4
A Camada de Apresentação

O objetivo da camada de apresentação em uma aplicação multicamadas é de expor a lógica de negócios ao usuário e possibilitar a interação do usuário com a aplicação. Em aplicações baseadas na internet como o AulaNet, esta camada também costuma ser chamada de camada web.

Neste capítulo são apresentadas vantagens e desafios decorrentes do uso de interfaces HTML. Logo depois são vistos os benefícios obtidos com a adoção do padrão de arquitetura Model View Controller (MVC) (Fowler, 2002). São apresentados e comparados frameworks de infra-estrutura que implementam o MVC e fornecem recursos para contornar os desafios citados, concluindo a seção com a escolha de um deles para o AulaNet 3.0.

4.1.
Vantagens e Desafios no Desenvolvimento de Interfaces HTML

HTML é a linguagem usada no desenvolvimento da interface com o usuário da maior parte das aplicações baseadas em web atualmente. O uso de interfaces HTML traz vantagens. Segundo Johnson (2002) e (2004) estas a seguir são as principais:

V1. A aplicação é instalada no servidor. O usuário usa o navegador instalado em seu computador que funciona como um cliente universal;

V2. Desacopla o usuário da tecnologia usada no servidor;

V3. Geralmente firewalls são configurados para liberar o trafego de rede na porta utilizada pelo servidor de páginas HTML, diminuindo o esforço de configuração;

V4. Impõem restrições de configurações de hardware mais modestas às máquinas dos usuários já que a maior parte do processamento ocorre no servidor.

Contudo, o desenvolvimento de aplicações baseadas na internet com HTML/HTTP impõe uma série de desafios:

D1. A interface de aplicações web muda freqüentemente sem que necessariamente a lógica de negócios seja alterada. É comum, por exemplo, que um cliente solicite a EduWeb mudanças no visual do AulaNet para que ele se identifique com sua marca;

D2. Uma interface HTML é limitada pelo modelo de requisição e resposta, diferentemente de outras tecnologias de construção de interface como o Swing (Topley, 1999), usada comumente em aplicações Java Desktop, onde um componente de interface pode ser atualizado automaticamente quando ocorre alguma mudança no modelo;

D3. Interfaces web costumam apresentar marcações complexas como tabelas aninhadas e extensos trechos de código JavaScript sendo que apenas uma pequena parte desta marcação se destina a conteúdo gerado dinamicamente. O código de marcação que constrói o layout da interface deve ser separado do código que efetua as operações de negócio de forma a possibilitar a separação em papéis: desenvolvedor e web designer;

D4. Requisições HTTP carregam apenas parâmetros do tipo String, que freqüentemente precisam ser convertidos para outros tipos;

D5. Interfaces HTML tornam a validação da entrada do usuário mais importante, pois a aplicação tem controle limitado sobre o navegador onde o usuário insere os dados;

D6. HTML oferece um conjunto limitado e não expansível de componentes de interface;

D7. Garantir que uma aplicação tenha o mesmo visual em todos os navegadores é custoso, devido as variações na aderência aos padrões HTML, JavaScript e CSS;

D8. Questões de desempenho e concorrência devem ser consideradas, pois é impossível prever o número de usuários acessando uma aplicação web simultaneamente;

D9. Interfaces web são executadas no navegador, ao qual se tem controle limitado. Por conseqüência, são difíceis de testar e depurar.

A divisão da aplicação em camadas, com uma delas destinada a apresentação e interface com o usuário, contribui para a solução de alguns destes desafios. Como uma camada só depende de outra imediatamente inferior, a camada de apresentação pode ser alterada sem que a de negócios sofra alterações (D1). Além disso, o web designer pode concentrar-se na camada de apresentação enquanto o programador concentra-se na camada de negócios (D3). Finalmente, a lógica de negócios não depende da camada web e cada camada pode ser testada, depurada e otimizada isoladamente (D8 e D9).

Contudo, apenas a divisão em camadas não basta para solucionar estes desafios. Para que isto ocorra é preciso também que a camada de apresentação seja limpa, isto é, o fluxo de controle e a chamada dos métodos de negócios são separados da visão, e magra, ou seja, não deve possuir mais código do que o necessário para chamar a camada de negócios. Para garantir que seja limpa, usa-se o padrão de arquitetura Model View Controller (MVC), discutido na próxima seção. Para garantir que seja magra é importante disciplinar os desenvolvedores e realizar auditorias periódicas no código.

4.2. Model View Controller

O padrão de arquitetura Model View Controller (MVC) surgiu pela primeira vez na década de 70 em um framework desenvolvido por Trygve Reenskaug para a plataforma Smalltalk (Lalonde, 1994) e desde então tem influenciado a maior parte dos frameworks voltados para interface com o usuário e a maneira de pensar o design de interfaces (Fowler, 2002).

O MVC divide os elementos da camada de apresentação em três tipos: o controlador (*controller*), encarregado de receber a entrada do usuário, acessar a camada de negócios para manipular o modelo e selecionar a visão; o modelo (*model*), um objeto representando uma parte do domínio e que provê os dados que a visão vai exibir. É o contrato entre o controlador e a visão; a visão (*view*), encarregada de exibir os dados do modelo para o usuário.

A Figura 4.1 mostra o fluxo do MVC clássico, usado em aplicações desktop. Este modelo é adotado em aplicações web com poucas modificações, com é visto mais adiante.

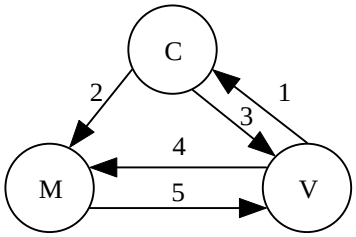

Figura 4.1 – MVC clássico (Ramachandran, 2002)

1) A visão notifica o controlador. Ocorre quando um formulário em uma aplicação web é submetido ou quando um botão é clicado em uma aplicação desktop, por exemplo.
2) O controlador acessa a lógica de negócios e atualiza o modelo. Corresponde a chamada de um método do *Façade* na arquitetura do AulaNet 3.0.
3) O controlador seleciona a visão mais adequada, passando para ela o estado do modelo que ela deve exibir. O controlador pode, por exemplo, selecionar uma visão caso a operação tenha sido bem sucedida ou outra visão em caso de erro.
4) A visão consulta o estado do modelo e exibe as informações ao usuário. Por exemplo, a visão exibe as mensagens postadas em uma conferência.
5) O estado do modelo é alterado. O modelo dispara um evento e a visão se atualiza. Por exemplo, quando um aprendiz envia uma mensagem em um debate. Os outros aprendizes têm sua tela atualizada sem que seja necessário solicitar atualizações ao servidor.

Em aplicações para a internet com interface HTML, como o AulaNet, o modelo de requisição e resposta (Desafio 2) é um impedimento para que o modelo atualize a visão. Neste caso, usa-se o MVC modificado que costuma ser chamado de MVC Web ou de MVC *pull*, pois a visão “puxa” informações do modelo

enquanto que no MVC clássico, ou MVC *push*, o modelo "empurra" atualizações para a visão. A Figura 4.2 exibe o fluxo no MVC *pull*.

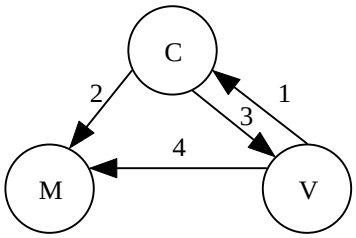

Figura 4.2 – MVC *pull* (Johnson, 2002, 2004)

1) A visão notifica o controlador.
2) O controlador acessa a lógica de negócios e atualiza o modelo.
3) O controlador seleciona a visão mais adequada, passando a ela o modelo que ela irá exibir.
4) A visão consulta o modelo e exibe as informações ao usuário.

A única diferença entre os MVC *push* e *pull* é que no segundo o modelo não atualiza a visão.

Na plataforma J2EE, o modelo costuma ser implementado com JavaBeans, a visão com páginas JSP e o controlador com servlets. Embora seja possível implementar uma solução ad hoc que implante o MVC, há vários frameworks que trazem uma estrutura MVC e tratam alguns dos desafios listados na seção anterior, dentre eles, são analisados nas seções a seguir três destes frameworks: JavaServer Faces (JSF, 2005), Struts (2005) e Spring MVC, o módulo MVC do framework Spring (2005) apresentado no capítulo anterior. Estes frameworks, comumente chamados de web frameworks, possuem diversas características em comum, descritas a seguir.

4.3. Características Comuns a Web Frameworks

Há cerca de 54 frameworks para o desenvolvimento web em plataforma Java gratuitos disponíveis atualmente (Manageability, 2005). Este número é ainda maior se contabilizados os frameworks comerciais. Boa parte destes frameworks implementam o MVC de forma similar e têm características comuns que visam

solucionar alguns dos desafios descritos na seção 4.1. Estas características assim como o comportamento destes frameworks são descritos a seguir.

O framework usualmente traz o controlador do MVC pronto. Este é implementado através de um servlet que deve ser configurado ao instanciar o framework. O controlador mapeia requisições em classes da instância que realizam o padrão de projeto comando (*Command*) (Gamma et al., 1995). A visão é programada pela instância, mas o framework fornece componentes que auxiliam a sua construção. Já o modelo é implementado inteiramente pela instância e não deve possuir nenhum tipo de dependência com o framework escolhido, já que ele também é usado em outras camadas.

