

**LIGA DE ENSINO DO RIO GRANDE DO NORTE
FACULDADE NATALENSE PARA O DESENVOLVIMENTO
DO RIO GRANDE DO NORTE**

**ESTUDO DE PRODUTIVIDADE DO
DESENVOLVIMENTO DE APLICAÇÕES BASEADAS NOS
FRAMEWORKS APACHE STRUTS E JAV
ANDRÉ MEDEIROS DANTAS**

**ESTUDO DE PRODUTIVIDADE DO
DESENVOLVIMENTO DE APLICAÇÕES BASEADAS NOS
FRAMEWORKS APACHE STRUTS E JAVASERVER FACES**

NATAL/RN

2008

ANDRÉ MEDEIROS DANTAS

**ESTUDO DE PRODUTIVIDADE DO
DESENVOLVIMENTO DE APLICAÇÕES BASEADAS NOS
FRAMEWORKS APACHE STRUTS E JAVASERVER FACES**

Monografia apresentada ao Curso de Especialização em Desenvolvimento de Sistemas Corporativos da Faculdade Natalense para o desenvolvimento do Rio Grande do Norte como requisito obrigatório para obtenção do título de Especialista em Sistemas Corporativos.

Orientador:

Gleydson de A. F. Lima, Msc.

NATAL/RN

2008

ANDRÉ MEDEIROS DANTAS

ESTUDO DE PRODUTIVIDADE DO
DESENVOLVIMENTO DE APLICAÇÕES BASEADAS NOS
FRAMEWORKS APACHE STRUTS E JAVASERVER FACES

Catálogo na Publicação – Biblioteca da FARN
Setor de Processos Técnicos

Dantas, André Medeiros

Estudo de produtividade do desenvolvimento de aplicações baseadas
nos Frameworks Apache Struts e JavaServer Faces / André Medeiros
Dantas. – Natal: FARN, 2008.

85f.

Orientador: Gleydson de A. F. Lima (M.Sc.)

Monografia (Especialização em Desenvolvimento de Sistemas
Corporativos) – Faculdade Natalense para o Desenvolvimento do Rio
Grande do Norte.

1. Desenvolvimento de Softwares Web – Monografia. 2. Tecnologias
de Aplicação Web – Monografia. 3. Frameworks Web – Monografia. 4.
JavaServer Faces – Monografia. Faces I. Título.

RN/FARN/BC

CDU 004.4: 004.738.5



004.4:004.738.5

D21 e

2008

ex 01

ANDRÉ MEDEIROS DANTAS

Um dos frameworks para as camadas do Controlador e Visão do perfil arquitetural MVC aplicado para aplicações Web se tornou bastante popular ao longo dos anos. Estes são classificados em "Orientados a Ações" ou "Orientados a Componentes", de acordo com a estratégia de solução adotada pelas ferramentas. De acordo com a estratégia de desenvolvimento e produtividade do desenvolvimento das aplicações pode ser afetada, bem como a razão dos recursos disponíveis e forma com que estas aplicações são desenvolvidas devido a limitações e características do framework escolhido. O objetivo desta pesquisa consiste em analisar o quanto a produtividade pode ser impactada de acordo com a escolha de um tipo de framework Web. Com esse intuito, foram realizadas análises de caso através da implementação de algumas aplicações de exemplo acadêmico se utilizando diferentes versões de frameworks Web. Para isso, foram feitas as seguintes representações: Frameworks Web, Struts, JavaServer Faces, Produtividade de Desenvolvimento.

ESTUDO DE PRODUTIVIDADE DO DESENVOLVIMENTO DE APLICAÇÕES BASEADAS NOS FRAMEWORKS APACHE STRUTS E JAVASERVER FACES

Monografia apresentada ao Curso de Especialização em Desenvolvimento de Sistemas Corporativos da Faculdade Natalense para o Desenvolvimento do Rio Grande do Norte, aprovada em ____ de ____ de ____, pela banca examinadora composta pelos seguintes membros:

Gleydson de Azevedo F. Lima

Gleydson Lima, Msc.

Orientador

Membro

Examinador

RESUMO

O uso de frameworks para as camadas do Controlador e Visão do padrão arquitetural MVC adaptado para aplicações Web se tornou bastante popular ao longo dos anos. Eles são classificados em “Orientados a Ações” ou “Orientados a Componentes”, de acordo com a estratégia de solução adotada pelas ferramentas. De acordo com a estratégia desses tipos de frameworks a produtividade do desenvolvimento das aplicações pode ser afetada. Isso pode acontecer em razão dos recursos disponíveis e forma com que essas aplicações são desenvolvidas devido a limitações e características do framework escolhido. O trabalho desta monografia consiste em analisar o quanto a produtividade pode ser impactada de acordo com a utilização de um tipo de framework Web. Com esse intuito, foram realizados estudos de caso através da implementação de pequenas aplicações de controle acadêmico se utilizando das mais recentes versões dos frameworks Apache Struts e JavaServer Faces, os principais representantes de frameworks Web da plataforma Java. Para essa avaliação, foram definidos problemas e funcionalidades típicas desse tipo de aplicação que cada um dos estudos de caso deve atender e solucionar. A partir da análise de como cada framework permite a implementação desses aspectos identificados, foi possível comparar as vantagens de cada um dos frameworks com relação à produtividade do desenvolvimento.

Key words: Web Frameworks, Struts, JavaServer Faces, Software Development Productivity

Palavras-chave: *Frameworks Web, Struts, JavaServer Faces, Produtividade de Desenvolvimento*

LISTA DE ABSTRACTS

Over the years the use of application frameworks designed for the View and Controller layers of MVC architectural pattern adapted to web applications has become very popular. These frameworks are classified into “Actions Oriented” and “Components Oriented”, according to the solution strategy adopted by the tools. According to the strategy of such frameworks the applications development productivity can be affected. This can happen because of the available resources and the way the applications are developed due to the chosen framework limitations characteristics. The work of this document consists to examine how the productivity may be impacted according to the use of such Web frameworks. To accomplish this, small academic management applications were developed using the latest versions of Apache Struts and JavaServer Faces frameworks, the main representatives of Java platform Web frameworks of. For this assessment were defined problems and typical features of this type of application that each of the case studies must meet and resolve. From the analysis of how each framework allows the implementation of these aspects identified, it was possible to compare the benefits of each framework with regard to the development productivity.

Capítulo 2: Tecnologias Java para implementar o padrão MVC-2	23
Capítulo 3: Padrão MVC usando tecnologias Java	24
Capítulo 4: Aplicações Multicamadas	26
Key words: Web Frameworks, Struts, JavaServer Faces, Software Development Productivity	
Capítulo 5: Visão geral dos serviços providos por frameworks Web	31
Capítulo 6: Abstrações de serviços providos pelas tecnologias para aplicações Web	32
Capítulo 7: Arquitetura do Struts2	35
Capítulo 8: Ciclo de vida de uma requisição no Struts2	37
Capítulo 9: Tela do NetBeans 5.5 com suporte ao RAD de aplicações Web com JSF	41
Capítulo 10: Visão geral da arquitetura do JSF	44
Capítulo 11: Ciclo de vida de uma requisição no JSF	45
Capítulo 12: Diagrama de Casos de Uso dos experimentos	51
Capítulo 13: Modelo de Domínio dos experimentos	52
Capítulo 14: Camadas de negócio e acesso a dados	55
Capítulo 15: Componentes da camada de acesso aos dados	57
Capítulo 16: Componentes da camada de negócio	58
Capítulo 17: Camada de dados compartilhada pelas implementações das duas aplicações	60
Capítulo 18: Componentes implementados da aplicação em Struts	61
Capítulo 19: Componentes implementados da aplicação em JSF	63
Capítulo 20: Trecho do código do formulário JSP usando Struts para cadastro de disciplina	64
Capítulo 21: Trecho do código do formulário JSP usando JSF para cadastro de disciplina	65
Capítulo 22: Trecho do código usando Struts para carregar arquivo	66
Capítulo 23: Trecho do código da formatação de uma tabela em Struts	68

LISTA DE FIGURAS

Figura 1. Arquitetura em camadas	14
Figura 2. Camadas de arquitetura de sistemas de informação.....	14
Figura 3. Representação abstrata do padrão arquitetural Modelo-Visão-Controlador.....	16
Figura 4. Comportamento das camadas do MVC	17
Figura 5. Esquema de acesso a conteúdo estático	19
Figura 6. Esquema de acesso a conteúdo dinâmico através de scripts CGI.....	20
Figura 7. Esquema de acesso a conteúdo dinâmico através de Servlets	21
Figura 8. Funcionamento das Java Server Pages	22
Figura 9. Tecnologias Java para implementar o padrão MVC-2	23
Figura 10. Padrão MVC usando tecnologias Java.....	24
Figura 11. Aplicações Multicamadas.	26
Figura 12. APIs do Java EE 5 em diferentes <i>containers</i>	26
Figura 13. Visão geral dos serviços providos por <i>frameworks</i> Web	31
Figura 14. Abstrações de serviços providos pelas tecnologias para aplicações Web.	32
Figura 15. Arquitetura do Struts2.	35
Figura 16. Ciclo de vida de uma requisição no Struts2.....	37
Figura 17. Tela do NetBeans 5.5 com suporte ao RAD de aplicações Web com JSF	41
Figura 18. Visão geral da arquitetura do JSF	44
Figura 19. Ciclo de vida de uma requisição no JSF	45
Figura 20. Diagrama de Casos de Uso dos experimentos	51
Figura 21. Modelo de Domínio dos experimentos	52
Figura 22. Camadas de negócio e acesso a dados	55
Figura 23. Componentes da camada de acesso aos dados.....	57
Figura 24. Componentes da camada de negócio	58
Figura 25. Camada de dados compartilhada pelas implementações das duas aplicações.....	60
Figura 26. Componentes implementados da aplicação em Struts.....	61
Figura 27. Componentes implementados da aplicação em JSF	62
Figura 28. Trecho do código do formulário JSP usando Struts para cadastro de disciplina....	64
Figura 29. Trecho do código do formulário JSP usando JSF para cadastro de disciplina	65
Figura 30. Trecho de código usando Struts para carregar arquivo.....	66
Figura 31. Trecho de código da formatação de uma tabela em Struts	68

