fornecedores de servidores de aplicação têm que cumprir certos requisitos para declarar seus produtos compatíveis com Java EE. Ela inclui várias especificações de API como JDBC, RMI, e-mail, JMS entre outras.

Servlets

"Um Servlet extende a funcionalidade de um servidor, como um servidor Web que serve páginas da Web para um navegador do usuário com o protocolo Http" (DEITEL; DEITEL, 2004, p. 929).

O servlet trabalha com o conceito de requisição e resposta, através do protocolo Http. Um Servlet recebe uma requisição e, baseado nela, fornece uma resposta. A resposta pode ser um documento HTML, XML, entre outros.

O contêiner WEB é responsável pelo ciclo de vida do Servlet, pelo tratamento das múltiplas chamadas que podem ser feitas pelos clientes e também pela tradução de uma chamada http para uma requisição do Servlet.

Java ServerPages (JSP)

"A tecnologia das Java ServerPages, um extensão da tecnologia dos servlets, simplifica o processo de criação de páginas separando a apresentação do conteúdo. Normalmente, as JSPs são utilizadas quando a maior parte do conteúdo enviado ao cliente é texto estático e quando uma pequena parte do conteúdo é gerada dinamicamente com código Java" (DEITEL; DEITEL, 2004, p. 929).

As JSP combinam o uso de HTML e linguagem Java. Em relação aos Servlets, elas facilitam a divisão de trabalho entre os designers e os programadores, pois permitem a escrita das páginas em HTML intercaladas com código Java.

As JSPs são compiladas para Servlets, passando a responder requisições de maneira semelhante a eles. A compilação pode ser feita no momento da instalação da aplicação no servidor ou no momento em que a JSP é requisitada pela primeira vez.

Bibliotecas de Tags (Tag Libs)

As bibliotecas de tags definem funcionalidades que podem ser usadas em qualquer JSP, em forma semelhante às tags HTML. As tags são processadas no servidor no momento da requisição, podendo gerar código HTML.

As bibliotecas mantém o código mais legível, pois evitam a mistura de código Java com código HTML. As tags costumam ser mais amigáveis também para os designers, que não estão acostumados em ver trechos de código no meio das páginas.

Padrões de projeto

Os padrões de projeto são soluções já testadas para resolver problemas recorrentes. Os padrões são documentados e, a medida que se mostrem eficientes, devem ser reutilizados. Os padrões de projeto precisam ser adaptados ao contexto da aplicação. Eles não se propõem a determinar a maneira como o código deve ser escrito, mas como os elementos estão estruturados.

Padrão MVC

O padrão MVC teve origem no SmallTalk, onde era usado para separar a interface com o usuário dos dados. Dessa forma, mudanças as mudanças de interface causavam menos impacto na camada de dados e vice-versa.

Modelo

O modelo é responsável pelos dados e pelas regras de negócio da aplicação. Ele que interage com os repositórios de dados e executa as operações da aplicação.

Visão

A visão é responsável por apresentar os dados para o usuário. Os mesmos dados podem ser apresentados de diversas formas diferentes, como tabelas, formulários ou gráficos, sem que seja necessário reescrever o código de acesso aos dados para cada forma de apresentação.

Controlador

O controlador é responsável pela comunicação entre as camadas visão e modelo. Ele trata entradas dos usuários, invoca as regras de negócio pertinentes na camada modelo e apresenta os dados através da camada de visão.

Framework

Segundo Silva (2000, p. 21) “Frameworks são estruturas de classes que constituem implementações incompletas que, estendidas, permitem produzir diferentes artefatos de software. A grande vantagem desta abordagem é a produção de reuso de código e projeto, que pode diminuir o tempo e o esforço exigidos na produção de software”.

Injeção de Dependência

Segundo Ladd(2006, p. 11) uma especialização de Inversão de Controle, Injeção de Dependência é uma técnica usadas por frameworks para unir a aplicação. O framework faz o trabalho de conectar as dependências da aplicação, removendo o código específico para criação de objetos da aplicação.

3- FRAMEWORKS AVALIADOS

Neste trabalho foram avaliados os frameworks Struts2, Spring MVC e JBoss Seam. Struts 2 foi escolhido por sua popularidade no mercado. A versão 2 trouxe avanços sobre a anterior, especialmente na simplificação dos arquivos de configuração. Spring MVC foi selecionado pela quantidade de artigos escritos sobre o

framework. JBoss Seam foi selecionado pois o conceito do framework é semelhante ao do Web Beans, que é um modelo de componentes que integra JSF e EJB3 e pode vir a ser incorporado à especificação Java EE.