Ao receber uma requisição o controlador assume o controle da aplicação. Cabe a ele extrair os parâmetros da requisição, que são do tipo String, convertê-los para tipos mais específicos da aplicação (desafio 4), que podem ser tipos primitivos mais restritos como inteiros e boleanos ou objetos do modelo, e validá-los (desafio 5). Em caso de erro de validação, o controlador exibe novamente a visão que causou o erro. Componentes fornecidos pelo framework são usados na visão, para repovoar os campos preenchidos e exibir mensagens sobre os erros ocorridos.

Quando não ocorre erro de validação, o controlador chama o comando que aciona a camada de negócios para modificar o modelo. Terminada a operação, o comando sinaliza para o controlador, através do valor de retorno, por exemplo, qual visão deve ser exibida e este finaliza a requisição mostrando esta visão.

Para que o controlador extraia os dados da requisição e transforme-os em tipos específicos da instância, o framework fornece um mecanismo que mapeia os campos de um formulário a propriedades destes tipos.

Este é o comportamento geral da maioria dos web frameworks disponíveis atualmente. Cada um deles implementa estas características a sua maneira, além de oferecer recursos extras.

4.4. Comparação Técnica entre Web Frameworks

Com tantos web frameworks disponíveis no mercado, uma solução ad hoc para a camada de apresentação foi descartada. Faz-se necessário escolher 1 dentre

os 54 web frameworks disponíveis. Para poder escolher um deles, é preciso analisá-los e compará-los. Devido ao grande número de opções, sentiu-se a necessidade de realizar uma comparação mais aprofundada do que as comparações realizadas no capítulo 3. Contudo, analisar todos estes web frameworks seria excessivamente custoso e fugiria do escopo deste trabalho. Em vez disto, foram selecionados três: Struts, Spring MVC e JSF.

Struts foi escolhido para ser analisado, pois como será visto em mais detalhes na seção 4.5, é o mais popular. Spring MVC foi escolhido porque o Spring já é usado para a camada de negócios. Usá-lo também para a camada de apresentação reduziria o esforço de aprendizado e possibilitaria uma melhor integração entre as camadas de negócios e apresentação. Por fim, JSF foi escolhido porque é um padrão da Sun Microsystems e define um modelo de componentes. Estes três frameworks podem ser usados com o Spring e o Hibernate, de forma que não há problemas decorrentes da integração. Nesta seção, estes três web frameworks são apresentados seguidos de uma comparação técnica entre eles.

4.4.1. Struts

Por ter sido um dos primeiros web frameworks desenvolvidos, o Struts (2005) também é um dos mais populares entre os desenvolvedores. Contudo, o surgimento de tantos outros é um indicativo de que o Struts não satisfez completamente as necessidades das aplicações. Nesta seção é visto como o Struts implementa as funcionalidades comuns aos web frameworks.

Comandos no Struts são chamados de Ações (*Actions*), que devem estender a classe org.apache.struts.action.Action ou uma de suas subclasses. Ações podem vir acompanhadas de *Action Forms* que são objetos que representam os dados de um formulário e que devem estender a classe org.apache.struts.action.ActionForm ou uma de suas subclasses.

O mapeamento entre requisições e ações é feito através da URL da requisição. A configuração da URL que chama o servlet controlador do Struts, inserida no descritor web.xml da instância, deve conter uma parte fixa, que serve

para identificar o controlador do Struts, e uma parte variável, que identifica qual ação será executada.

A Listagem 4.1 exibe um fragmento do descritor da aplicação que registra o controlador do Struts. O mapeamento realizado na linha 10, "*.do" indica que a parte fixa é ".do" e a parte variável é o que quer que venha antes.

```
01: <web-app>
02:    <servlet>
03:       <servlet-name>Struts Controller</servlet-name>
04:       <servlet-class>
05:          org.apache.struts.action.ActionServlet
06:       </servlet-class>
07:    </servlet>
08:    <servlet-mapping>
09:       <servlet-name>Struts Controller</servlet-name>
10:       <url-pattern>*.do</url-pattern>
11:    </servlet-mapping>
12: </web-app>
```

Listagem 4.1 – Declaração do Controlador do Struts na Instancia (web.xml)

Exemplo: uma requisição para http://aulanet.les.inf.puc-rio.br/postMessage.do, "http://aulanet.les.inf.puc-rio.br/" identifica o servidor que trata a requisição, ".do" identifica que o controlador do Struts será executado e "postMessage" identifica a ação que o controlador irá executar.

Através do arquivo XML de configuração do Struts, que por padrão chama-se struts-config.xml, a parte variável da URL é mapeada em ações. A Listagem 4.2 exibe um exemplo deste arquivo.

```
13: <struts-config>
14:     <form-beans>
15:         <form-bean name="postMessageForm"
16:                 type="br.puc-rio.inf.les.aulanet.PostMessageForm"/>
17:     </form-beans>
18:     <action-mappings>
19:         <action
20:                 path="/postMessage"
21:                 type="br.puc-rio.inf.les.aulanet.PostMessageAction"
22:                 name="postMessageForm"
23:                 scope="request" validate="true"
24:                 input="/conference/postMessage.jsp">
25:             <forward
26:                     name="success"
27:                     path="/conference/postMessageSuccess.jsp"/>
28:             <forward
29:                     name="failure"
30:                     path="/conference/postMessageFailure.jsp"/>
31:         </action>
32:     </action-mappings>
33: </struts-config>
```

Listagem 4.2 – Descritor struts-config.xml

Entre as linhas 19 e 31 encontra-se a definição de uma ação. Através do atributo *path* na linha 20, é indicado ao controlador do Struts que ação deve ser executada sempre que a parte variável da URL for “postMessage”. O atributo *type* na linha 21 identifica a classe que implementa a ação e que é chamada pelo controlador do Struts. A Listagem 4.3 mostra o exemplo de uma classe de ação.

```
34: public class PostMessageAction extends Action {

35:     public ActionForward execute(ActionMapping mapping,
36:             ActionForm form, HttpServletRequest request,
37:             HttpServletResponse response) throws ServletException {

38:         PostMessageForm formBean = (PostMessageForm) form;

39:         try {
40:             ConferenceFacade facade = new ConferenceFacadeImpl();
41:             facade.postMessage(request.getRemoteUser(),
42:                     formBean.getParentId(), formBean.getConferenceId(),
43:                     formBean.getCategoryId(), formBean.getMessage());

44:         } catch (Exception e) {
45:           return mapping.findForward("failure");
46:         }
47:         return mapping.findForward("success");
48:     }
49: }
```

Listagem 4.3 – Ação postMessage

Na linha 38, a ação realiza uma conversão de tipo para o *Action Form* especificado na linha 22 do arquivo struts-config.xml (Listagem 4.2). Na linha 41, o *Façade* é instanciado. Na linha 41, a ação chama a camada de negócios para atualizar o modelo. Na linha 47, o controlador é notificado para mostrar a vista de sucesso ou, caso aconteça algum erro, a vista de fracasso na linha 45. O *Action Form* para esta ação, descrito entre as linhas 15 e 16 do descritor struts-config.xml, é exibido na Listagem 4.4.

```
50: public class PostMessageForm extends ValidatorForm {

51:     private String parentId;
52:     private String conferenceId;
53:     private String categoryId;
54:     private Message message;

55:     public void setParentId(String newValue) {
56:         parentId = newValue;
57:     }
58:     public String getParentId() { return parentId; }

59:     public void setConferenceId(String newValue) {
60:         conferenceId = newValue;
61:     }
62:     public String getConferenceId() { return conferenceId; }

63:     public void setCategoryId(String newValue) {
64:         categoryId = newValue;
65:     }
66:     public String getCategory() { return categoryId; }

67:     public void setMessage(Message newValue) {
68:         message = newValue;
69:     }
70:     public Message getMessage() { return message; }
71: }
```

Listagem 4.4 – Action Form da ação postMessage

O *Action Form* é uma classe que não precisa implementar nenhum método especial além dos métodos *gets* e *sets* que definem propriedades JavaBeans (2005). Na linha 50 é declarado que este *Action Form* estende a classe ValidatorForm, uma subclasse de ActionForm que possibilita o uso de validações declarativas no Struts.

Usando o esquema de validação do Struts, são executadas validações tanto no lado do cliente quanto no lado do servidor através de regras declaradas em um arquivo descritor chamado validation.xml. A validação no lado do cliente é feita por JavaScript e evita que requisições desnecessárias sejam enviadas. Por outro lado, o JavaScript do navegador do usuário pode ser desabilitado então esta deve ser realizada também no lado do servidor, para garantir a integridade dos dados. A Listagem 4.5 exibe um exemplo de arquivo validation.xml.

```
72: <form-validation>
73:    <formset>
74:       <form name="postMessageForm">
75:          <field property="categoryId" depends="required">
76:             <msg name="required" key="postMessage.category.required"/>
77:             <arg0 key="postMessageForm.categoryId"/>
78:          </field>
79:          <field property="conferenceId" depends="required">
80:             <msg name="required" key="postMessage.conference.required"/>
81:             <arg0 key="postMessageForm.conferenceId"/>
82:          </field>
83:          <field property="message.subject" depends="required">
84:             <msg name="required" key="postMessage.message.subject.required"/>
85:             <arg0 key="postMessageForm.message.subject"/>
86:          </field>
87:          <field property="message.text" depends="required">
88:             <msg name="required" key="postMessage.message.text.required"/>
89:             <arg0 key="postMessageForm.message.text"/>
90:          </field>
91:       </form>
92:    </form-set>
93: </form-validation>
```

Listagem 4.5 – Regras de Validação (validation.xml)

A linha 74 declara um conjunto de regras de validação para o *Action Form* postMessageForm. Os trechos entre as linhas 75 e 78 definem que o campo *categoryId* é requirido. Assim como os trechos entre as linhas 79 e 82, 83 e 86, 87 e 90 fazem o mesmo para os campos *confereceId*, *message.subject* e *message.text*. Há outras regras de validações não usadas neste exemplo, como por exemplo, validar a sintaxe de e-mails e URLs, validar se um número pertence a uma determinar faixa, dentre outras.