SUMÁRIO

1 INTRODUÇÃO	8
1.1 OBJETIVOS DO TRABALHO	9
1.2 ESTRUTURA DO DOCUMENTO	11
2 TECNOLOGIAS DE DESENVOLVIMENTO DE APLICAÇÕES WEB.....	12
2.1 APLICAÇÕES WEB	12
2.2 ARQUITETURA DE SOFTWARE EM CAMADAS.....	13
2.2.1 O Padrão MVC.....	15
2.3 TECNOLOGIAS	18
2.3.1 Java EE – Edição Corporativa da Plataforma Java.....	24
3 FRAMEWORKS WEB	30
3.1 FRAMEWORKS WEB	30
3.2 APACHE STRUTS	33
3.2.1 Componentes.....	35
3.2.2 Principais Características.....	37
3.3 JAVASERVER FACES	40
3.3.1 Conceitos Chave.....	43
3.3.2 Arquitetura	43
3.3.1 Ciclo de Vida	45
4 APRESENTAÇÃO DA AVALIAÇÃO	47
4.1 METODOLOGIA	47
4.1.1 Problemas e Funcionalidades	48
4.2 EXPERIMENTOS	50
5 ESTUDOS DE CASO	54
5.1 ARQUITETURA E CAMADA DE MODELO	54
5.1.1 Camada de Acesso a Dados	56
5.1.2 Camada de Negócio	58
5.1.3 Camada de Modelo das Aplicações	59
5.2 IMPLEMENTAÇÃO EM STRUTS	60
5.3 IMPLEMENTAÇÃO EM JSF	61
6 AVALIAÇÃO DE PRODUTIVIDADE	63
6.1 OPERAÇÕES DE CADASTRO	63
6.2 UPLOAD DE ARQUIVO	65
6.3 PAGINAÇÃO E CLASSIFICAÇÃO DE TABELAS	67
6.4 OPEN SESSION IN VIEW.....	69
7 CONCLUSÕES.....	71
REFERÊNCIAS.....	74
ANEXOS.....	77

1 INTRODUÇÃO

Aplicações que fornecem serviços pela Internet são cada vez mais comuns. Muitas vezes são serviços simples, com forte apelo comercial e não exigem conhecimento técnico avançado. Entretanto, aplicações Web também são usadas para grandes sistemas, principalmente em ambientes corporativos.

As tecnologias de desenvolvimento Web evoluíram bastante desde o início das primeiras aplicações dinâmicas. De tempos em tempos surgem tecnologias que prometem ainda mais facilidade no desenvolvimento e resultados mais rápidos. Entretanto, quando se é preciso desenvolver sistemas de grande porte, com restrições de qualidade rígidas, regras de negócio complexas e um longo período de evolução e manutenção, a escolha da plataforma de desenvolvimento geralmente é feita pelas tecnologias mais maduras e com maiores recursos.

Plataformas mais consagradas possuem um número maior de colaboradores, reunidos como grupos de desenvolvedores ou de empresas que se beneficiam com o sucesso da tecnologia. Isso conseqüentemente reflete na quantidade de recursos disponíveis, distribuídos de forma gratuita ou comercial. Esses recursos podem ser encontrados na forma de ferramentas, documentação, listas de discussão, etc.

O desenvolvedor de sistemas corporativos dessas plataformas se aproveita bastante de todos esses recursos, principalmente das ferramentas e das “soluções já prontas” que são uma ótima forma de evitar “reinventar a roda” (FOWLER, 2003). É inevitável na elaboração de arquiteturas de grandes sistemas ser necessário recorrer a soluções já implementadas de problemas comuns, principalmente nas camadas de mais baixo nível.

Uma das formas para reutilizar soluções para problemas comuns é a utilização de *frameworks*. *Framework* de aplicação é um conjunto de artefatos que se complementam e solucionam um problema comum de maneira genérica (PRESSMAN, 2006). Ele deve disponibilizar pontos de configuração que possibilitem a customização do comportamento de seus recursos. Os artefatos normalmente fornecidos são algumas bibliotecas com a implementação das funcionalidades básicas, modelos de arquivos de configuração para a personalização dos seus serviços e a documentação para guiar os desenvolvedores no uso da tecnologia.

Em sistemas de grande porte, diversos *frameworks* podem compor uma arquitetura

sendo responsáveis pelos serviços básicos de muita importância como persistência de dados, controle de transacionalidade das operações, controle de processamento distribuído e criação de componentes de *interface* gráfica. A utilização de soluções “semi-prontas” traz grandes vantagens para o desenvolvimento desses sistemas, como permitir que os desenvolvedores possam se concentrar na implementação de componentes de mais alto nível e diminuir todo o custo que envolve o desenvolvimento da infra-estrutura do sistema.

Existem muitos tipos de *frameworks* de aplicação com objetivos e funcionalidades distintas. No desenvolvimento de aplicações Web, os *frameworks* que auxiliam na infra-estrutura responsável pela manipulação das requisições HTTP e a formatação e geração do conteúdo respondido às máquinas clientes podem ser chamados de *frameworks* Web. Na plataforma Java existem diversos *frameworks* desse tipo, distribuídos gratuita e comercialmente.

Atualmente os *frameworks* Apache Struts e JavaServer Faces (JSF) são os dois *frameworks* Web mais cotados a serem utilizados em projetos orientados a objetos para grandes sistemas Web baseados na plataforma Java. O Struts, baseado no tratamento de ações de formulários de páginas HTML, ainda hoje domina o mercado com sua primeira e mais popular versão (1.2) por ter sido uma das primeiras implementações que atendia aos principais requisitos para esse tipo de aplicação. Por outro lado o JSF, lançado no fim de 2004, é hoje anunciado como a API oficial da plataforma JavaEE e tem como principais vantagens seu suporte a ferramentas de desenvolvimento RAD e o gerenciamento de eventos de todos os componentes da *interface* do usuário exibidos do lado cliente da aplicação.

A utilização de *frameworks* de aplicação sem dúvida é uma prática muito importante incrementar a produtividade do desenvolvimento, principalmente de aplicações de sistemas de grande porte. Por outro lado, os recursos oferecidos por esses *frameworks* e forma com que a aplicações acabam por ser desenvolvida, devido as vezes ao uso extenso de arquivos de configuração também pode comprometer a produtividade do desenvolvimento.

1.1 OBJETIVOS DO TRABALHO

Frameworks para desenvolvimento de aplicações Web têm se tornado populares principalmente pelo ganho de produtividade devido à disponibilização de uma infra-estrutura semi-pronta que dá suporte à adaptação do padrão MVC para aplicações Web. Como será discutido no Capítulo 3, existem estratégias diferentes de implementar esses *frameworks*

Web: orientados a ação (representados nesse trabalho pelo Struts) e a componentes (representados pelo JavaServer Faces). Em ambos os tipos, os recursos são direcionados a aspectos diferentes do desenvolvimento aos objetivos distintos dos frameworks utilizados. As diferenças de como esses frameworks fornecem subsídios para solucionar os principais problemas e funcionalidades de aplicações Web podem influenciar na produtividade do desenvolvimento das mesmas.

O trabalho apresentado nessa monografia teve como principal objetivo avaliar qual o impacto na produtividade do desenvolvimento dos componentes construídos em aplicações baseadas nesses dois *frameworks* Web citados. Em tempos de altas competições no mercado quanto maior a produtividade do desenvolvimento disponibilizando sistemas de qualidade com maior agilidade, maiores são as chances de sucesso do negócio envolvido.

Os *frameworks* Struts e JSF possuem poucas avaliações mais profundas e sistemáticas, principalmente no que diz respeito às versões mais recentes de cada um deles. As comparações se limitam evidenciar as diferenças óbvias entre eles, como a biblioteca de componentes de GUI de um ou o número de recursos para o controle da aplicação do outro. A decisão de projetistas e arquitetos de software precisa ser fundamentada em análises mais valorosas.

A diferença entre os *frameworks* vai além dos objetivos tipicamente distintos de um se dedicar à disponibilização de recursos para a camada do controlador e outro para a camada de visão através da componentização dos objetos de GUI. Na verdade, seus modos de operação são tão diferentes que induzem o desenvolvedor a criar e relacionar seus componentes de forma diferente. A intenção da avaliação desse trabalho é identificar o impacto na produtividade de desenvolvimento da forma de desenvolver em cada um dos *frameworks* escolhidos.

Até a presente data, o autor desta monografia desconhece trabalhos acadêmicos que dêem ênfase na avaliação de produtividade dos frameworks Web escolhidos. Análises comparativas entre *frameworks* desse tipo costumam ser superficiais, examinando apenas funcionalidades e conceitos obviamente distintos entre eles. Um exemplo disso é restringir suas diferenças ao uso de componentes do controlador ou da visão.

Avaliações mais completas e profundas poderiam fornecer informações mais importantes aos profissionais, desde projetistas, arquitetos de software e programadores. Melhor informados, importantes decisões como a escolha do *framework* para novos projetos

poderiam ser realizadas com mais embasamento, dessa forma diminuindo possíveis riscos dos projetos.

A produtividade do desenvolvimento de aplicações baseadas nesses *frameworks* é um dos aspectos importantes dessa análise comparativa mais complexa. Outros aspectos interessantes que podem auxiliar no processo de decisão de escolha entre os *frameworks* são: dificuldade de aprendizado para o uso das tecnologias, suporte da comunidade através de ferramentas e documentação e demanda do mercado de trabalho, bem como a análise de atributos de qualidade das aplicações como portabilidade, confiabilidade, manutenibilidade, entre outros.