Struts 2

O Struts é um framework de código aberto desenvolvido pela fundação apache para desenvolvimento de aplicações Web com Java EE que encoraja a programação no modelo MVC(Model-View-Controller). Ele provê um tratador de requisições e de respostas, que auxilia o tratamento das interações entre cliente e servidor. Também é disponibilizado um conjunto de bibliotecas de tags para facilitar a construção de aplicações baseadas em forms HTML.

Arquitetura Básica

As requisições aos recursos são captadas pelos filtros de servlet que de acordo com a requisição determina a Ação (*Action*) a ser executada. Interceptores (*Interceptors*) aplicam funcionalidades comuns como validação, tratamento de upload, e fluxo de trabalho (*workflow*). Após os tratamentos, a Ação é executada e os resultados são renderizados no cliente na forma de HTML, PDF, imagens ou outra saída.

A operação de negócio é executada na Ação invocada pelo *framework* e os interceptores fazem tratamentos prévios da requisição, de acordo com as configurações e com o conteúdo enviado pelo cliente.

Ações

O Struts é um framework baseado em ações, portanto as classes de ação são parte central do framework. As ações podem ser usadas de diferentes formas. Na sua forma mais simples, temos uma classe que implementa o método execute (o método pode ter outro nome, desde que esteja indicado em arquivos de configuração), que retorna um String indicando o resultado da ação

Alguns conjuntos de ações podem ter resultados comuns, podendo ser criados mais de um método na classe de ação. Nesse caso, quando a ação for invocada dever passado como entrada o método que será executado. Caso não seja informado um método o será executado o método execute.

Interceptores

Interceptores funcionam de maneira semelhante aos filtros dos Servlets, podendo assim ser ordenados e acessar dados da atual requisição. Para a seleção da ação a ser executada, extração dos parâmetros de entrada e tratamento de exceção, por exemplo, são utilizados interceptores. Alguns interceptores são pré-configurados pela configuração padrão do Struts.

O usuário pode configurar outros interceptores se desejar ou pode criar seus próprios interceptores. Para criar seu próprio interceptor, basta implementar a interface Interceptor do *framework*. Existem também classes que provêm algumas funcionalidades básicas que podem ser estendidas.

Validação

Há duas maneiras de prover validação em uma aplicação feita com Struts2: de maneira programática e de maneira declarativa (ROUGHLEY, 2006, p. 66). A implementação programática consiste em implementar nas ações interfaces

específicas do framework. Na maneira declarativa, a validação é feita usando anotações ou arquivos XML associados às ações.

No caso de validação declarativa em XML, deve ser seguido o modelo definido pelo DTD do framework. Todas as validações oferecidas por padrão no XML de validação possuem uma anotação correspondente, que pode ser aplicada diretamente aos atributos da ação, eliminando a necessidade de um arquivo XML adicional.

Spring Web MVC

O Spring Web MVC Framework é um framework robusto, flexível e bem projetado para o desenvolvimento rápido de aplicações Web usando o padrão de projeto MVC (HEMRAJANI, 2006, p.128). Ele é parte do Spring Framework, que provê diversas soluções para persistência de dados, programação orientada a aspectos, entre outras.

Arquitetura Básica

O Spring MVC é projetado ao redor do DispatcherServlet, que é um Servlet que distribui as requisições para as classes configuradas para trata-las, com visão, localização e temas configuráveis (Spring Reference).

O modelo MVC do Spring é muito similar ao do Struts, apesar de não ser derivado dele. Um *Controller* Spring é semelhante a uma *Action* do Struts, sendo um objeto de serviço multi-thread, com uma única instancia respondendo para todos os clientes (JOHNSON, 2005).

Para Ladd(2006, p. 52), *Controllers* são responsáveis pelo processamento das requisições HTTP, pelo execução das regras de negócio, pela composição dos objetos de resposta e por passar o controle de volta ao fluxo de tratamento principal. O *Controller* não trata a rederização da visão, focando no tratamento de requisições e respostas e delegando ações à camada de serviço.

O framework oferece vários *Controllers* que podem ser estendidos de acordo com a necessidade da aplicação. São *controllers* para tratamentos desde funções simples de processamento e redirecionamento até classes para tratar formulários de dados do usuário e *upload* de arquivos.

Para construir a resposta ao usuário, o método invocado no Controller pode retornar um objeto da classe ModelAndView. O objeto dessa classe mantém tanto a visão que será apresentada ao cliente quanto o modelo usado para construir a visão (JOHNSON, 2005).