O Struts também fornece um conjunto de bibliotecas de tags usadas nas páginas JSP que compõem a visão. Estas bibliotecas contêm tags que possibilitam o vínculo de campos de um formulário com atributos dos *Action Forms* e também tags especializadas em mostrar erros de validação. A Listagem 4.6 mostra como a página de envio de mensagens é construída, usando as tags do Struts.

```
094: <%@ taglib uri="http://jakarta.apache.org/struts/tags-html" prefix="html"%>
095: <%@ taglib uri="http://jakarta.apache.org/struts/tags-logic" prefix="logic"%>
096: <%@ taglib uri="http://jakarta.apache.org/struts/tags-bean" prefix="bean"%>

097: <html>
098:     <head>
099:         <title>Envio de Mensagem</title>
100:         <html:javascript formName="PostMessageForm"/>
101:     </head>
102:     <body>
103:         <html:form action="/postMessage" method="POST"
104:                 onsubmit="return validatePostMessageForm(this);">
105:             <html:hidden property="conferenceId" value="${param.conferenceId}"/>
106:             <html:hidden property="messageId" value="${param.messageId}"/>
107:             <logic:messagesPresent>
108:                 <html:messages id="error">
109:                     <bean:write name="error"/>
110:                 </html:messages>
111:             </logic:messagesPresent>
112:             Categoria:
113:             <html:select property="categoryId">
114:                 <html:option value="1">Questão</html:option>
115:                 <html:option value="2">Seminário</html:option>
116:                 <html:option value="3">Argumentação</html:option>
117:             </html:select><br>
118:             Assunto:
119:             <html:text property="message.subject" size="83" maxlength="100" /><br>
120:             Mensagem: <br>
121:             <html:textarea property="message.text" cols="65" rows="16" style="wrap" />
122:         </html:form>
123:     </body>
124: </html>
```

Listagem 4.6 – Página de Envio de Mensagens

Entre as linhas 094 e 096 é feita a declaração das bibliotecas de tags do Struts. Na linha 100 a tag html:javascript gera o código JavaScript responsável pela validação no lado cliente. Já na linha 103 a tag html:form gera o formulário que submete a requisição da ação postMessage. É importante notar que na linha 104, a propriedade onsubmit é definida com o valor validatePostMessageForm(this). Se este atributo não for configurado desta forma, o código de validação no lado cliente é gerado, mas nunca executado.

O trecho de código entre as linhas 107 e 111 geram a lista de erros de validação. A tag logic:messagesPresent é usada para evitar que uma lista sem itens seja exibida quando não há erros de validação. A tag html:messages é usada para

iterar sobre a lista de erros de validação. Finalmente a tag bean:write, é usada para escrever a mensagem de erro na página.

Nota-se na listagem 4.6 que cada uma das tags de formulário HTML (<form>, <input>, etc.) possui uma tag análoga na biblioteca de tags HTML do Struts. Estas tags são responsáveis por criar o vinculo entre o campo do formulário e a propriedade do *Action Form*.

4.4.2. Spring MVC

O Spring (2005), apresentado no capítulo 3 e que foi escolhido para ser usado na camada de negócios possui um módulo que implementa o MVC. Desta forma, é um candidato natural a ser usado também na camada de apresentação.

Os comandos no Spring MVC recebem o mesmo nome do controlador do Model View Controller e devem implementar a interface org.springframework.web.servlet.mvc.Controller. Geralmente não é necessário implementar esta interface diretamente. O Spring MVC possui classes apropriadas para situações específicas, como a submissão de um formulário web (classe SimpleFormController) ou então um conjunto de telas no formato de Wizard (classe AbstractWizardFormController). Ambas estão localizadas no pacote org.springframework.web.servlet.mvc.

De forma similar ao Struts, os controladores do Spring MVC podem vir acompanhados de classes que representam os dados submetidos em um formulário. Estas classes recebem o nome de Comandos (*Commands*) no Spring MVC. Este termo é inadequado, pois remete ao padrão de projeto Comando (Gamma et al., 1995). Para preservar a clareza do texto, nesta dissertação os comandos do Spring MVC serão chamados de beans de formulário.

O mapeamento entre requisições e controladores é feito de forma similar ao Struts. O servlet controlador é associado a uma URL com uma parte fixa e outra variável. A parte fixa serve para especificar que o servlet controlador do Spring MVC deve ser executado. A parte variável serve para identificar qual controlador deve ser executado. A Listagem 4.7 exibe o mapeamento do servlet controlador no descritor da aplicação web.xml. Nota-se que o mapeamento realizado na linha

08, “*.spring”, indica que a parte fixa é “.spring” e a parte variável é o que quer que venha antes.

```
01: <web-app>
02:    <servlet>
03:       <servlet-name>SpringDispatcher</servlet-name>
04:       <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
05:    </servlet>
06:    <servlet-mapping>
07:       <servlet-name>SpringDispatcher</servlet-name>
08:        <url-pattern>*.spring</url-pattern>
09:    </servlet-mapping>
10: </web-app>
```

Listagem 4.7 - Declaração do Controlador do Spring MVC na Instancia (web.xml)

Exemplo: em uma requisição enviada para http://aulanet.les.inf.puc-rio.br/postMessage.spring, “http://aulanet.les.inf.puc-rio.br/” identifica o servidor que atende a requisição, “.spring” identifica que o controlador do Spring MVC será executado e “postMessage” identifica que o controlador associado a ação “postMessage” será executado.

Através do arquivo XML de configuração do Spring MVC mostrado na Listagem 4.8 a parte variável da URL é mapeada aos controladores.

```
11: <beans>
12:    <bean id="postMessageController"
13:            class="br.pucrio.inf.les.aulanet.PostMessageController">
14:       <property name="conferenceFacade">
15:          <ref bean="conferenceFacade"/>
16:       </property>
17:       <property name="formView">
18:          <value>/protected/conference/postMessage.jsp</value>
19:       </property>
20:       <property name="successView">
21:          <value>/conference/postMessageSuccess.jsp</value>
22:       </property>
23:       <property name="failureView">
24:          <value>/conference/postMessageFailure.jsp</value>
25:       </property>
26:      <property name="validator">
27:          <bean class="br.pucrio.inf.les.aulanet.PostMessageValidator"/>
28:       </property>
29:    </bean>
30:    <bean id="urlMapping"
31:          class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
32:       <property name="mappings">
33:          <props>
34:             <prop key="/postMessage.spring">postMessageController</prop>
35:          </props>
36:       </property>
37:    </bean>
38: </beans>
```

Listagem 4.8 - Descritor SpringDispatcher-servlet.xml

Este arquivo por padrão chama-se <Nome da Servlet>-servlet.xml, onde <Nome da Servlet> é o nome da servlet declarado na linha 03 da Listagem 4.7. Na linha 34 da Listagem 4.8 a requisição “postMessage” é mapeada para o controlador de nome “postMessageController”, declarado no mesmo arquivo entre as linhas 12 e 29. Neste exemplo são configuradas cinco propriedades importantes do controlador: entre as linhas 14 e 16 é configurado o *Façade* usado para executar a lógica de negócios. Entre as linhas 17 e 19 é configurado a página de entrada, ou seja, a tela que origina a requisição. Entre as linhas 20 e 22 é configurada a página de sucesso, exibida quando a operação é executada corretamente. Por fim, entre as linhas 23 e 25 é configurada a página de erros, executada quando ocorre algum problema durante o acesso à camada de negócios.

Entre as linhas 26 e 28 o controlador é mapeado ao seu validador, que será explicado mais adiante. Vale ressaltar que a sintaxe usada neste descritor é a mesma dos descritores dos outros módulos do Spring usados na camada de

negócios. O controlador associado à requisição postMessage é mostrado na Listagem 4.9.

```
39: public class PostMessageController extends SimpleFormController {

40:     private ConferenceFacade confFacade;
41:     private String failureView;

42:     public PostMessageController() {
43:         setCommandClass(Message.class);
44:     }

45:     public void setConferenceFacade(ConferenceFacade newValue) {
46:         confFacade = newValue;
47:     }
48:     public void setFailureView(String newValue) {
49:         failureView = newValue;
50:     }

51:     protected ModelAndView onSubmit(HttpServletRequest request,
52:             HttpServletResponse response, Object command, BindException errors)
53:             throws Exception {
54:         Message message = (Message) command;

55:         try {
56:             facade.postMessage(request.getRemoteUser(),
57:                     message.getParentId(), message.getConferenceId(),
58:                     message.getCategoryId(), message);
59:         } catch (Exception e) {
60:             return new ModelAndView(failureView);
61:         }
62:         return new ModelAndView(getSuccessView());
63:     }
64: }
```

Listagem 4.9 – Controlador Associado à Requisição postMessage

Na linha 39 da Listagem 4.9 a classe do controlador estende a classe SimpleFormController, indicando que este controlador responde à submissão de um formulário. As linhas 40 e 41 apresentam as propriedades para, respectivamente, o *Façade* e a página exibida quando ocorre erro. Os métodos de acesso para estas propriedades são definidos entre as linhas 45 e 50. Na linha 43, o bean de formulário Message é associado a este controlador.