1.2 ESTRUTURA DO DOCUMENTO

Este documento apresenta o desenvolvimento e as conclusões da avaliação realizada, dividindo seu conteúdo em três segmentos:

- **A fundamentação teórica sobre aplicações Web:** No segundo é apresentado um breve histórico de como as tecnologias para essas aplicações evoluíram e uma visão geral sobre a plataforma de desenvolvimento usada no desenvolvimento das aplicações de experimento. No Capítulo 3, o foco é apresentar e descrever os *frameworks* Web escolhidos e evidenciar suas diferenças conceituais como implementação do padrão MVC Modelo-2;
- **Apresentação da avaliação e experimentos:** No quarto capítulo são apresentados em detalhes os objetivos do que a avaliação deve abranger e a metodologia aplicada para sua realização. Em seguida, no Capítulo 5, as aplicações desenvolvidas com cada *framework* são descritas, e detalhes sobre as implementações são representados através de diagramas de classes;
- **Resultados e conclusões:** No sexto capítulo é apresentada a comparação da produtividade com base nos critérios definidos. Finalmente no último capítulo são relatadas as considerações finais sobre a avaliação realizada e conclusões sobre o impacto sofrido na produtividade do desenvolvimento das aplicações a partir do uso de cada *framework*.

2 TECNOLOGIAS DE DESENVOLVIMENTO DE APLICAÇÕES WEB

Aplicações Web são baseadas na comunicação remota entre máquinas através do protocolo HTTP. A partir dessa arquitetura distribuída, na qual as informações ficam centralizadas em um servidor, é possível disponibilizar serviços que podem atingir a abrangência de toda Internet. Entretanto, essa abordagem possui desvantagens, como a dependência da qualidade da transmissão remota desses dados, que precisam ser levadas em consideração pelas tecnologias que desenvolvem essas aplicações. As tecnologias de desenvolvimento estão concentradas em disponibilizar recursos do lado do servidor com o intuito de aproveitar os benefícios e contornar as desvantagens intrínsecas a esse tipo de comunicação.

Esse capítulo é dedicado a apresentar algumas dessas principais tecnologias, sobretudo as da plataforma Java, na qual estão incluídas as ferramentas de estudo do trabalho. A popularização do uso de *frameworks* baseados no padrão MVC passou a uniformizar arquiteturas de aplicações Web. O capítulo inicia com uma visão geral sobre aplicações Web. No segundo tópico, explica a origem do padrão MVC para o desenvolvimento dessas aplicações. Em seguida, é realizado um breve histórico de como aplicações Web deixaram de ser repositório de documentos estáticos para sistemas dinâmicos robustos.

2.1 APLICAÇÕES WEB

O desenvolvimento das tecnologias tem ajudado cada vez mais a gerenciar o crescente volume de informações em todas as áreas de atuação. Em alguns sistemas de informação, existem processos imprescindíveis, porém que se tornam consideravelmente tediosos e trabalhosos se a execução for realizada de forma manual. Realizados dessa maneira, certamente são responsáveis por significantes perdas de tempo e eficiência.

A informatização desses processos através de sistemas computadorizados tem se tornado a melhor opção nos dias de hoje. Devido às inúmeras facilidades (como portabilidade, pouco custo de implantação, entre outras), os sistemas de informação estão se voltando cada vez mais ao uso da tecnologia Web. A partir dela, podem prover serviços acessados por um navegador da Internet e serem executados apenas em um local, o servidor. Isso torna as aplicações Web uma das melhores soluções para automatização de importantes processos.

Esse tipo de aplicação baseia-se em arquitetura de sistemas distribuídos, distinguindo,

principalmente ao desenvolvedor, as responsabilidades da *interface* gráfica apresentada ao usuário (páginas HTML interpretadas pelos navegadores no lado cliente) da estrutura e processamento do sistema em si (através da execução de scripts ou programas no lado servidor).

Um dos grandes motivos para a popularização dessas aplicações é a simples portabilidade no lado do cliente, onde não é preciso mais que um navegador em qualquer sistema operacional para interpretar as respostas do servidor. Com o avanço na maneira de representar o sistema no lado do servidor, as aplicações web estão abrangendo sistemas de portes cada vez maiores. Simples páginas informativas sobre uma empresa se tornaram *sites* dinâmicos que passaram a gerar conteúdo automaticamente com maior interatividade com os usuários, e progressivamente se tornaram concentradores de informações chegando a serem considerados verdadeiros portais de informações sobre uma determinada área de conhecimento.

2.2 ARQUITETURA DE SOFTWARE EM CAMADAS

Na literatura sobre arquitetura de software em camadas, discutido em (PRESSMAN, 2006; SOMMERVILLE, 2006), foram identificados conjuntos de sistemas com comportamentos e características similares que ajudaram a classificar as arquiteturas em estilos arquiteturais. Essa classificação permite aos desenvolvedores com maior clareza na hora de vislumbrarem possíveis soluções para o problema de determinadas aplicações. Segundo Pressman, cinco estilos arquiteturais são identificados: arquitetura centrada nos dados, de fluxos de dados, de chamada e retorno, orientada a objetos e arquitetura em camadas.

O estilo arquitetural em camadas é um dos mais comumente encontrado, principalmente em arquiteturas de sistemas de informação de grande porte. Nele a arquitetura é dividida em camadas independentes com objetivos distintos e que só se comunicam com as camadas imediatamente superiores e inferiores, como ilustrado na Figura 1 (MENDES, 2002). Projetar e implementar uma arquitetura de software baseada em camadas representa utilizar um estilo arquitetural que preza pela separação de interesses dos seus componentes, um dos principais princípios de projetos orientados a objetos.

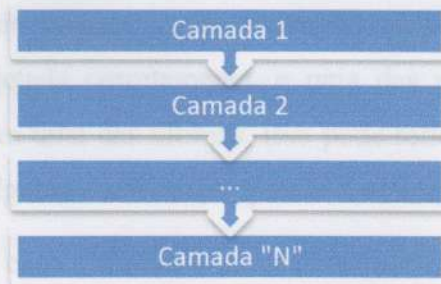


Figura 1. Arquitetura em camadas
Fonte: Dados da Pesquisa (2007)

Em arquiteturas dessa natureza, componentes são agrupados de acordo com suas funcionalidades e responsabilidades dentro da estrutura, formando camadas distintas que se comunicam. Com essa organização, é possível abstrair detalhes das camadas e analisar cada uma separadamente como pode ser observado na Figura 2. Esta figura representa a separação de interesses em um sistema de informação clássico com funcionalidades para a apresentação dos dados separadas da forma como eles são acessados e a que regras (específicas do domínio da aplicação) esses dados são submetidos.

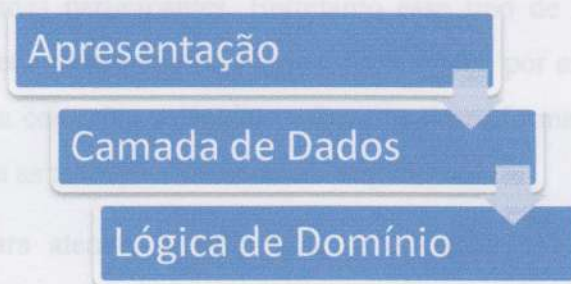


Figura 2. Camadas de arquitetura de sistemas de informação
Fonte: Dados da Pesquisa (2007)

Devido a essa separação, esse tipo de arquitetura é bastante recomendado para processos de desenvolvimento incrementais. Os desenvolvedores podem concentrar esforços em qualquer camada sem se preocupar com a implementação das demais, apenas voltado para atender aos requisitos da *interface* com a camada imediatamente superior. Além disso, a arquitetura fica melhor preparada para aceitar as inevitáveis mudanças de requisitos do sistema, e torna mais simples a manutenção e aprimoramento de cada camada separadamente.

Entretanto, alguns aspectos podem ser prejudicados devido às características inerentes ao estilo. Os principais aspectos são:

- Desempenho: a adição de camadas afeta diretamente no desempenho da arquitetura. Quanto mais camadas existirem, mais processamento para a comunicação entre elas;

- Modificações em cascata: caso as dependências entre todas as camadas não tenha sido projetada corretamente, e uma das primeiras camadas tenha uma dependência com uma das últimas, isso pode refletir em alterações por todas as camadas caso alguma modificação precise ser feita.
- Complexidade: cada camada a ser adicionada na arquitetura precisa ser bem analisada, pois o entendimento sobre seu funcionamento pode ser prejudicado se aumentar o número de camadas desnecessariamente;

Como já mencionado anteriormente, um típico exemplo de uma arquitetura que agrupa componentes de acordo com suas responsabilidades são sistemas de informação de grande porte. Isso porque esses sistemas são quase sempre organizados da mesma maneira: uma *interface* gráfica para interação com o usuário e exibição dos dados manipulados pelo sistema e, nos “bastidores”, componentes de software responsáveis por implementarem as regras de negócio e acesso a uma base de dados centralizada. A partir dessa descrição é possível identificar as três camadas participantes. Entretanto esse tipo de arquitetura não pode ser considerado uma arquitetura em camadas porque é inevitável, por exemplo, que a camada de apresentação não possua certa dependência do formato das informações na base de dados, e isso obviamente infringe as restrições desse estilo arquitetural.