Validação

O Spring traz uma interface chamada *Validator*, que pode ser usada para validar objetos. Ela trabalha usando a classe *Errors*, também do *framework*, para receber as mensagens de falhas durante o processo de validação (Spring Reference).

A classe que implementa a interface Validator deve implementar os métodos supports, que deve verifica se o objeto informado pode ser validado por essa classe, e validate, que é responsável pela validação dos campos.

JBoss Seam

JSF e EJB 3.0 são duas das melhores novidades do Java EE 5. EJB3 é um novo modelo de componentes para lógica de negócio e persistência de dados no lado do servidor. JSF é um excelente modelo de componentes para a camada de apresentação. Infelizmente, nenhum dos modelo de componentes é capaz de resolver todos os problemas da computação sozinho. Certamente, JSF e EJB3 trabalham melhor juntos. Mas a especificação Java EE 5 não traz uma maneira padrão para

integrar os dois modelos de componentes. Felizmente, os criadores dos dois modelos previram essa situação e proveram pontos de extensão padrão para extensão e integração de outras soluções. O JBoss Seam unifica o modelo de componentes do JSF e do EJB3, eliminando o código de integração, deixando o desenvolvedor pensar na regra de negocio (Seam Reference, 2007).

Segundo Yuan e Heute (2007, p. 5) o Seam foi projetado para aplicações Web com estado (*Statefull*). Aplicações Web são inerentemente multi-usuário e aplicações de comércio eletrônico são inerentemente com estado. Contudo, a maioria dos *frameworks* para aplicações Web são direcionados para aplicações sem estado. Você precisa armazenar os objetos na sessão HTTP para gerenciar o estado do usuário. Isso não apenas complica sua aplicação com código não relacionado ao negócio, mas também traz uma série de problemas de performance.

Arquitetura Básica

O Seam trabalha basicamente com contextos e componentes. Os componentes são criados e associados a um contexto. Os contextos são gerenciados pelo conteiner, de acordo com as transações executadas pelo usuário.

Contextos

Os contextos do Seam são: Stateless (sem estado), Event (evento), Page (Página), Conversation, Session, Business Process e Application.

Alguns contextos são semelhantes aos da especificação dos Servlets ou de especificações relacionadas, mas alguns são novos no Seam, como o Conversation e o Business Process (Seam Reference, 2007, p. 52).

Entre os contextos mais utilizados estão o Event e o Conversation. O ciclo de vida do contexto Event dura até o fim da requisição. No caso de uma requisição JSF, o contexto é finalizado após a renderização da página. O contexto Conversation é um dos mais elaborados do framework. Uma conversação é uma unidade de trabalho, um processo de negócio. O estado dos componentes é mantido durante as várias requisições do processo. Várias conversações do mesmo cliente podem ser executadas em paralelo.

Os contextos do Seam se propõem a limitar o uso desnecessário de memória, uma vez que o desenvolvedor não precisa se preocupar com a gerência dos componentes na sessão do usuário. O framework garante, por exemplo, que os componentes de uma conversação serão excluídos da sessão do usuário se ela for finalizada ou expirar.

Componentes

Os componentes do Seam podem ser Beans de Sessão EJB, Beans Dirigidos por Mensagem (*Message Driven Beans*), Beans de Entidade ou simples JavaBeans. Cada componente é armazenado em um Contexto.

Todo componente do Seam tem um nome. Ele é definido pela anotação @Name. Esse nome não está relacionado a nenhum outro nome presente no Java EE, ele é específico para o Seam. Nenhuma das outras anotações do framework funcionará se o componente envolvido não tiver um nome.

O contexto do componente pode ser definido pela anotação @Scope. Caso não seja definido explicitamente o contexto do componente é atribuído de acordo com o tipo de componente. Os Beans de Sessão com Estado, por exemplo, por padrão são associado ao contexto Conversation. Como os Beans de Sessão Sem Estado não são capazes de armazenar estado, eles são sempre armazenados no contexto Stateless.

Validação

Para facilitar a validação das entradas do usuário o Seam usa o hibernate validator. As entidades são anotadas com instruções de validação padrão. O Seam provê tags para serem usadas com JSF que fazem com que as entradas do usuário sejam validadas e que a página seja recarregada em caso de erro de validação.