Entre as linhas 51 e 63 encontra-se o código do controle que é executado quando o formulário é submetido. Primeiro, é realizada uma conversão de tipos na

linha 54. Depois, o *Façade* é chamado na linha 56. Diferentemente do Struts, o *Façade* é configurado através de *Dependency Injection* ao invés de ser instanciado, o que leva a um menor acoplamento entre a camada de apresentação e a implementação da camada de negócios.

Se a operação for bem sucedida, é retornado um objeto ModelAndView que encapsula o modelo e a visão apontando para a tela de sucesso (linha 62). Se ocorrer algum erro durante a execução da lógica de negócios, a tela de erros é exibida (linha 60).

Diferentemente do Struts, os beans de formulário no Spring MVC não estendem uma classe do framework. Dessa forma, os beans de formulário não possuem dependências com o Spring MVC, podendo inclusive ser reusadas classes do modelo como beans de formulário sempre que for adequado. A Listagem 4.10 mostra o bean de formulário usado pelo controlador postMessage.


```
65: public class Message {

66:     private String parentId;
67:     private String conferenceId;
68:     private String categoryId;
69:     private String message;
70:     private String subject;

71:     public void setParentId(String newValue) { parentId = newValue; }
72:     public String getParentId() { return parentId; }

73:     public void setConferenceId(String newValue) { conferenceId = newValue; }
74:     public String getConferenceId() { return conferenceId; }

75:     public void setCategoryId(String newValue) { categoryId = newValue; }
76:     public String getCategory() { return categoryId; }

77:     public void setMessage(String newValue) { message = newValue; }
78:     public String getMessage() { return message; }

79:     public void setSubject(String newValue) { subject = newValue; }
80:     public String getSubject() { return subject; }
81: }
```

Listagem 4.10 – Bean de Formulário Associado ao Controlador postMessage

Com relação a validação, quando este capítulo foi escrito em novembro de 2005, o esquema de validação do Spring MVC não se encontrava num estado de maturidade tão elevado quanto o do Struts. Além de não oferecer validação no

lado do servidor, o esquema do Spring MVC não é baseado em regras descritivas, ou seja, a validação precisa ser programada e associada à classe do controlador. Atualmente há uma iniciativa para se criar um esquema de validação similar ao do Struts para o Spring MVC, mas ainda não há previsão de quando haverá uma versão estável. A Listagem 4.11 mostra a classe de validação para o controlador postMessage.

```
82: public class SendMessageValidator implements Validator {

83:     public boolean supports(Class clazz) {
84:         return clazz.equals(Message.class);
85:     }

86:     public void validate(Object command, Errors errors) {
87:         ValidationUtils.rejectIfEmptyOrWhitespace(errors, "categoryId",
88:             "postMessage.categoryId.required", "Category empty.");
89:         ValidationUtils.rejectIfEmptyOrWhitespace(errors, "conferenceId",
90:             "postMessage.conference.required", "Conference empty.");
91:         ValidationUtils.rejectIfEmptyOrWhitespace(errors, "message",
92:             "postMessage.message.required", " Message empty.");
93:         ValidationUtils.rejectIfEmptyOrWhitespace(errors, "subject",
94:             "postMessage.subject.required", "Subject empty.");

95:     }
96: }
```

Listagem 4.11 – Classe de Validação para o Controlador postMessage

Na linha 84 o validador especifica os tipos de beans de formulário que ele valida, neste caso apenas o bean Message. Nas linhas 87 e 88 é validado se o campo *categoryId* é vazio. O mesmo é feito nas linhas 89 e 90 para a propriedade *conferenceId*, nas linhas 91 e 92 para a propriedade *message* e nas linhas 93 e 94 para a propriedade *subject*.

O Spring MVC também oferece um conjunto de tags usadas para exibir mensagens de erro de validação na camada de apresentação, mas ao contrário do Struts, possibilita que sejam usadas as tags do próprio HTML para gerar os campos de entrada dos formulários. O vínculo entre as propriedades dos beans de formulário e o formulário é feito com outras tags do Spring MVC. Esta abordagem deixa as telas da camada de apresentação menos acopladas com o web

framework. A Listagem 4.12 exibe o código fonte da página de envio de mensagens.

```
097: <%@ taglib uri="http://www.springframework.org/tags" prefix="spring" %>
098: <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
099: <html>
100:    <head><title>Envio de Mensagem</title></head>
101:    <body>
102:       <form action="sendMessage.spring" method="POST">
103:          <spring:bind path="command.conferenceId">
104:             <input type="hidden" name='<c:out value="${status.expression}"/>'
105:                    value='${param.conferenceId}'/>
106:          </spring:bind>
107:          <spring:bind path="command.parentId">
108:             <input type=hidden name='<c:out value="${status.expression}"/>'
109:                    value='${param.messageId}'/>
110:          </spring:bind>
111:          <spring:hasBindErrors name="command">
112:             <ul>
113:                <c:forEach items="${errors.allErrors}" var="error">
114:                   <li><spring:message
115:                            code="${error.code}"
116:                            text="${error.defaultMessage}"
117:                            arguments="${error.arguments}"/></li>
118:                </c:forEach>
119:             </ul>
120:          </spring:hasBindErrors>
121:          Categoria:
122:          <spring:bind path="command.categoryId">
123:             <select name='<c:out value="${status.expression}"/>'>
124:                <option value="1">Questão</option>
125:                <option value="2">Seminário</option>
126:             </select><br>
127:          </spring:bind>
128:          Assunto:
129:          <spring:bind path="command.subject">

130:             <input type="text" name='<c:out value="${status.expression}"/>'
131:                    value='<c:out value="${status.value}"/>'
132:                    size="83" maxlength="100"/>
133:          </spring:bind>
134:          Mensagem: <br>
135:          <spring:bind path="command.message">
136:             <textarea name='<c:out value="${status.expression}"/>' rows="15"
137:                       cols="60"  cols=65 rows=16 wrap>
138:                <c:out value="${status.value}"/>
139:             </textarea>
140:          </spring:bind>
141:       </form>
142:       </body>
143: </html>
```

Listagem 4.12 - Página de Envio de Mensagem

Nota-se o uso constante da tag spring:bind, que é usada para vincular as propriedades do bean de formulário aos campos do formulário html. Outro ponto que deve ser observado é o uso das tag spring:hasBindErrors e spring:message usadas para listar os erros de validação quando estes houverem.

4.4.3. JavaServer Faces

JavaServer Faces (JSF, 2005) é diferente dos web frameworks apresentados até então pois além de ser um framework de infra-estrutura, também pode ser classificado como um framework de componentes, pois define um modelo de componentes para serem usados na camada de apresentação. Há três tipos principais de componentes no JSF: componentes de interface, usado para compor interfaces com o usuário, componentes de conversão de dados, usados para converter os dados inseridos pelo usuário em tipos da aplicação e componentes de validação de dados, usados para validar a entrada do usuário. JSF vem com um conjunto padrão de componentes, que podem ser estendidos.

Os comandos no JSF recebem o nome de *backing beans* ou *managed beans*. Estes não precisam implementar interfaces específicas ou estender classes, bastando que o método do comando seja público, não receba parâmetros e tenha o tipo de retorno String. Componentes de interface vinculam *hyperlinks* ou botões a ações de comando. Os *backing beans* contém os dados dos formulários relacionados de forma que não é necessário criar classes extras, como beans de formulário ou *Action Forms* para isto.

O mapeamento entre requisições e controladores é feito através de campos escondidos em formulários, inseridos automaticamente pelos componentes de comando. Assim como no Spring MVC e no Struts, as URLs de requisição são divididas em uma parte fixa e uma parte variável. A parte fixa serve para associar a URL de requisição ao Servlet controlador do JSF e a parte variável da URL serve para identificar qual página JSP que contém componentes JSF será executada. A **Listagem 4.13** exibe o mapeamento do servlet controlador no descritor da aplicação web.xml. Nota-se que o mapeamento realizado na linha 08, “*.jsf”, indica que a parte fixa é “.jsf” e a parte variável é o que quer que venha antes.

```
01: <web-app>
02:    <servlet>
03:       <servlet-name>Faces Servlet</servlet-name>
04:       <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
05:    </servlet>
06:    <servlet-mapping>
07:       <servlet-name>Faces Servlet</servlet-name>
08:       <url-pattern>*.jsf</url-pattern>
09:    </servlet-mapping>
10: </web-app>
```

Listagem 4.13 - Declaração do Controlador do JSF na Instancia (web.xml)

Exemplo: em uma requisição enviada para http://aulanet.les.inf.puc-rio.br/postMessage.jsf, “http://aulanet.les.inf.puc-rio.br/” identifica o servidor que atende a requisição, “.jsf” identifica que o controlador do JSF será executado e “postMessage” identifica a página JSP com componentes JSF associados solicitada.

Através do arquivo XML de configuração do JSF são configurados casos de navegação, *backing beans*, componentes de validação e de conversão de tipos. A Listagem 4.14 exibe um exemplo deste arquivo, que tem como nome padrão faces-config.xml.

```
11: <faces-config>
12:    <navigation-rule>
13:       <from-view-id>/conference/postMessage.jsp</from-view-id>
14:       <navigation-case>
15:          <from-outcome>success</from-outcome>
16:          <to-view-id>/conference/postMessageSuccess.jsp</to-view-id>
17:       </navigation-case>
18:       <navigation-case>
19:          <from-outcome>failure</from-outcome>
20:          <to-view-id>/conference/postMessageFailure.jsp</to-view-id>
21:       </navigation-case>
22:    </navigation-rule>
23:    <managed-bean>
24:       <managed-bean-name>messageController</managed-bean-name>
25:       <managed-bean-class>MessageController</managed-bean-class>
26:       <managed-bean-scope>request</managed-bean-scope>
27:       <managed-property>
28:          <property-name>conferenceId</property-name>
29:          <value>#{param['conferenceId']}</value>
30:       </managed-property>
31:       <managed-property>
32:          <property-name>parentId</property-name>
33:          <value>#{param['parentId']}</value>
34:       </managed-property>
35:    </managed-bean>
36: </faces-config>
```

Listagem 4.14 – Arquivo de Configuração faces-config.xml

Entre as linhas 12 e 22 são declarados dois casos de navegação a partir da página /conference/postMessage.jsp. O caso entre as linhas 14 e 17 é executado quando a operação é executada com sucesso enquanto que o caso entre as linhas 18 e 21 é executado quando ocorre algum erro durante a execução da operação.