Dessa forma para atender a essa maneira diferente de utilizar a arquitetura em camadas, é necessária fazer adaptações a essas restrições. As adaptações de um estilo arquitetural que identificam outra solução parecida com a do estilo é denominada de Padrão Arquitetural (PRESSMAN, 2006). O padrão Model-View-Controller (MVC) é uma solução para adaptar o uso de sistemas de informação às arquiteturas que trabalham com as já mencionadas três camadas (CHRISTENSEN, 2004). Ele trabalha com os mesmos conceitos de separação de interesses em camadas diferentes, mas garante maior flexibilidade para o relacionamento entre elas. Esse padrão é especialmente importante para esse trabalho porque, atualmente, a implementação dos *frameworks* para desenvolvimento de aplicações Web é baseada nele.

2.2.1 O Padrão MVC

Como discutido anteriormente, um exemplo de aplicações que tipicamente se valem da estratégia de dividir sua arquitetura em camadas são as que baseiam sua utilização nas

interações com usuários através de uma *interface* gráfica (ou GUI). Normalmente, uma forma simplificada de arquitetura em camadas é elaborada e a aplicação é separada em três camadas distintas: 1) a visão, para a formatação e exibição das informações na GUI; 2) o modelo responsável pela manutenção dessas informações da aplicação; e 3) o controlador que determina exatamente o comportamento da GUI de acordo com os dados do modelo. Essa solução tornou-se um padrão arquitetural para esse tipo de aplicação chamado MVC (*Model-View-Controller*, do inglês Modelo-Visão-Controlador) (MENDES, 2002).

Segundo Dudney et al. (2004), algumas das principais motivações para a aplicação desse padrão são:

- Necessidade de representar as mesmas informações de diferentes maneiras;
- Eventos acionados pelos usuários devem atualizados os dados da aplicação e imediatamente refletir em todas as suas visualizações;
- Utilização de componentes re-usáveis independentes dos dados da aplicação.

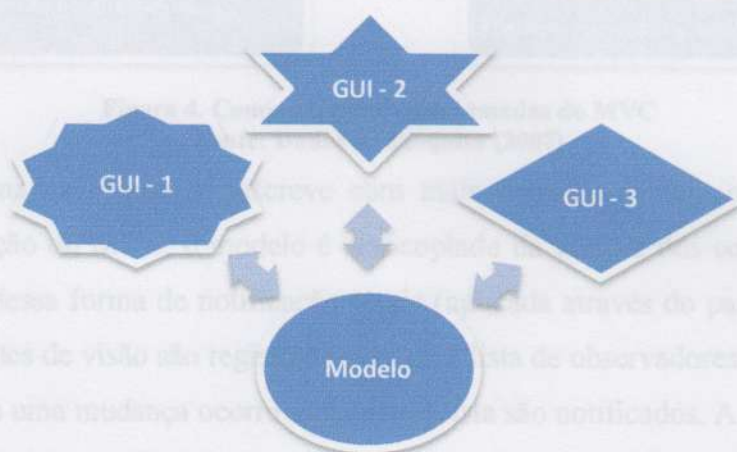


Figura 3. Representação abstrata do padrão arquitetural Modelo-Visão-Controlador
Fonte: Dados da Pesquisa (2007)

O comportamento do MVC é ilustrado genericamente na Figura 3 em que diferentes formas de *interface* com usuário (GUI-1, GUI-2 e GUI-3) exibem o mesmo conteúdo representado pelo Modelo. Nessa arquitetura, o modelo pode ser alterado por ações de usuários de qualquer uma das *interfaces*, e ela deve garantir que todas as outras *interfaces* apresentem esses dados atualizados. Para assegurar essa atualização constante, a comunicação das camadas dá-se da seguinte maneira: assim que o modelo é alterado por uma ação de alguma das *interfaces*, ele notifica todas as *interfaces* dessa alteração. Dessa forma também é possível adicionar novos tipos de *interfaces* diferentes sem precisar alterar o modelo.

Percebe-se na Figura 3 que o controlador encontra-se omitido para exemplificar

melhor o funcionamento de forma abstrata. Ao incluí-lo nessa representação, o próprio modelo se abstrai das interfaces de usuário que o acessam. Essa tarefa passa a ser do controlador. É possível reconhecer alguns padrões de projeto clássicos identificados por Gamma et al. (1995). Os principais padrões aplicados nessa arquitetura são *Strategy*, *Composite* e *Observer*.

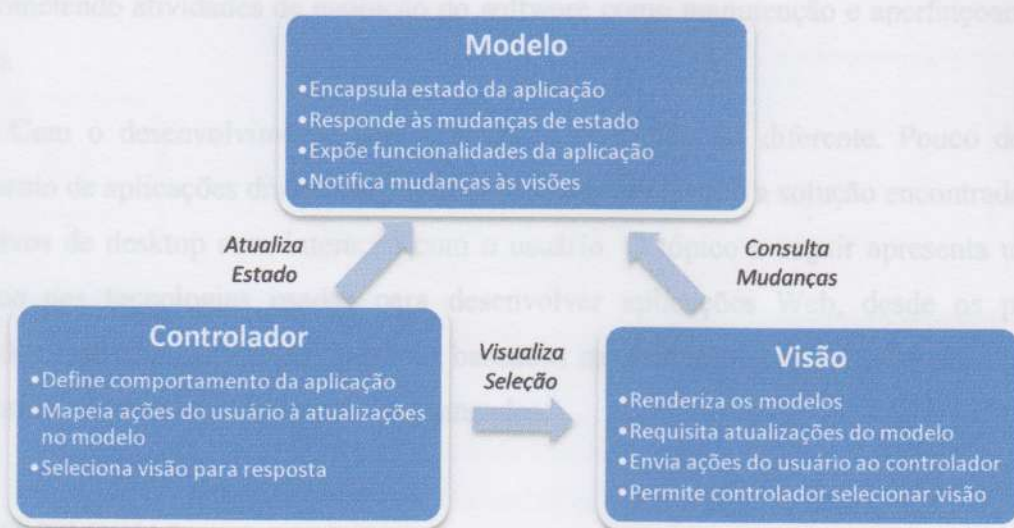


Figura 4. Comportamento das camadas do MVC
Fonte: Dados da Pesquisa (2007)

O diagrama na Figura 4 descreve com mais detalhes as funcionalidades de cada camada. Na solução do MVC, o modelo é desacoplado da visão e dos controladores. Isso é possível através dessa forma de notificação usada (aplicada através do padrão *Observer*) em que os componentes de visão são registrados em uma lista de observadores de atualizações do modelo, e quando uma mudança ocorre todas nessa lista são notificados. A partir da descrição de alguns comportamentos é possível identificar aplicações dos padrões de projeto citados:

- Através do padrão *Observer* é possível separar objetos de maneira que as mudanças ocorridas em um possam afetar um número qualquer de outros objetos, sem exigir que o objeto alterado conheça detalhes dos outros;
- Para criar componentes re-usáveis na visão, utiliza-se *Composite* para determinar uma hierarquia de componentes da GUI diferente que acessam o modelo da mesma maneira independente de como são renderizados;
- Enquanto é usando o *Strategy* que se pode definir diferentes estratégias de quais algoritmos usar para enviar diferentes respostas ao controlador.

O MVC foi criado originalmente para desenvolver aplicações com interações com usuários criadas com *Smalltalk*, uma das pioneiras linguagens de programação completamente

orientadas a objetos. Até então, objetos desenvolvidos para esse tipo de aplicação eram criados de forma monolítica, com rotinas de formatação de dados para serem exibidos na *interface* junto com procedimentos que implementavam regras específicas do negócio tratado pela aplicação. Essa situação resultava em alto acoplamento entre os componentes e dificultava a organização da própria arquitetura o que conseqüentemente acabava comprometendo atividades de evolução do software como manutenção e aperfeiçoamento do mesmo.

Com o desenvolvimento para aplicações Web não foi diferente. Pouco depois do surgimento de aplicações dinâmicas, desenvolvedores adaptaram a solução encontrada para os aplicativos de desktop com interação com o usuário. O tópico a seguir apresenta um breve histórico das tecnologias usadas para desenvolver aplicações Web, desde os primeiros conteúdos estáticos até grandes sistemas baseados na interação com o navegador de páginas se utilizando de recursos de *interfaces* avançados.

2.3 TECNOLOGIAS

Com a implantação da rede mundial de computadores, o HTML¹ (*HyperText Markup Language*) surgiu como o formato padrão de visualização de documentos, onde é permitido “navegar” entre documentos relacionados através dos chamados “*hyper-links*”. Para disponibilizar alguma informação na rede, se passou apenas a deixar o documento em um servidor Web de onde qualquer outro computador poderia ter acesso. Em pouco tempo, inúmeras organizações e instituições começaram a reunir várias páginas HTML (termo que se popularizou chamar esses documentos) contendo suas mais diversas informações para acesso público, e então nasceram os web-sites. Toda essa comunicação é baseada no protocolo HTTP (*Hypertext Transfer Protocol*), e que devido à sua natureza “sem estado” (do inglês, *stateless*) consegue manter a escalabilidade dos inúmeros documentos disponíveis atualmente nos servidores ao redor do mundo (SOARES; LEMOS; COLCHER, 1995).

Outras tecnologias surgiram para auxiliar a formatação do HTML: o CSS² (*Cascating Style Sheets*), que permite ao profissional responsável pela criação dessas páginas melhor organizar toda a formatação do conteúdo independente dos dados a serem exibidos; e ainda o

¹ <http://www.w3.org/html/>

² <http://www.w3.org/Style/CSS/>

JavaScript¹, uma linguagem de programação interpretada pelos navegadores, que inicialmente tentou suprir requisitos para dar caráter dinâmico a essas páginas. A Figura 5 ilustra o acesso de máquinas clientes a um simples servidor que retorna conteúdo HTML de acordo com os documentos solicitados pelos navegadores das máquinas clientes. A numeração indica a sequência dos eventos gerados em um acesso.