A validação das entradas do usuário nos forms do Seam é feita no lado do servidor. Você deve sempre validar seus dados no lado do servidor porque um usuário mal intencionado pode desabilitar qualquer mecanismo de validação no cliente, como JavaScripts, por exemplo (YUAN e HEUTE, 2007, p. 139).

4- METODOLOGIA

Para avaliação e levantamento de critérios de comparação entre os frameworks foi desenvolvido um protótipo com cada ferramenta estudada. A camada de negócio da aplicação foi desenvolvida anteriormente e foi usada da mesma forma para todos os frameworks.

O protótipo é uma simplificação de uma aplicação de cálculo de contratos de empréstimo para fins de renegociação. A aplicação foi escolhida por conter alguns cadastros simples, sem regras de negócio complexas, e uma rotina de cálculo, específica da aplicação. Dessa forma foi possível avaliar o desempenho dos frameworks tanto nas tarefas simples, de simplesmente manter uma tabela num banco de dados, quanto para tarefas mais complexas, como uma rotina de cálculo.

Casos de Uso

- Manter Tipos de Contrato

Manutenção da tabela de tipos de contrato que serão utilizados posteriormente no cadastro de contratos. Esse caso de uso foi incluído para avaliar os *frameworks* em operações simples de manutenção de tabelas em banco de dados.

- Cadastrar Contrato

Cadastro dos contratos e lançamentos do protótipo. Esse caso de uso foi incluído por se tratar de uma inclusão envolvendo várias entidades. Existe um relacionamento muitos para um, entre Contrato e TipoContrato, e um relacionamento um para muitos, entre Contrato e LancamentoContrato.

- Consultar Contrato

Consulta dos contratos cadastrados através de critérios simples, como número e nome do contratado. Esse caso de uso foi incluído por ser uma função comum em aplicações gerenciais e por seu formulário não conter exatamente os mesmo campos das entidades.

- Calcular Contratos

Calcula um contrato e seus lançamentos, gerando um recalculo. O cálculo é feito em cada lançamento usando juro composto. O saldo final do recalculo é a diferença

entre o somatório do saldo final dos lançamentos do tipo Crédito e o somatório do saldo final dos lançamentos do tipo Débito.

Esse caso de uso foi incluído por implementar uma regra de negócio e gerar com resultado um novo registro de outra entidade.

Camada Visão e Camada Controle

Foram criadas quatro páginas JSP, uma para a manutenção de tipos de contrato, uma para consulta de contrato, uma para cadastro de contratos e uma para mostrar o resultado de um cálculo. Para cada protótipo foram usadas as bibliotecas de *tags* específicas do *framework*.

Foram criados quatro controladores para tratar os eventos da camada visão. Os controladores e a forma como a camada visão se comunica com eles é específica de cada *framework*.

Empacotamento e distribuição

Cada protótipo foi empacotado em pacotes ear, contendo os arquivos de configuração do Java EE e os específicos do *framework*.

Os protótipos foram testados no servidor JBoss, por ser o servidor JEE de código aberto mais utilizado no mercado e por já ser de conhecimento do desenvolvedor do trabalho.

5- CRITÉRIOS

Integração com EJB3

Foi verificado se a solução é capaz de interagir com a camada de negócio escrita com EJB3, se são necessárias configurações ou bibliotecas adicionais. A camada desenvolvida com o *framework* deve ser capaz de utilizar uma implementação prévia das regras de negócio.

Com todos os frameworks estudados a utilização das regras de negócio escritas anteriormente foi bastante simples. Contudo foi possível perceber algumas diferenças.

Para tirar proveito do modelo de componentes do JBoss Seam, foi necessário adicionar anotações em algumas entidades das regras de negócio. Dessa forma, o framework facilitava as operações de inserção.

No JBoss Seam, como para construção dos componentes foram utilizados Beans de Sessao com Estado, não foi necessário acessar diretamente o JNDI para acessar os Beans de Sessão sem Estado das regras de negócio. Foi possível usar a anotação @EJB, que faz com que o conteiner seja responsável pela injeção do bean de sessão.

Para avaliação dos frameworks, foram criadas as categorias **Ótima**, **Boa** e **Ruim**. O JBoss Seam ficou classificado na categoria Ótima integração com EJB3, por oferecer facilidades além das disponíveis pelos outros frameworks. Struts e Spring MVC foram classificados na categoria Boa integração com EJB3, por não apresentarem nenhuma dificuldade em trabalhar com a tecnologia. A categoria Ruim foi mantida para o caso de, em futuras análises, for observado que em algum framework o uso de EJB3 acarrete dificuldades no desenvolvimento.