O *backing bean* responsável pelo comando postMessage é declarado entre as linhas 23 e 37. Na linha 24 o nome do *backing bean* é definido, enquanto que na linha 25 é configurada a classe. Nota-se também que este bean tem duas propriedades gerenciadas (*managed properties*) definidas entre as linhas 27 e 34. O JSF se encarrega de configurar estas propriedades em tempo de execução, antes de executar o método do *backing bean*. A Listagem 4.15 exibe o código fonte deste *backing bean*.

```
37: public class MessageController {

38:     private String parentId;
39:     private String conferenceId;
40:     private String categoryId;
41:     private Message message;

42:     public void setParentId(String newValue) { parentId = newValue; }
43:     public String getParentId() { return parentId;  }
44:     public void setConferenceId(String newValue) { conferenceId = newValue; }
45:     public String getConferenceId() { return conferenceId;  }
46:     public void setCategoryId(String newValue) { categoryId = newValue; }
47:     public String getCategoryId() { return categoryId;  }
48:     public void setMessage(Message message) { message = newValue; }
49:     public Message getMessage() { return message;  }

50:     public String postMessage() {

51:         try {
52:             ConferenceFacade facade = new ConferenceFacadeImpl();
53:             FacesContext ctx = FacesContext.getCurrentInstance();
54:             String author = ctx.getExternalContext().getRemoteUser();
55:             facade.postMessage(author, parentId, conferenceId,  categoryId, message);
56:         } catch (Exception e) {
57:             return "failure";
58:         }
59:         return "success";
60:     }
61: }
```

Listagem 4.15 – Backing Bean do Comando postMessage

Entre as linhas 38 e 41 são declaradas as propriedades *parentId*, *conferenceId*, *categoryId* e *message* e entre as linhas 42 e 49 são declarados os métodos de acesso a estas propriedades.

O método que executa o comando é declarado entre as linhas 50 e 60. Na linha 52, a referência para o *Façade* que acessa a camada de negócios é obtida. Na linha 54, é recuperado o nome de usuário logado a partir do contexto Faces recuperado na linha 53. A lógica de negócios é executada na linha 55 e, caso haja sucesso, o comando aciona o caso de navegação “success” (linha 59). Se houver algum erro na execução da lógica de negócios, a tela de erro é acionada (linha 57).

Apesar de não precisar implementar interfaces especificas do JSF, o *backing bean* não está totalmente desacoplado do framework. Na linha 53, o contexto

faces (FacesContext) é recuperado para posteriormente o nome do usuário logado ser recuperado.

A declaração das regras de validação assim como o vínculo entre campos de formulário e propriedades do *backing bean*, e entre os componentes de comando e os métodos de comando nos *backing beans* são feitas no arquivo JSP. A Listagem 4.16 mostra a página JSP usada no comando postMessage.