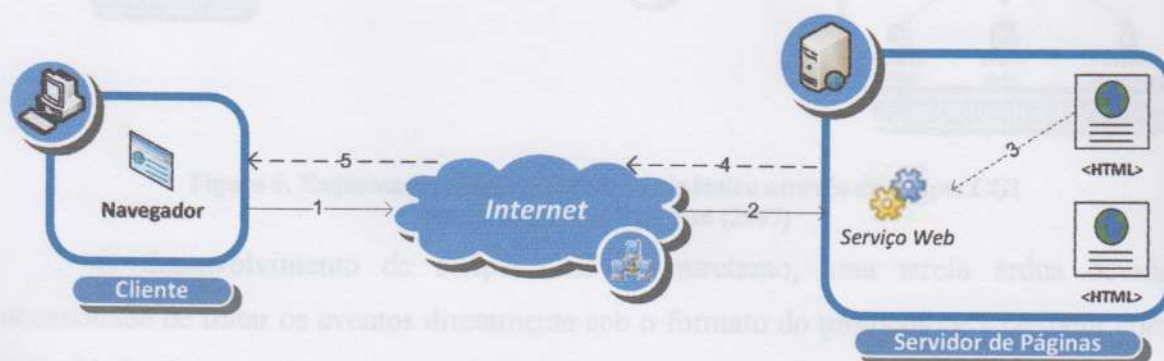


Figura 5. Esquema de acesso a conteúdo estático
Fonte: Dados da Pesquisa (2007)

A popularidade dos conteúdos disponibilizados na rede foi crescente, e com ela a necessidade de uma maior interação com os usuários que os acessavam. As páginas precisavam ser capazes, de alguma forma, de manipular dados submetidos pelo usuário para fins de armazenamento e consulta. A solução mais popular veio com o CGI (*Common Gateway Interface*) (GUNDAVARAM, 1996), tecnologia que permitia o processamento de informações vindas de formulários contidos em páginas HTML. Essa manipulação era realizada de fato no servidor web, onde as páginas e os scripts CGI ficam armazenados. Seu funcionamento, como ilustrado na Figura 6, é baseado na execução de programas (elaborados com linguagens de programação, como Goerzen (2004), Castro (1998), Choi (1998) e Felton (1997) entre outras, que se associam ao CGI) no servidor. Esses programas recebem como entrada os dados vindos dos usuários nas máquinas cliente, e após o processamento retornam o resultado como uma nova página HTML. A Figura 6 ilustra três possibilidades de operações que estariam relacionadas com scripts CGI, como envio de e-mails, acesso à base de dados armazenada no próprio servidor ou então a execução de algum comando no sistema operacional.

¹ <http://www.w3schools.com/js/>

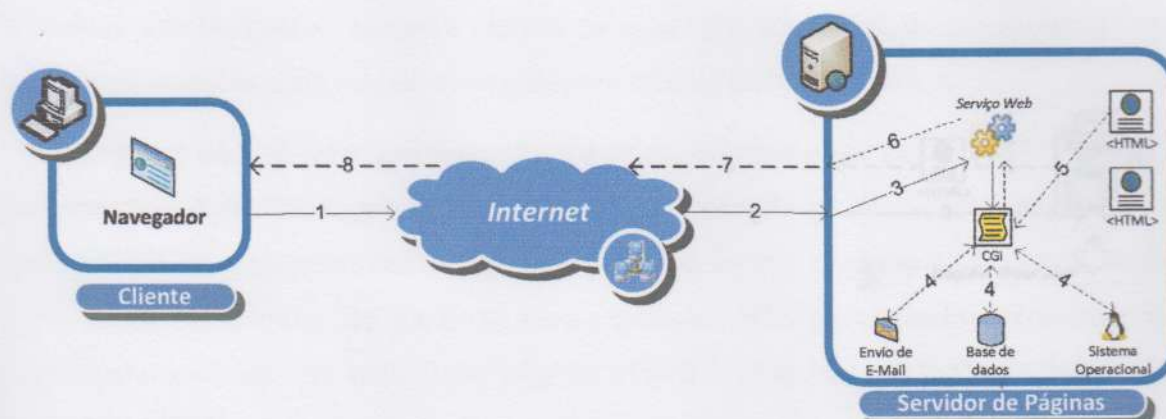


Figura 6. Esquema de acesso a conteúdo dinâmico através de scripts CGI
Fonte: Dados da Pesquisa (2007)

O desenvolvimento de scripts CGI é, entretanto, uma tarefa árdua devido à necessidade de tratar os eventos diretamente sob o formato do protocolo HTTP, bem como à dificuldade de gerenciar o estado das requisições de um mesmo cliente. Surgiram, então, pacotes e APIs (*Application Programming Interface*) em diversas linguagens para facilitar essa tarefa.

Depois de um tempo essa solução passou a ser também implementável na plataforma Java com o surgimento da API de Servlets (MORDANI, 2003). Essa interface define as interações entre os Servlets e os *web containers*, que são nada mais que aplicações Java executadas no servidor web responsáveis pela manipulação dos Servlets, como o mapeamento de uma URL para a invocação de uma determinada classe que representa o Servlet.

Essa API permite aos desenvolvedores abstrair o funcionamento do protocolo HTTP (ou qualquer outro protocolo previsto na API) e preocupar-se com a implementação da lógica da aplicação. Uma das grandes vantagens dessa abstração, por exemplo, é a possibilidade de manter o estado das conexões por todas as transações do servidor. O *container* fica responsável por implementar uma das estratégias para solucionar esse problema, seja através do uso de “*cookie*” ou re-escrita de URLs (BALL, et al., 2006), entretanto para desenvolvedor isso é abstraído através da manipulação de objetos da classe `javax.servlet.http.HttpSession`.

O objetivo dos Servlets é exatamente o mesmo dos scripts CGI, entretanto como são componentes de software implementados na plataforma Java, disponibilizam todo o poder da plataforma ao processamento dos dados submetidos pelo cliente. Entre os diversos oferecidos, é possível citar o acesso às APIs de acesso a base de dados e envio de e-mails (como ilustrado na Figura 7) independentemente de plataforma.

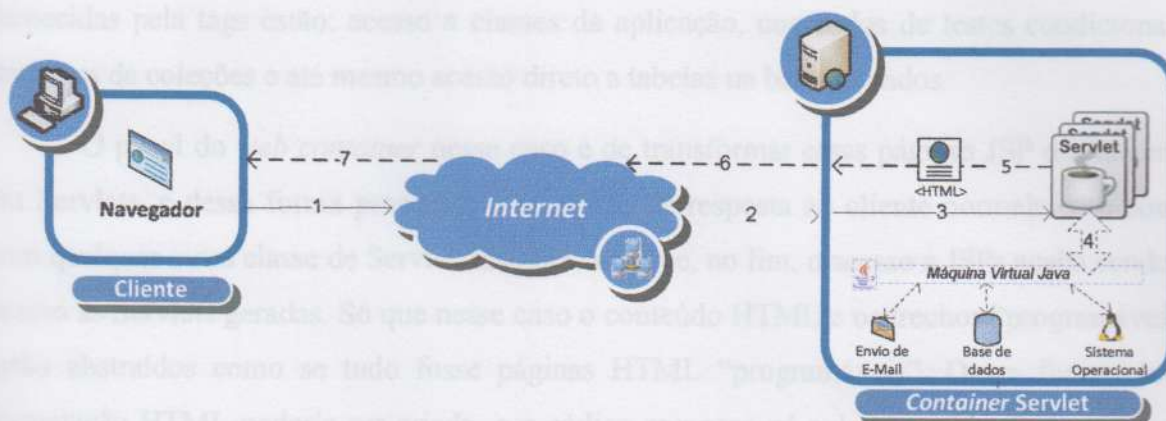


Figura 7. Esquema de acesso a conteúdo dinâmico através de Servlets
Fonte: Dados da Pesquisa (2007)

O funcionamento das Servlets se assemelha ao do CGI, porém as classes da API de Java, de uma forma mais organizada, precisam ser executadas em algum *web container*, empacotado em uma aplicação Web. A partir disso é possível manipular as requisições vindas do navegador e devolver respostas em forma de texto (já que o HTTP é baseado em texto) com marcações HTML através somente da criação dessas classes no *container*. Porém, a geração dessas páginas linha-a-linha nas classes, de certa forma forçava os *web designers*, que deveriam ser responsáveis pela apresentação visual da aplicação, a exercer papéis que se confundiam com os dos programadores.

Uma solução para desenvolvimento de aplicações Web que envolva somente a criação de Servlets está sujeita a sérios problemas com manutenção e evolução do sistema. Essa abordagem monolítica força a edição de conteúdos HTML em classes Java, o que torna a tarefa de codificação no mínimo improdutiva.

Desde o surgimento das Servlets o desenvolvimento para aplicações Web utilizando Java tem se tornado cada vez mais prático e organizado. A comunidade criou o JSP (JavaServer Pages) (PELEGRI-LLOPART; ROTH, 2003), uma linguagem de script pré-processada que pode ser inserida em páginas com códigos HTML, sendo armazenadas no servidor e requisitadas pelos clientes. Com JSPs, os documentos passaram a não somente formatar dados, mas também usar recursos comuns a linguagens de programação como testes condicionais, laços iterativos e até acesso direto a banco de dados.

As páginas JSPs são muito semelhantes a páginas HTML comuns, exceto pela inserção de trechos de código Java ou tags específicas da tecnologia. O uso de tags permite uma maior abstração aos profissionais (sem conhecimento da linguagem de programação) de como a aplicação irá acessar os dados para preencher a página. Entre várias funcionalidades

fornechas pela tags estão: acesso a classes da aplicação, comandos de testes condicionais, iterações de coleções e até mesmo acesso direto a tabelas na base de dados.