Controle de estados

O controle de estados é um dos desafios da programação Web. Em algumas aplicações é necessário que a ferramenta permita um controle da maneira adaptável a diversos cenários e de maneira simples, para facilitar o desenvolvimento.

O Struts e o Spring MVC trabalham em geral sem estado. Para armazenar informações do usuário entre várias requisições é necessário que a aplicação utilize algum recurso para manter o estado, como por exemplo a sessão http. O Seam foi projetado especialmente para aplicações complexas e com estado, por isso provê o modelo de componentes contextual, que oferece vários níveis (contextos) de manutenção de estado.

Tendo em vista os frameworks estudados, foram criadas as categorias **Com controle de estado próprio** e **Sem controle de estado próprio**. Não existe hierarquia entre as categorias pois o fato de o framework possuir um controle de estados específico pode acarretar um aumento de complexidade que, dependendo da aplicação, pode não ser desejável. Dessa forma, o Spring MVC e o Struts foram classificados na primeira categoria e o JBoss Seam na segunda.

Curva de aprendizado e Documentação

O custo de difusão do conhecimento na equipe de desenvolvimento depende em parte da complexidade das novas tecnologias a serem adotadas bem como da documentação disponível.

O tempo de aprendizado em um novo framework depende muito do atual conhecimento da equipe. Para esse trabalho, foi avaliada a dificuldade para desenvolvedores que tenham conhecimento intermediário de EJB, JSP e Servlets, bem como conhecimento básico de http, html e java script.

Como documentação foram considerados a documentação oficial, oferecida pelo desenvolvedor do framework, livros publicados na área, exemplos e tutoriais disponíveis no site do desenvolvedor e na Internet.

Foram criados dois critérios separados para documentação e curva de aprendizado, pois, embora relacionados, o segundo depende muito do conhecimento da equipe que irá utilizar o framework. Tratando os itens separadamente, é possível modificar a classificação de acordo com o conhecimento de uma equipe específica.

Foram criadas quatro categorias para classificar a documentação. **Ruim**, **regular**, **boa** e **muito boa**. Para curva de aprendizado foram criadas também quatro categorias: **curta**, **média**, **longa** e **muito longa**.

O Struts, por sua maturidade e popularidade, possui vasta documentação na Internet e vários livros publicados a respeito. Seu manual de referência é bastante completo. Por esses motivos teve sua documentação classificada como **muito boa**.

O Spring vem adquirindo muitos adeptos e há vários livros publicados sobre o framework, contudo, como o framework MVC é apenas uma pequena parte do Spring, ele acaba tendo uma documentação menos detalhada tanto nos livros quanto na documentação oficial. Sua documentação foi classificada como **boa**.

O JBoss Seam possui um bom manual de referência e alguns livros publicados, porém por ser um framework relativamente novo ainda não possui muitos desenvolvedores e exemplos disponíveis na Internet. Sua documentação foi classificada como **boa**.

A curva de aprendizado dos frameworks Struts e Spring MVC para desenvolvedores Web experientes não deve ser muito longa, pois o tratamento de requisições com pouco controle de estados é comum na programação com JSP e Servlets. Já o JBoss Seam, por seu modelo de componentes com estado, requer mais esforço para compreensão. O Seam também requer conhecimentos no framework JSF.

Pelos motivos acima, a curva de aprendizado dos frameworks Struts e Spring foram classificadas como **médias** e o a do JBoss Seam como **longa**.

Facilidade de Efetuar Testes Automatizados

Boa parte do desenvolvimento de software nos dias de hoje necessita de testes automatizados. Os artefatos produzidos com o frameworks devem poder ser facilmente testados fora do ambiente de produção.

Para avaliar a facilidade de testar automaticamente o código feito para os frameworks foram criadas três categorias, **baixa**, **alta** e **muito alta**.

Os três frameworks trabalham com POJOs ou beans de sessão, que podem rodar independentes do contêiner e portanto, podem ser facilmente testados usando JUnit ou outro framework de testes.

Os frameworks Struts e Spring possuem classes para auxiliar o desenvolvimento de testes automatizados e foram classificados na categoria alta. O JBoss Seam vai além, implementando seu próprio framework de testes, baseado no framework testNG, sendo portando enquadrado na categoria muito alta.

Utilização no mercado

Em alguns casos, a utilização no mercado é muito importante para seleção de um framework. Quanto mais difundida a tecnologia, maior a possibilidade de se encontrar profissionais já experientes e maior a facilidade de encontrar exemplos e discussões na Web. Além disso, soluções bem aceitas pelo mercado costumam ser levadas em conta nas especificações de novos padrões.