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

64: <f:view>
65:     <html>
66:         <head>
67:             <title>Envio de Mensagem</title>
68:         </head>
69:         <body>
70:             <h:form>
71:                 <h:inputHidden id="conferenceId" value="#{param['conferenceId']}"/>
72:                 <h:inputHidden id="parentId" value="#{param['parentId']}"/>
73:                 <h:messages/><br>
74:                 Categoria:
75:                 <h:selectOneListbox value="#{messageController.categoryId}"
76:                                     required="true">
77:                     <f:selectItem itemLabel="Questão" itemValue="1"/>
78:                     <f:selectItem itemLabel="Seminário" itemValue="2"/>
79:                     <f:selectItem itemLabel="Argumentação" itemValue="3"/>
80:                 </h:selectOneListbox><br>
81:                 Assunto:
82:                 <h:inputText value="#{messageController.message.subject}"
83:                              size="83" maxlength="100" required="true">
84:                     <f:validateLength minimum="10"/>
85:                 </h:inputText><br>
86:                 Mensagem: <br>
87:                 <h:inputTextarea value="#{messageController.message.text}"
                                     rows="15" cols="60"  cols="65" rows="16"
88:                                  required="true"/>
89:             </h:form>
90:         </body>
91:     </html>
92: </f:view>
```

Listagem 4.16 – Página de Envio De Mensagens (JSF)

As bibliotecas de tags definidas nas linhas 62 e 63 possibilitam que JSF seja usado em conjunto com JSP. A tag f:view (linha 64) serve para demarcar árvore de componentes JSF. A tag h:form (linha 70) é usada para definir um formulário

de dados. As linhas 71 e 72 inserem campos escondidos, com as propriedades *conferenceId* e *parentId*. Expressões delimitadas por "#{" e "}" demarcam o uso da linguagem de expressões do JSF e servem para criar vínculos. No caso destas expressões, o uso de "#{param['nome-do-parametro']}" indica o vínculo do valor dos componentes com as propriedade de requisição indicadas pelo nome "nome-do-parametro".

O item entre as linhas 75 e 80 delimita um componente do tipo lista onde apenas um item pode ser selecionado. O atributo *value* na linha 75 vincula o valor do item selecionado a propriedade *categoryId* do *backing bean* messageController. Ainda na linha 75, o atributo required="true" demarca que o atributo é requerido, gerando regras de validação automaticamente. JSF ainda não gera regras de validação do lado do cliente, mas há como integrar o módulo de validação no lado cliente do Struts ao JSF, conforme descrito em Geary & Horstmann (2005). A linha 84 indica uma validação de tamanho mínimo, indicando que o campo associado a esta só será válido se tiver um tamanho mínimo de 10 caracteres.

JSF possibilita que novos componentes sejam criados caso os componentes padrão não satisfaçam as necessidades da aplicação. Projetos como o MyFaces (2005) e WebGalileo Faces (WebGalileo, 2005) disponibilizam componentes, como por exemplo, um que possibilita que o usuário selecione uma data em um calendário (**Figura 4.3**) ou um que apresenta um editor de textos HTML estilo *What You See Is What You Get* (**Figura 4.4**), que já foi solicitado por diversos usuários do AulaNet (Barreto et al., 2005).

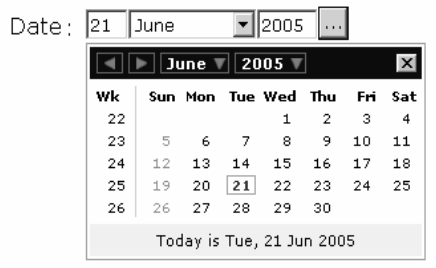

Figura 4.3 - Calendário

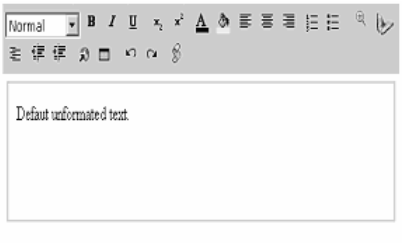

Figura 4.4 - Editor HTML

Além disso, JSF foi desenvolvido visando à integração com ferramentas de produtividade. Consequentemente há uma boa quantidade de ferramentas

compatíveis com JSF no mercado conforme descrito na seção 4.5.3, apesar deste ser o mais novo dos 3 web frameworks analisados.

4.4.4. Resultado da Análise Técnica

Após analisar tecnicamente os web frameworks Struts, Spring MVC e JSF percebe-se que cada um tem suas vantagens e desvantagens.

Struts é o mais antigo dos três e, portanto, o mais maduro. Seu módulo de validação baseado em regras capaz de validar dados tanto no lado do cliente quanto no servidor é sua principal vantagem. Entretanto, o Struts força um grau elevado de acoplamento entre a camada de apresentação e o web framework usado ao impor que tanto *Actions* quanto *Action Forms* estendam classes próprias do frameworks. Além disso, o Struts exige que sejam criadas classes específicas para representar os dados inseridos em formulários, mesmo quando uma classe do modelo é mais apropriada.

Spring MVC por outro lado apresenta um fraco grau de acoplamento. Os beans de formulário não precisam estender classes ou implementar classes do framework, podendo ser aproveitadas classes do modelo para esta função. Spring MVC também fornece a melhor solução para integração entre os controladores na camada de apresentação e os *Façades* na camada de negócios, através de *Dependency Injection* (Fowler, 2004). Contudo, seu mecanismo de validação é sua principal desvantagem, sendo o pior dos 3 analisados.

JSF se difere dos três por ser um framework de componentes. JSF não possui um grau de acoplamento tão forte quanto o Struts e nem tão fraco quanto o o Spring MVC. Seu módulo de validação é baseado em regras, como o do Struts, mas não fornece validação do lado do cliente.

O que torna JSF a escolha técnica mais adequada ao AulaNet 3.0 é o fato dele ser um framework de componentes. JSF possibilita que componentes de interfaces sejam criados e aproveitados (desafio 6). Além disso, como foi visto em seções anteriores, já há um mercado de componentes compostos por projetos gratuitos (por exemplo, MyFaces (2005) e WebGalileo Faces (WebGalileo, 2005)) e comerciais (por exemplo, WebCharts (2005)) que complementam o conjunto padrão de componentes JSF, diminuindo a necessidade de criar novos.

JSF apresenta um custo inicial de adoção superior aos dos outros web frameworks analisados, pois exige que componentes sejam criados. Este custo é inerente à abordagem de desenvolvimento baseado a componentes e é reduzido à medida que o projeto evolui até atingir massa crítica, quando há componentes de variedade suficiente e quando a necessidade de criar um novo componente diminui (Szyperski, 1997). Como o AulaNet encontra-se em constante desenvolvimento espera-se que após a massa crítica ser atingida o custo do desenvolvimento com JSF seja menor do que com os outros web frameworks.

Apesar de não gerar validação no lado do cliente, principal vantagem do Struts, há maneiras de usar este esquema de validação junto com o JSF caso seja necessário (Geary & Horstmann, 2005). Quanto ao seu grau de acoplamento superior ao do Spring MVC não é um motivo suficientemente forte para abandoná-lo tendo em vista suas outras vantagens.

Somente a comparação técnica entre frameworks não é suficiente para tomar a decisão sobre qual dos web frameworks é mais adequado às necessidades de um projeto. No caso do AulaNet, como foi visto anteriormente, deve se levar em conta que ele é desenvolvido por alunos de graduação, mestrado e doutorado no Laboratório de Engenharia de Software da PUC-Rio e que EduWeb realiza customizações para seus clientes. Na próxima seção são analisados os seguintes fatores não-técnicos: como, por exemplo, a documentação disponível sobre cada web framework e a quantidade de profissionais habilitados a trabalhar com este web framework. Ao considerar estes fatores, evita-se a escolha de um web framework com custo de adoção proibitivo para o LES ou para a EduWeb.

4.5. Comparação Não-Técnica entre Web Frameworks

Além de analisar como os web frameworks funcionam tecnicamente, a escolha do web framework mais adequado para um projeto deve levar em considerações outros fatores, como: quantidade de documentação disponível, disponibilidade de suporte (seja um suporte oficial ou da comunidade que usa o framework), disponibilidade de ferramentas compatíveis, grau de aceitação de mercado e disponibilidade de profissionais com habilidades nos frameworks. Raible (2005) apresenta dados importantes que ajudarão esta análise, comparando