O papel do *web container* nesse caso é de transformar essas páginas JSP diretamente em Servlets, e dessa forma processar o conteúdo de resposta ao cliente normalmente como com qualquer outra classe de Servlet. Isso faz com que, no fim, o acesso a JSPs acaba sendo o acesso às Servlets geradas. Só que nesse caso o conteúdo HTML e os trechos “programáveis” estão abstraídos como se tudo fosse páginas HTML “programáveis”. Dessa forma, toda formatação HTML poderia ser criada, e o código responsável pelo conteúdo dinâmico seria inserido posteriormente pelos programadores. Essa maneira de desenvolver esse tipo de aplicação passou a ser conhecida como Modelo 1 (DUDNEY, et al., 2004). Seguindo o modelo das figuras anteriores que mostram o comportamento da aplicação de acordo com a tecnologia, a Figura 8 ilustra o funcionamento de uma aplicação que utiliza JSP.

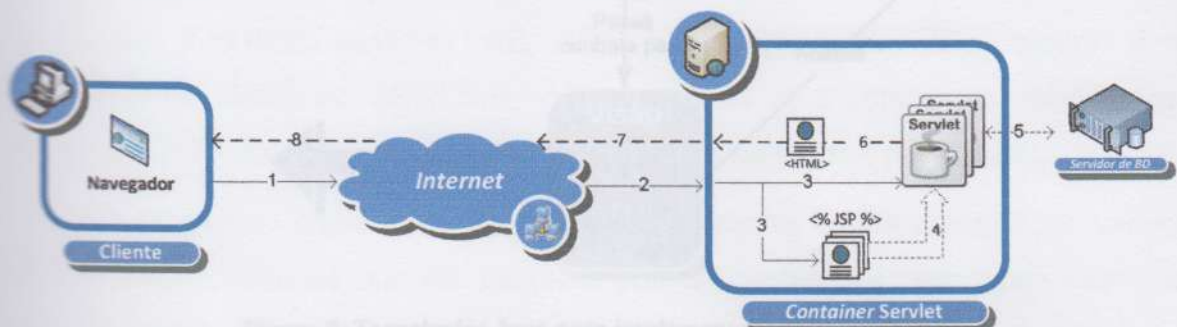


Figura 8. Funcionamento das Java Server Pages
Fonte: Dados da Pesquisa (2007)

Entretanto, esse cenário ainda não era o ideal sob o ponto de vista dos desenvolvedores, pois ainda havia certa confusão com os papéis dos profissionais. O mesmo acontecia com alguns procedimentos realizados pelo sistema, como acesso à base de dados e decisão de qual página seria exibida, que também ficavam todos em meio a códigos de formatação de textos, o que dificultava a manutenção e organização dos sistemas.

Apesar das facilidades providas das JSPs, esse modelo de desenvolvimento ainda renegava os princípios da separação de interesses. Dessa forma hoje pode-se concluir que a grande contribuição das JSPs foi o aumento da produtividade na geração dos Servlets, porque os maiores problemas como os de manutenção e evolução do sistema continuavam presentes.

A partir dessa motivação, o desenvolvimento de aplicações Web passou a adaptar o padrão arquitetural MVC (até então mais aplicado para aplicações de desktop, como descrito no tópico anterior). Essa estratégia passou a ser conhecida como Modelo 2 (MANN, 2005) (por ser considerada uma evolução do chamado Modelo 1) ou MVC-2 (por ser uma adaptação

do original MVC aplicado às aplicações de desktop). Nesse modelo, é possível separar as responsabilidades de maneira simplificada como formatação dos dados na Visão, acesso aos dados no Modelo e o controlador como mapeador das requisições do cliente para operações sobre o modelo e escolha da formatação dos dados.

Como ilustrado na Figura 9, o MVC-2 não é uma nova tecnologia, é apenas uma forma mais organizada de desenvolver aplicações Web se utilizando de tecnologias e padrões arquiteturais maduros. Nessa figura especificamente estão representadas as tecnologias da plataforma Java comumente aplicadas em cada camada do modelo de desenvolvimento.

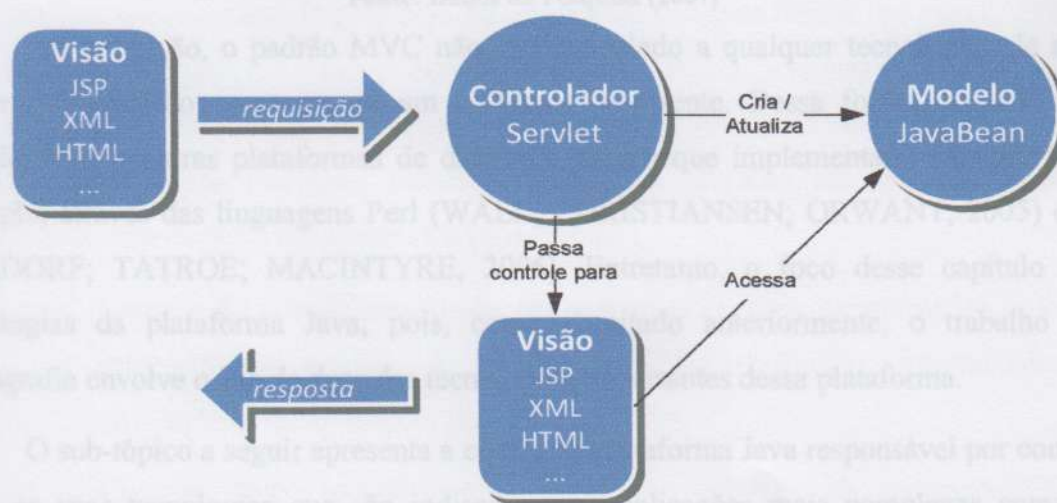


Figura 9. Tecnologias Java para implementar o padrão MVC-2
 Fonte: Dados da Pesquisa (2007)

A arquitetura de um sistema que se utilize desse padrão pode ser observada na Figura 10, onde é indicado o uso das camadas e das respectivas tecnologias:

- O controlador: uma Servlet (ou várias Servlets) responsável por tratar as requisições do cliente e decidir que classe auxiliar deve invocar para executar determinada operação e em seguida decidir que página JSP deve formatar o resultado obtido nessa operação;
- O modelo: as entidades do domínio da aplicação são comumente representadas por classes no padrão JavaBeans (FLANAGAN, 2006), que podem ou não serem submetidas a um mapeamento objeto-relacional para abstrair a persistência dos dados;
- A visão: páginas JSPs são utilizadas para formatar os dados a serem exibidos ao usuário e auxiliar na maneira de submissão de dados.

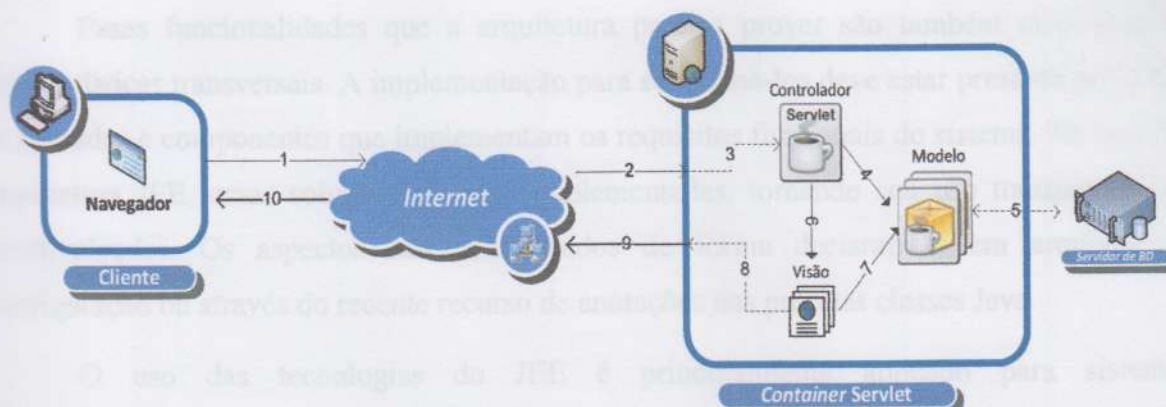


Figura 10. Padrão MVC usando tecnologias Java
Fonte: Dados da Pesquisa (2007)

Por definição, o padrão MVC não está vinculado a qualquer tecnologia, ele apenas indica uma solução comum para um problema recorrente. Dessa forma, existem outras tecnologias em outras plataformas de desenvolvimento que implementam esse padrão, por exemplo, através das linguagens Perl (WALL; CHRISTIANSEN; ORWANT, 2005) e PHP (LERDORF; TATROE; MACINTYRE, 2006). Entretanto, o foco desse capítulo é nas tecnologias da plataforma Java, pois, como já citado anteriormente, o trabalho dessa monografia envolve o uso de duas das tecnologias importantes dessa plataforma.

O sub-tópico a seguir apresenta a edição da plataforma Java responsável por convergir todas as suas tecnologias que são indicadas para aplicações mais complexas como para ambientes empresariais. Nos tópicos seguintes serão apresentadas as ferramentas que auxiliaram o desenvolvimento das aplicações baseadas no padrão MVC que foi apresentado.

2.3.1 Java EE – Edição Corporativa da Plataforma Java

A plataforma Java possui uma edição especial para atender aos principais requisitos não funcionais de aplicações corporativas de grande porte, e ainda prover alta produtividade para o desenvolvimento. Essa edição é chamada de JEE¹ (do inglês *Java Enterprise Edition*), e visa fornecer uma infra-estrutura às aplicações que atenda a alguns desses principais requisitos não funcionais para sistemas de grande porte: desempenho, segurança, confiabilidade e escalabilidade. A importância de projetar uma arquitetura que os atenda já foi abordada anteriormente. Entretanto ao utilizar JEE, tão importante quanto atendê-los, é evitar que eles ocupem o desenvolvedor demasiadamente, enquanto ele deveria estar focado em solucionar os problemas de negócio da aplicação.