O Struts, na sua versão 1, tornou-se extremamente popular. Segundo muitos especialistas e segundo a própria página do produto, é o framework para desenvolvimento Web mais utilizado no mercado. Há uma infinidade de código e discussões a respeito na Web. O Spring MVC, também bastante popular, embora não tanto quanto o Struts. O Seam é o mais recente dos três frameworks estudados e, portanto, menos utilizado até o momento.

Foram criadas três categorias, **muito utilizado**, **pouco utilizado** e **novo**. A categoria novo foi criada, pois embora com pouco uso, um framework novo pode vir a ser mais utilizado em um curto espaço de tempo caso se destaque e portanto pode ser analisado de maneira diferente. Struts e Spring MVC foram classificados como muito utilizados, enquanto o JBoss Seam foi classificado como novo.

Quadro comparativo dos frameworks avaliados

Critério/Frameworks	Struts	Spring MVC	Seam
Integração com EJB3	Boa	Boa	Ótima
Controle de estados	Sem	Sem	Controle Próprio
Documentação	Muito Boa	Boa	Boa
Curva de Aprendizado	Média	Média	Longa
Facilidade de efetuar testes	Alta	Alta	Muito Alta
Utilização no Mercado	Muito Utilizado	Muito Utilizado	Novo

6- CONCLUSÃO

Há uma grande variedade de frameworks para o desenvolvimento Web em Java, o que torna muito difícil a sua avaliação. O levantamento de critérios auxilia a escolha de um framework para uma determinada situação, pois permite a tabulação das características de cada artefato estudado, facilitando assim a análise.

O estabelecimento de critérios, embora subjetivos, deve auxiliar futuras avaliações de frameworks, permitindo que o analista investigue diretamente a classificação do artefato nos critérios pré-estabelecidos.

A elaboração de um quadro comparativo facilita as análises por sintetizar boa parte do estudo elaborado. Ele pode ser facilmente atualizado, no caso de uma nova versão do framework, ou expandido, no caso da avaliação de um novo framework.

A escolha do melhor framework deve também levar em consideração especificidades das aplicações e da equipe de desenvolvedores da organização. No entanto, o estudo feito deve auxiliar na análise numa situação específica, uma vez que os dados estão tabulados e seu embasamento está contido no trabalho.

7- REFERÊNCIAS

DEITEL H. M., DEITEL P.J. **Java Como Programar**. Sexta Edição, Prentice Hall. 2004.

FORD, N. **Art of Java Web Development**.Greemwich: Manning, 2003.

FOWLER, M. **Inversion of Control Containers and the Dependency Injection Pattern**, 23 jan. 2004. Disponível em http://martinfowler.com/articles/injection.html Acesso em: 08 out. 2007.

HEMRAJANI, A. **Agile Java Development With Spring, Hibernate and Eclipse**, SAMS, 2006

Hunter, J. **Java Servlet Programming.** Primeira Edição. Sebastopol: O'Reilly, 1998.

HUSTED, T.; LUPPENS, P. **Nutshell**, Disponível em http://struts.apache.org/2.0.9/docs/nutshell.html Acesso em: 08 out. 2007.

JOHNSON, R. **Introduction to Spring Framework**, Disponível em http://www.theserverside.com/tt/articles/article.tss?l=SpringFramework Acesso em 08 out. 2007.

JOHNSON, R. et al. **Professional Java Development with the Spring Framework**, Indianápois: Wiley, 2005

LADD, S. **Expert Spring MVC and Web Flows**, Primeira Edição, Nova Iorque: Apress, 2006

SILVA, R. P. **Suporte ao desenvolvimento e uso de frameworks e componentes**, Porto Alegre, 2000, 262 p. Tese (Doutorado em Ciência da Computação) – Instituto de Informática, Universidade Federal do Rio Grande do Sul.

ROUGHLEY, I. **Starting Struts2**, C4Media, 2006

Spring Reference Manual, Disponível em http://static.springframework.org/spring/docs/2.1.x/spring-reference.pdf Acesso em: 22 out. 2007.

SRIGANESH, P. et al. **Mastering Enterprise Java Beans 3.0**, Quarta Edição, Indianápolis: Wiley, 2006.

YUAN, M. J.; HEUTE, T. **JBoss Seam Simplicity and Power Beyond Java EE**, Primeira Edição, Upper Saddle River: Prentice Hall, 2007