os frameworks que foram analisados tecnicamente na seção 4.4 e mais outros 3 web frameworks: Cocoon (2005), WebWork (2005) e Tapestry (2005).

Para uma análise mais precisa dos dados, foi necessário realizar comunicação pessoal com o autor que gentilmente cedeu os dados de sua pesquisa. Além disso, Matt Raible forneceu atualizações nos dados referentes à disponibilidade de suporte. As transcrições dos e-mails trocados com Raible estão disponíveis no apêndice A.

4.5.1. Quantidade de Documentação Disponível

Para medir a quantidade da documentação disponível faz-se uma pesquisa na quantidade de livros publicados e de tutoriais escritos sobre cada web framework. O resultado da pesquisa realizada por Raible (2005) no verão (hemisfério norte) de 2005 é apresentado na Figura 4.5 e Figura 4.6.

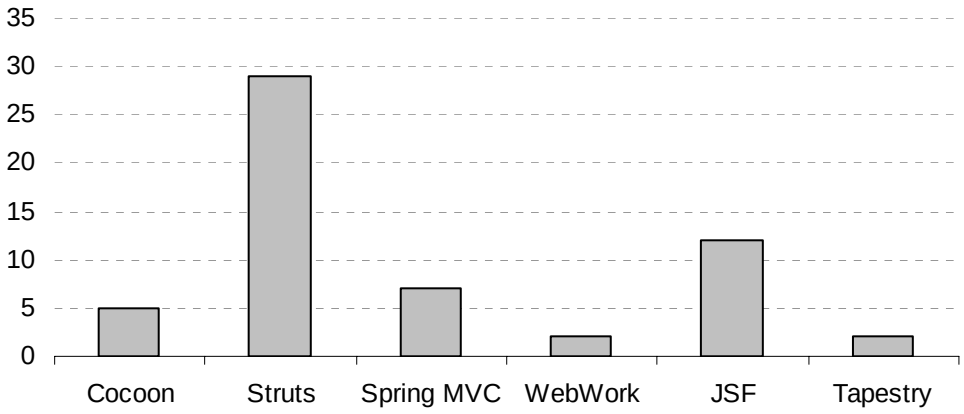

Figura 4.5 - Número de Livros Publicados
Fonte: Amazon.com (Raible, 2005), Verão de 2005 (hemisfério norte)

Há mais livros publicados sobre o Struts, com JSF vindo em segundo lugar e Spring MVC em terceiro. A Figura 4.6 mostra o número de tutoriais disponíveis na web para cada web framework segundo a pesquisa realizada no Google por Raible (2005) no verão de 2005 (hemisfério norte).

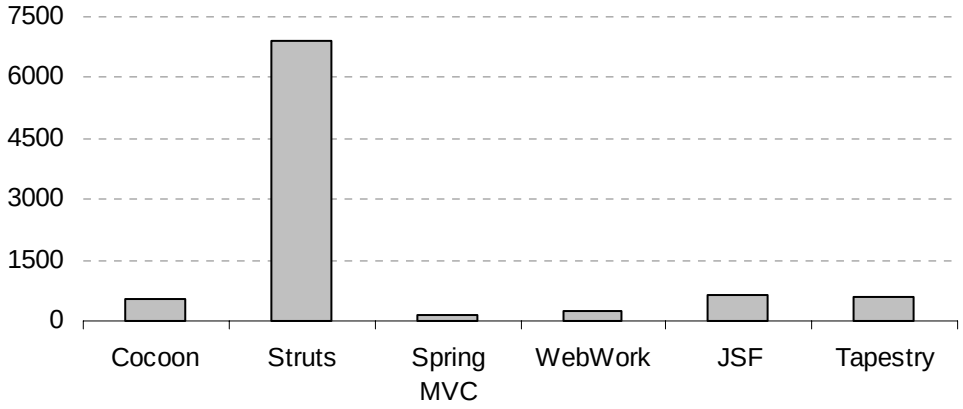

Figura 4.6 - Número de Tutoriais escritos
Fonte: Google.com (Raible, 2005), Verão de 2005 (hemisfério norte)

Struts tem cerca de 7 mil artigos e tutoriais disponíveis na web, com o JSF vindo em segundo lugar com cerca de 620 e seguido por Tapestry e Cocoon praticamente empatados, com aproximadamente 570 artigos e tutoriais disponíveis cada um.

Nota-se que há uma quantidade significantemente maior de documentação disponível sobre o Struts, muito provavelmente devido a este ter sido um dos primeiros web frameworks a serem criados, com o JSF vindo em segundo lugar.

4.5.2. Disponibilidade de Suporte

Ao optar por um web framework é importante ter disponível algum tipo de suporte, para solucionar questões não respondidas em livros e tutoriais. Todos os web frameworks analisados têm comunidades ativas que oferecem suporte gratuitamente. A Figura 4.7 mostra a quantidade de mensagens trocadas sobre cada web framework nas listas de discussões mantidas pelas comunidades segundo a pesquisa realizada por Raible (2005) em julho de 2005.

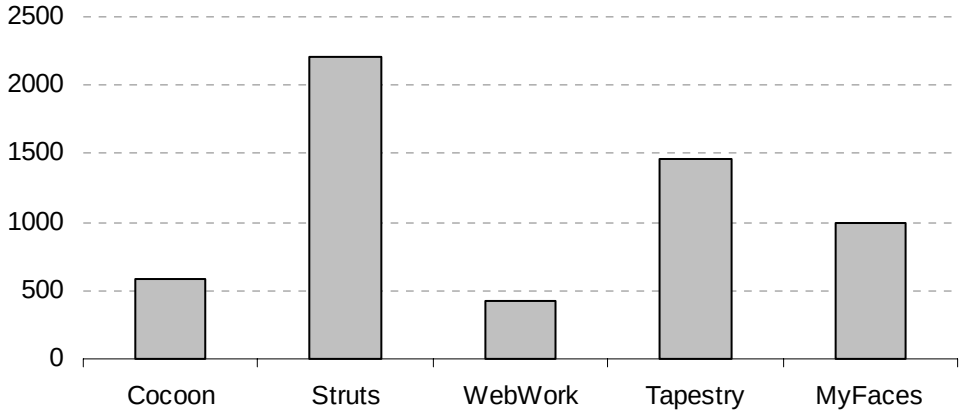

Figura 4.7 – Mensagens Trocadas em Listas de Discussão (Julho/2005)
(Raible, 2005)

A comunidade mais ativa é a do Struts com cerca de 2200 mensagens, sendo seguida pela comunidade do Tapestry com cerca de 1400 mensagens e depois pelas comunidades do MyFaces (implementação de JSF do grupo Jakarta) com aproximadamente 1000 mensagens. Nota-se que Spring MVC não aparece entre os listados. Isto porque o suporte do Spring MVC é feito através de um fórum, que não fornece meios de amostrar a quantidade de mensagens trocadas em determinado mês. Ainda que pudesse se amostrar o volume de e-mails trocados na lista de discussão não oficial do Spring (como feito no capítulo 4) não haveria como distinguir quantas mensagens dizem respeito ao módulo MVC do Spring e quantas dizem respeito aos outros módulos. Todavia, além do suporte prestado no fórum o Spring conta com suporte comercial prestado pela empresa Interface 21 (2005).

4.5.3. Disponibilidade de Ferramentas Compatíveis

Ferramentas compatíveis com um web framework auxiliam o desenvolvimento, diminuindo a necessidade de produção de código repetitivo que passa a ser feito pela ferramenta. A Figura 4.8 mostra o número de ferramentas disponíveis compatíveis com cada web framework, segundo pesquisa realizada por Raible (2005) no verão de 2005 (hemisfério norte).

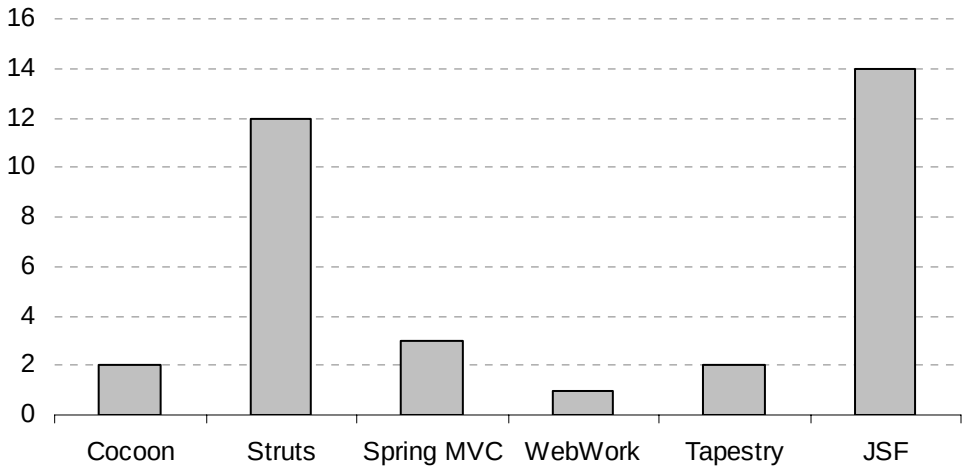

Figura 4.8 – Ferramentas Compatíveis
(Raible, 2005), Verão de 2005 (hemisfério norte)

Percebe-se que há mais ferramentas compatíveis com JSF, o que não é surpresa já que o JSF é um padrão aberto que foi desenvolvido objetivando compatibilidade com ferramentas de produtividade. Logo em seguida vem o Struts e em terceiro lugar o Spring MVC.

4.5.4. Grau de Aceitação no Mercado

Ao analisar o grau de aceitação que um web framework recebe no mercado pode ser inferido o rumo que a comunidade está tomando e desta forma, evitar uma decisão destoante da realidade do mercado. Para analisar o grau de aceitação de um web framework foi feita uma pesquisa nas ofertas de empregos que exigem conhecimento em cada um dos frameworks. A tendência do mercado pode ser observada ao comparar a variação do grau de aceitação dos web Frameworks em vários períodos de tempo.

Para analisar este critério Raible (2005) faz uma busca no site de empregos Dice.com. Os dados da pesquisa referem-se aos meses de outubro de 2004 e junho de 2005. Realizando uma busca em novembro de 2005 no mesmo site e reproduzindo o experimento de Raible complementei os dados da pesquisa com valores mais atuais. A Figura 4.9 mostra o resultado da pesquisa.

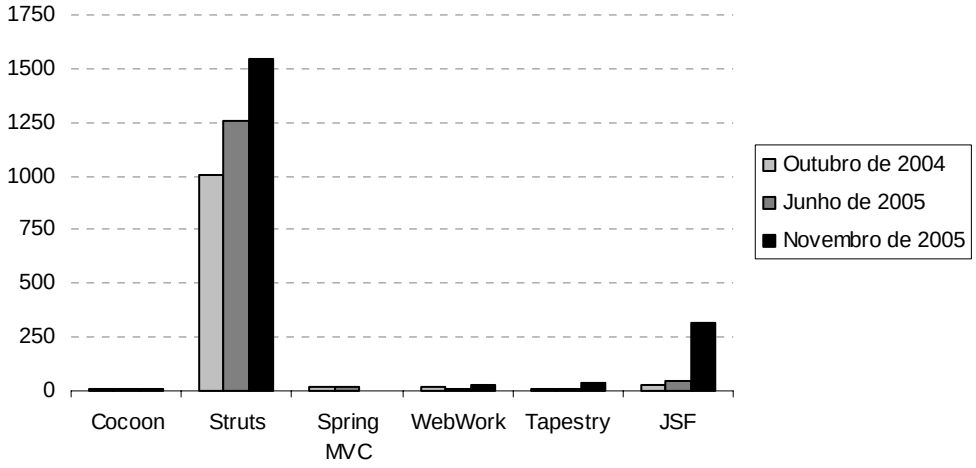

Figura 4.9 - Oferta de Empregos nos Meses 10/2004, 7/2005 e 11/2005
Fonte: Dice.com, 10/2004, 07/2005 (Raible, 2005) e 11/2005

Percebe-se claramente que o Struts é o framework mais mencionado em ofertas de empregos no site Dice.com, mas que JSF foi o web framework cujo número de ofertas de emprego mais cresceu ao longo do período analisado.

Estes dados representam à realidade do mercado norte-americano. Uma busca feita em novembro de 2005 no site brasileiro Manager Online (http://www.manageronline.com.br/) revela a realidade do mercado nacional (figura 4.11).

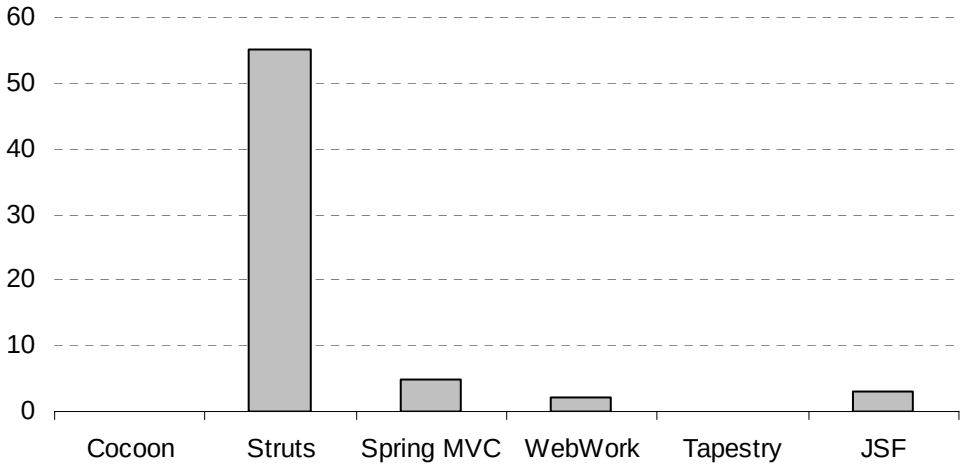

Figura 4.10 – Ofertas de Emprego
Fonte: Manager Online, Outubro de 2005

Novamente há uma discrepância entre o número de vagas disponíveis que pedem conhecimentos em Struts e as que pedem conhecimento dos outros web frameworks. Spring MVC encontra-se em segundo lugar com uma vantagem pouco significativa sobre JSF.

4.5.5. Disponibilidade de Profissionais

Antes de escolher um web framework é importante conhecer a disponibilidades de mão de obra no mercado de trabalho que esteja preparada para trabalhar com o framework. Escolher um web framework pouco conhecido implicará em custos de treinamento de pessoal. Para avaliar a disponibilidade de profissionais habilitados a lidar com estes frameworks foi feito uma pesquisa em sites que hospedam currículos. A Figura 4.11 mostra uma pesquisa feita em junho de 2005 por Raible (2005) no site Jobs.net que reflete a realidade do mercado norte-americano.

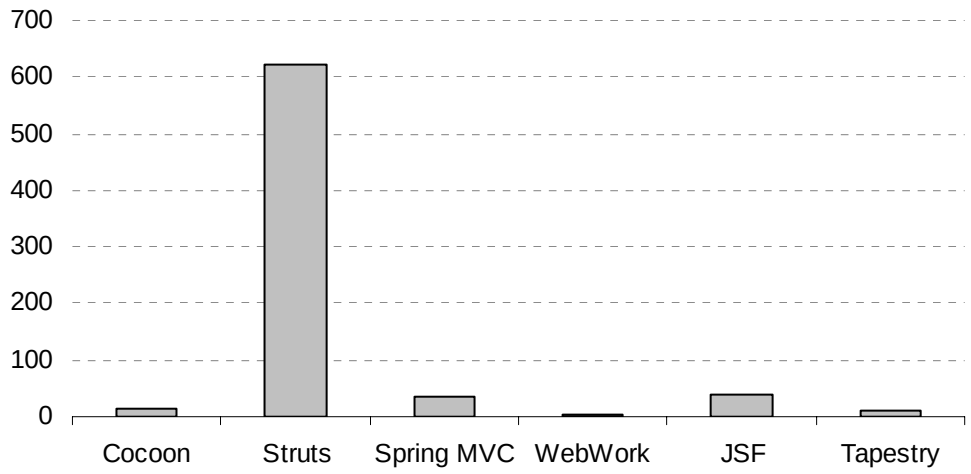

Figura 4.11 - Disponibilidade de Profissionais com Habilidades nos Frameworks
Fonte: Monster.com, Junho de 2005 (Raible, 2005)

Struts segue com clara liderança seguido pelo Spring e JSF que seguem empatados. A figura abaixo mostra uma pesquisa similar realizada no site brasileiro APInfo.com em novembro de 2005, revelando a disponibilidade de profissionais com habilidades nos web frameworks no mercado nacional.

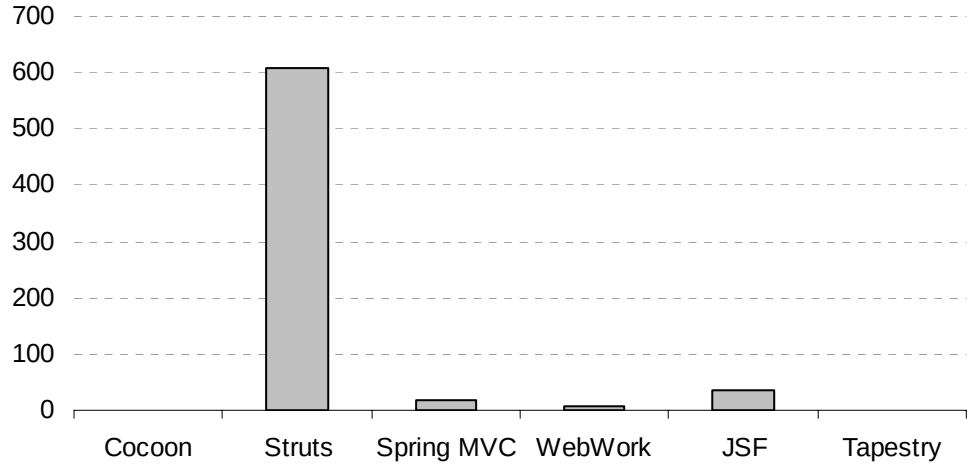

Figura 4.12 - Disponibilidade de Profissionais com Habilidades nos Frameworks
Fonte: APInfo.com, Novembro de 2005

O gráfico da Figura 4.12 mostra que, no Brasil, Struts também é o web framework com mais profissionais aptos disponíveis no mercado de trabalho. O JSF ocupa a segunda colocação com uma pequena vantagem sobre o Spring MVC.

Nota-se também que a quantidade de profissionais no Brasil e nos EUA é muito próxima, enquanto que nas pesquisas para as outras categorias de frameworks encontrou-se um número menor de profissionais aptos no Brasil. Isto pode ter ocorrido por diversos motivos: os profissionais brasileiros utilizam mais os web frameworks do que frameworks ORM e frameworks para *Dependency Injection*; os dados de Raible (2005) são referentes a junho de 2005 enquanto que a pesquisa no mercado nacional foi realizada em novembro de 2005 e entre estes meses pode ter ocorrido a inserção de novos profissionais; Raible (2005) utilizou o site americano Monster.com, que é pago, enquanto que eu utilizei o site americano Jobs.net, que possibilita a visualização do número de currículos que satisfazem o critério de busca gratuitamente.