¹ <http://java.sun.com/javaee/>

Essas funcionalidades que a arquitetura precisa prover são também chamadas de características transversais. A implementação para solucioná-los deve estar presente por todas as camadas e componentes que implementam os requisitos funcionais do sistema. No caso da arquitetura JEE, essas soluções já estão implementadas, tornando seu uso transparente ao desenvolvedor. Os aspectos são configurados de forma declarativas em arquivos de configuração ou através do recente recurso de anotações nas próprias classes Java.

O uso das tecnologias do JEE é principalmente aplicado para sistemas “multicamadas”. A Figura 11 exibe uma disposição clássica de uma arquitetura com várias camadas lógicas (ou *layers*) implantadas em diferentes camadas físicas (ou *tiers*). O uso de camadas separadas permite o menor acoplamento entre os componentes da arquitetura além de possibilitar a distribuição dos mesmos em camadas físicas diferentes. Na Figura 11 é possível identificar as principais tecnologias do JEE aplicadas em camadas específicas normalmente consideradas em sistemas corporativos:

- Camada do cliente: encontrada nas máquinas clientes, as aplicações podem ser representadas através de páginas HTML dinâmicas interpretadas por navegadores ou aplicações desktop de janelas;
- Camada Web: camada presente em um servidor de aplicações JEE, é responsável pela formatação de dados em páginas HTML, em que são utilizadas tecnologias vinculadas a *templates* de marcações HTML como JavaServer Pages (JSP);
- Camada de negócio: também presente em um servidor de aplicações JEE, utiliza-se de componentes Enterprise JavaBeans para implementar regras de negócio, que permite o uso de aspectos transacionais e distribuídos de forma transparente ao desenvolvedor;
- Camada de dados: essa camada pode servir tanto como acesso à base de dados distribuídas da aplicação, como a sistemas legados.



Figura 11. Arquitetura de JEE em diferentes camadas.

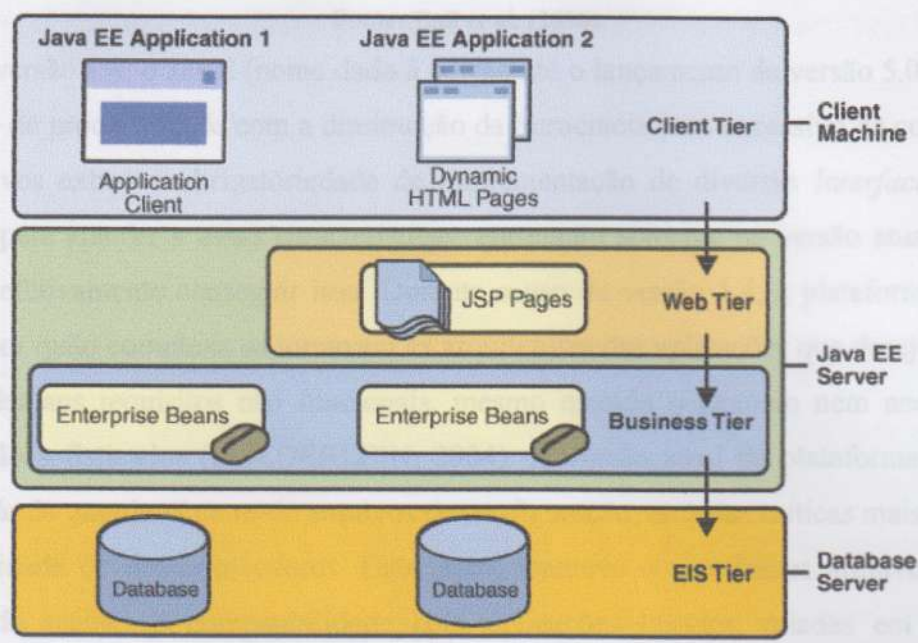


Figura 11. Aplicações Multicamadas.

Fonte: Ball et al. (2006).

Essa edição corporativa da plataforma Java é o resultado do agrupamento de várias APIs especificadas para atender a esses requisitos não funcionais de arquiteturas de grandes aplicações corporativas visando à produtividade e a compatibilidade entre as diversas funcionalidades. A Figura 12 representa a disposição das APIs do JEE em diferentes *containers*. Os *containers* exercem importante papel na arquitetura do JEE, eles representam ambientes de execução que disponibilizam um conjunto de determinadas APIs com a finalidade de implementar uma camada específica da arquitetura.

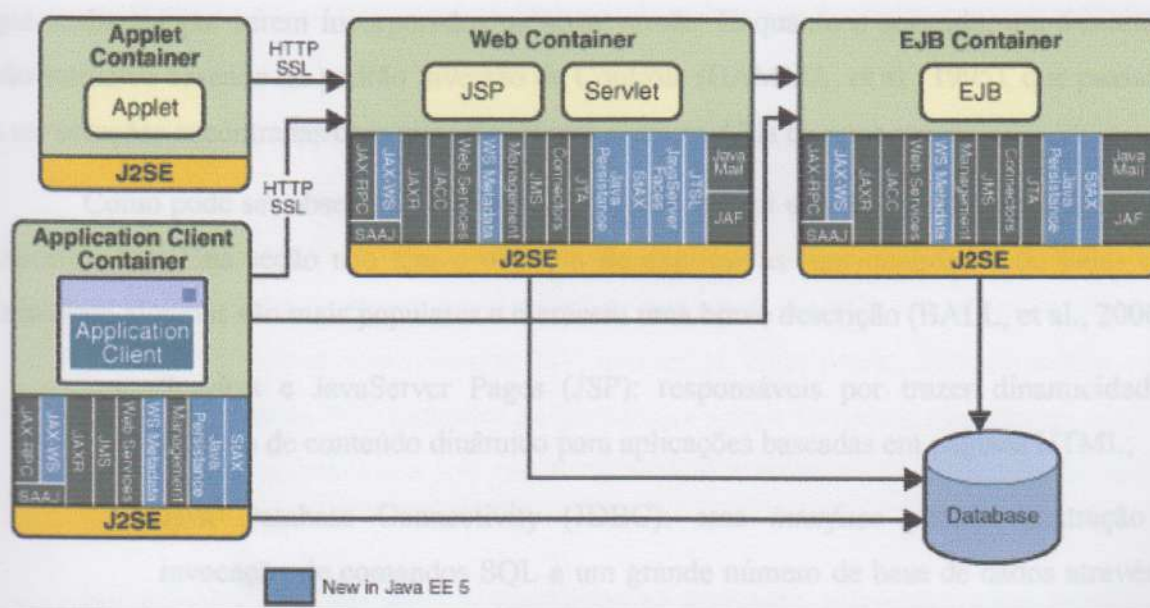


Figura 12. APIs do Java EE 5 em diferentes *containers*.

Fonte: Ball et al. (2006).

A versão 1.4, o J2EE (nome dado à edição até o lançamento da versão 5.0) já pregava o aumento de produtividade com a diminuição da burocracia (uso excessivo de configurações com arquivos extras e obrigatoriedade de implementação de diversas *Interfaces* e classes abstratas) para atender a essas características, entretanto somente na versão atual (JEE 5) a Sun pôde efetivamente conseguir isso. Durante o uso da versão 1.4, a plataforma foi muito criticada por quão complexa se tornavam as arquiteturas das aplicações que desejavam usá-la para atender aos requisitos não funcionais, mesmo quando o negócio nem necessitava de aparatos tão sofisticados (DOEDERLEIN, 2004). A versão atual da plataforma eliminou a necessidade do grande número de arquivos de configuração, uma das críticas mais recorrentes da comunidade de desenvolvedores. Entretanto, manteve o uso desses arquivos opcionais conseguindo manter a compatibilidade com aplicações legadas, criadas em as versões anteriores.

A comunidade de desenvolvedores da plataforma exerceu uma influência importante nas novidades e evoluções dessa nova versão. A popularidade dos projetos de ferramentas alternativas às soluções fornecidas pela plataforma foi suficiente para suas principais idéias e conceitos incorporarem a nova versão da edição corporativa. Talvez os principais exemplos disso sejam os *frameworks open-source* Hibernate (MINTER; LINWOOD, 2005) e Spring (WALLS; BREIDENBACH, 2005). O primeiro, como *framework* de persistência de dados, popularizou o uso significativo de metadados e persistência baseada em POJOs (*Plain Old Java Objects*) (MINTER; LINWOOD, 2005), e que acabaram por serem incorporados na nova versão. Enquanto o segundo, um *framework* não-intrusivo baseado no padrão Inversão de Controle (GAMMA, et al., 1995), que passaram a ser soluções encontradas com mais frequência entre as APIs da nova versão.

Como pode ser observado na Figura 12, o JEE possui uma grande quantidade de APIs especificadas. Esta seção não tem o objetivo de explicar as funcionalidades de todas elas, entretanto algumas são mais populares e merecem uma breve descrição (BALL, et al., 2006):

- Servlets e JavaServer Pages (JSP): responsáveis por trazer dinamicidade à geração de conteúdo dinâmico para aplicações baseadas em páginas HTML;
- Java Database Connectivity (JDBC): essa *interface* permite abstração na invocação de comandos SQL a um grande número de base de dados através de acesso a métodos de classes Java;

- Enterprise JavaBeans (EJB): componentes EJBs são classes Java criadas para executar alguma lógica de negócio da aplicação. Esses componentes são executados em um container específico nos servidores de aplicação JEE. Nesse ambiente de execução, alguns recursos fornecem aos EJBs características importantes como: transacionalidade das operações, execução distribuída e controle de acesso.
- Java Persistence API (JPA): responsável pela persistência de entidades em classes Java através mapeamento objeto-relacional entre classes e tabelas do banco de dados. Além do mapeamento através da configuração dos metadados, a API fornece uma linguagem de consulta própria para esse mapeamento;
- JavaServer Faces (JSF): definido como um *framework* para construção de *interface* gráfica para aplicações Web. A API baseia o desenvolvimento na criação de componentes GUI flexíveis que redenzizam diferentes marcações HTML (ou outras tecnologias de marcação) que compõe a *interface* do usuário na aplicação.