4.5.6. Resultado da Análise

Ao analisar os dados, nota-se que Struts liderou todas as categorias exceto na categoria de ferramentas compatíveis. JSF, que liderou esta categoria, ficou em

segundo lugar em todas as outras exceto na categoria de disponibilidade de suporte, onde foi terceiro.

É importante ressaltar que o Struts é desenvolvido desde 2000 e já se encontra num estado de maturidade elevado. Boa parte de seus bons resultados podem ser atribuídos a este fato. Por outro lado, JSF teve sua primeira versão lançada em 2004 e mesmo assim, assumiu uma boa colocação em todas as análises, mostrando crescimento bastante acentuado no número de posições disponíveis requerendo profissionais com habilidade neste web framework.

4.6. Conclusão

O objetivo da camada de apresentação é de expor a lógica de negócios ao usuário e possibilitar a interação do usuário com a aplicação. Esta camada também costuma ser chamada de camada web no caso das aplicações baseadas na internet. Atualmente, HTML é a linguagem mais usada na construções de interfaces de aplicações disponíveis na web.

HTML oferece várias vantagens, dentre elas: HTML é executado em navegadores padrões e, portanto, não necessita que novas aplicações sejam instalados na máquina cliente; desacoplam o cliente da tecnologia usado no servidor; costumam trafegar livremente por firewalls e impõem restrições de configurações modestas já que a maior parte do processamento ocorre no lado do servidor.

Contudo também são impostos uma série de desafios, dentre eles: a interface com o usuário muda sem que a lógica de negócios necessariamente mude (D1); o modelo de requisição e resposta impede que a camada de apresentação seja notificada de mudanças no modelo (D2); é necessário separar o código de layout estático do código gerado dinamicamente (D3); requisições HTTP só carregam parâmetros do tipo String que precisam ser convertidos em tipos mais específicos da aplicação (D4) e validados (D5); possui um conjunto limitado de componentes (D6); difícil garantir que a aplicação terá o mesmo visual em todos os navegadores (D7); questões de desempenho e concorrência tornam-se mais críticas, pois a aplicação está sujeita a um maior número de usuários (D8) e são difíceis de testar (D9).

A divisão em camadas, com uma camada de apresentação limpa e magra, contribui para solução dos desafios D1, D3, D8 e D9. Para que a camada de apresentação seja limpa, usa-se o padrão de arquitetura MVC. Para que seja magra, é preciso disciplinar os desenvolvedores e realizar auditorias periódicas no código.

O padrão de arquitetura MVC divide os elementos da camada de apresentação em visão (*View*), que recebe a entrada do usuário e exibe o resultado da operação, Controlador (*Controller*), que acessa a camada de negócios manipulando o modelo e selecionando a visão apropriada, e o Modelo (*Model*), objeto representando parte do domínio e que provê os dados para a visão. Apesar de ser possível construir uma solução ad hoc, há vários frameworks, denominados web frameworks, que se propõe a prover uma solução MVC e a resolver alguns dos desafios decorrentes do uso de HTML.

Há cerca de 54 web frameworks gratuitos disponíveis para a plataforma Java. Estes frameworks implementam o MVC e oferecem meios para superar os desafios D4 e D5. Dentre estes. Foram escolhidos o Struts, por ser mais popular, o Spring MVC, pois o Spring já foi selecionado para a camada de negócios, e o JavaServer Faces, que fornece um padrão para desenvolvimento de componentes de interface.

Ao analisar tecnicamente os web frameworks, conclui-se que o Struts apresenta um elevado grau de intrusão, pois as ações e os *Action Forms* precisam estender classes do framework e as páginas devem ser construídas utilizando tags do Struts que emulam as tags do HTML. Este elevado grau de intrusão torna difícil uma possível migração do Struts para outro web framework, caso seja necessário. Por outro lado, seu esquema de validação é o melhor, possibilitando que a validação dos dados de entrada seja executada no lado do cliente e do servidor a partir de um conjunto de regras declarativas. Validações no lado do cliente ajudam a reduzir o trafego entre o navegador e o servidor, mas podem ser desabilitadas pelo usuário e, portanto, validações no lado do servidor também são importantes.

Já o Spring MVC possui o pior módulo de validação entre os três analisados, exigindo que as regras sejam programadas ao invés de declaradas. Além disso, ocorre validação apenas no lado do servidor. Entretanto Spring MVC é o menos intrusivo dos web frameworks analisados. Seus beans de formulários

não precisam estender nenhuma classe específica, seus controladores estendem classes conforme o tipo de operação a ser realizada (submissão de formulário, conjunto de telas estilo wizard, etc.) e não precisam instanciar diretamente o *Façade*, pois eles são configurados através de *Dependency Injection*. As páginas são compostas com as próprias tags HTML, diminuindo o acoplamento das páginas com Spring MVC.

O JSF por outro lado é mais intrusivo que o Spring MVC, porém menos intrusivo que o Struts. Seus *backing beans* não precisam estender classes específicas, mas eventualmente precisam obter uma referência do contexto faces (FacesContext) o que acopla o *backing bean* ao JSF. As visões são construídas utilizando componentes de interface JSF. O principal diferencial é que o JSF define um modelo de componentes reusáveis que pode ser estendido (desafio D6 da seção 4.1). Além disso, por ser um padrão aberto, a tendência é que surjam componentes de terceiros que possam ser reaproveitados, tendência que já se confirma com o surgimento de projetos como o MyFaces (2005) e WebGalileo Faces (2005) gratuitos e WebCharts (2005) comercial.

A análise não-técnica indica que Struts é o web framework mais aceito pela comunidade, mas o JSF também tem boa aceitação, sendo o web framework com mais ferramentas compatíveis disponíveis no mercado e o web framework cujo número de empregos mais cresceu no mercado norte-americano segundo as pesquisas realizadas.

Após concluir a análise técnica e a não-técnica, julga-se que o JSF é o web framework mais adequado para ser usado na camada de apresentação do AulaNet 3.0. O critério decisivo para a efetivação desta escolha é técnico: apenas o JSF define um modelo de componentes. A criação de componentes de interfaces para o AulaNet 3.0 auxiliará na padronização das interfaces e contribuirá para o desenvolvimento rápido de interfaces. Além disso, podem ser aproveitados componentes desenvolvidos por terceiros, sejam estes desenvolvidos especificamente para o AulaNet ou não.

Ao considerar os desafios listados no início deste capítulo, constata-se que a abordagem de divisão em camadas, com uma camada de apresentação limpa e magra, contribuiu para a superação dos desafios D1, D3, D8 e D9. A utilização de um web framework baseado em componentes, o JSF, contribuiu para a superação dos desafios D4, D5 e D6. O desafio D2 não pode ser resolvido, pois é intrínseco

ao modelo de requisição e resposta utilizado nas aplicações web. Já o desafio D7 não é solucionado pelo AulaNet 3.0, pois o custo para obter uma interface padronizada em todos os navegadores é muito alto e implica em muitos testes em diferentes navegadores. Como o navegador Internet Explorer detém 85,31% de participação no mercado, segundo pesquisa realizada em janeiro de 2006 (MarketShare, 2006), os testes e desenvolvimentos no AulaNet 3 serão voltados em um primeiro momento para este navegador.