Como previamente colocado no início deste tópico, as tecnologias da edição empresarial da plataforma Java são destinadas ao desenvolvimento de sistemas multicamadas. A partir da Figura 11 e das descrições sobre as principais APIs do JEE, pode-se observar a aplicação específica de algumas delas em camadas diferente da arquitetura. O estudo realizado nesse trabalho está localizado na camada Web da arquitetura desse tipo de sistema, e mais especificamente com foco em dois *frameworks* especializados nessa camada: o Struts e o JSF.

O JavaServerFaces é considerado atualmente uma das principais APIs presente na atual versão do JEE (DOEDERLEIN, 2007). Apesar do Struts não fazer parte do padrão JEE, ele é mantido pela *Apache Software Foundation*¹ (ASF), uma das principais organizações da comunidade de desenvolvedores da plataforma Java, e é baseado em algumas principais tecnologias da plataforma como Servlet, JSP, anotações, entre outras. Embora ambos sejam *frameworks* para camada Web baseados na plataforma Java e forneçam importantes recursos, a natureza de funcionamento dos dois possui diferenças marcantes: uma baseada em ações e outra em componentes.

No próximo capítulo serão apresentadas as principais diferenças entre *frameworks*

¹ <http://www.apache.org/>

baseado em ações (Struts) e em componentes de GUI (JSF). O funcionamento de cada um dos *frameworks* será melhor detalhado.

No capítulo anterior foi demonstrada como as ideias do padrão arquitetural MVC, inicialmente concebido para construção de aplicações para desktop com interfaces gráficas, foram adaptadas para o desenvolvimento de aplicações Web. Essa adaptação, chamada de MVC Modelo-2, pode ser implementada através de algumas tecnologias da plataforma JEE, como *Servlets* (fazendo papel de controlador) e *JSPs* (como a linguagem que traduz diretamente as marcações HTML na camada de visão).

No entanto, a produtividade do desenvolvimento da infra-estrutura de uma aplicação Web é prejudicada se for necessário a implementação de todo esse controle. O principal objetivo dos *frameworks* Web é diminuir os custos do desenvolvimento dessa infra-estrutura e trazer mais produtividade ao desenvolvimento de aplicações Web. Esse capítulo descreve o funcionamento desses *frameworks* e a classificação baseada na manipulação das requisições dos clientes. Em seguida, apresenta uma visão geral sobre os principais componentes da arquitetura e o comportamento dos *frameworks* Struts e JSF, utilizados nos estudos de caso desse trabalho.

3.1 FRAMEWORKS WEB

Apesar de solucionar os problemas enfrentados no modelo de desenvolvimento que envolvia apenas a criação de *JSPs*, como modularidade do sistema e a falta de separação de interesses em camadas, o Modelo-2 também possui outras séries desvantagens, principalmente com a implementação das funcionalidades de controlador, no qual o *Servlet* teria duas principais responsabilidades:

1. Invocar a execução das operações de acordo com algum formato padrão na URL requisitada pelo navegador cliente;
2. Decidir e acionar a exibição de qual *JSP* deve ser renderizada e respondida ao navegador;

A implementação dessa solução para aplicações de grande porte acaba se tornando improdutiva, devido a esse foco excessivo na estrutura do sistema, quando o foco deve ser maior no negócio da aplicação. Visando essa correção de foco, começaram a surgir soluções generalizadas que forneciam partes de uma infra-estrutura implementadas que já tinham

3 FRAMEWORKS WEB

No capítulo anterior foi demonstrado como as idéias do padrão arquitetural MVC, inicialmente concebido para construção de aplicações para desktop com *interfaces* gráficas, foram adaptadas para o desenvolvimento de aplicações Web. Essa adaptação, chamada de MVC Modelo-2, pode ser implementada através de algumas tecnologias da plataforma JEE como Servlets (fazendo papel de controlador) e JSPs (como a linguagem que trouxe dinamismo às marcações HTML na camada de visão).

No entanto, a produtividade do desenvolvimento da infra-estrutura de uma aplicação Web é prejudicada se for necessário a implementação de todo esse controle. O principal objetivo dos *frameworks* Web é diminuir os custos do desenvolvimento dessa infra-estrutura e trazer mais produtividade ao desenvolvimento de aplicações Web. Esse capítulo descreve o funcionamento desses frameworks e a classificação baseada na manipulação das requisições dos clientes. Em seguida, apresenta uma visão geral sobre os principais componentes da arquitetura e o comportamento dos frameworks Struts e JSF, utilizados nos estudos de caso desse trabalho.

3.1 FRAMEWORKS WEB

Apesar de solucionar os problemas enfrentados no modelo de desenvolvimento que envolvia apenas a criação de JSPs, como modularidade do sistema e a falta de separação de interesses em camadas, o Modelo-2 também possui outras sérias desvantagens, principalmente com a implementação das funcionalidades do controlador, no qual o Servlet teria duas principais responsabilidades:

1. Invocar a execução das operações de acordo com algum formato padrão na URL requisitada pelo navegador cliente;
2. Decidir e acionar a exibição de qual JSP deve ser renderizada e respondida ao navegador;

A implementação dessa solução para aplicações de grande porte acaba se tornando improdutiva, devido a esse foco excessivo na estrutura do sistema, quando o foco deve ser maior no negócio da aplicação. Visando essa correção de foco, começaram a surgir soluções generalizadas que forneciam partes de uma infra-estrutura implementadas que já cuidavam

das responsabilidades do controlador. Essas soluções que reúnem partes de uma arquitetura de aplicações Web baseada no Modelo-2 e que facilitam o desenvolvimento principalmente da camada de visão e do controlador, são chamados de *frameworks* da camada Web, ou simplesmente, *frameworks* Web (DUDNEY, et al., 2004).

Na Figura 13 pode-se observar as principais funcionalidades providas pelos *frameworks* Web identificadas nas três camadas da arquitetura da aplicação baseada no Modelo-2 do MVC. Os *frameworks* modernos passaram a renderizar mais que apenas conteúdo HTML para navegadores comuns de computadores pessoais tradicionais. A apresentação pode ser gerada para outros tipos de dispositivos do lado do cliente, como aparelhos de telefone celular ou *handhelds*. O mais importante disso é que a geração dessas diferentes GUIs é transparente para as camadas restantes. A figura também demonstra qual camada é responsável pelo acesso a outros serviços disponível fora do escopo do *framework*.

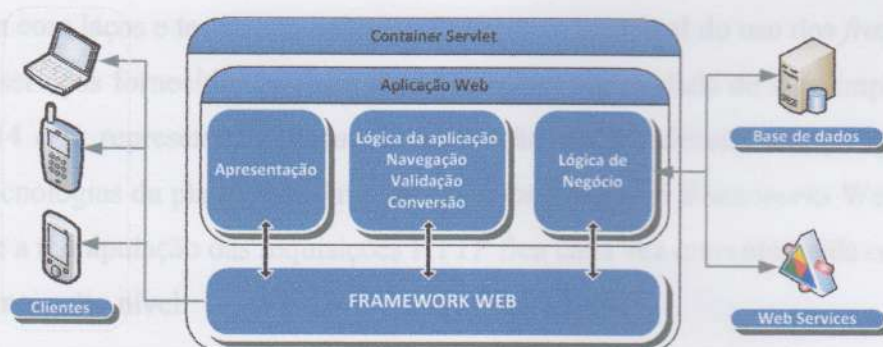


Figura 13. Visão geral dos serviços providos por *frameworks* Web
Fonte: Mann (2005)

Na camada intermediária representada na Figura 13 é onde se encontram as principais vantagens do uso desse tipo de *framework*. Essa camada representa o controlador da aplicação Web, que é responsável por importantes serviços da arquitetura como:

- Conversão de tipo de dados ao instanciar objetos do modelo a partir dos dados submetidos pelo usuário. Essa funcionalidade é muito importante devido à comunicação do cliente com o servidor ser baseada no HTTP. Esse protocolo é baseado no envio de texto, então atributos do modelo com tipos diferentes de String (como inteiros, datas, números decimais e outros) precisam de conversão para popular corretamente os objetos. Essa conversão sendo automática se torna uma facilidade a mais para o desenvolvedor;
- Validação de dados dos formulários preenchidos e submetidos pelos usuários;
- O fluxo de navegação da aplicação configurado, que permite ao *framework* saber

qual componente da apresentação deve ser renderizado e enviado ao cliente.

Os primeiros *frameworks* Web para plataforma Java começaram a surgir em meados do ano 2000 com os lançamentos do Apache Struts e WebWork (LIGHTBODY; CARREIRA, 2005). Para ambos, a maior contribuição era a implementação do controlador e podiam também contribuir com componentes auxiliares para formatação de dados mais complexos nas JSPs, entretanto nenhum se propunha a definir como implementar a camada Modelo.

O principal objetivo de qualquer *framework* é fornecer parte de uma infra-estrutura implementada para um domínio específico de aplicação, que possa ser extensível e adaptável aos problemas específicos de cada projeto. Esses *frameworks* “controladores” para aplicações Web fornecem um conjunto de componentes de software como classes, arquivos de configuração XML (EVJEN, et al., 2007) e marcações de formatação de dados específicas que auxiliam com laços e testes condicionais. O objetivo principal do uso dos *frameworks* é se utilizar dos serviços fornecidos por eles abstraindo a complexidade de suas implementações. Na Figura 14 está representado o grau de abstração do uso desses serviços presentes nas principais tecnologias da plataforma Java, de dois dos principais *frameworks* Web. É possível perceber que a manipulação das requisições HTTP fica cada vez mais abstraída com o uso dos serviços de mais alto nível.

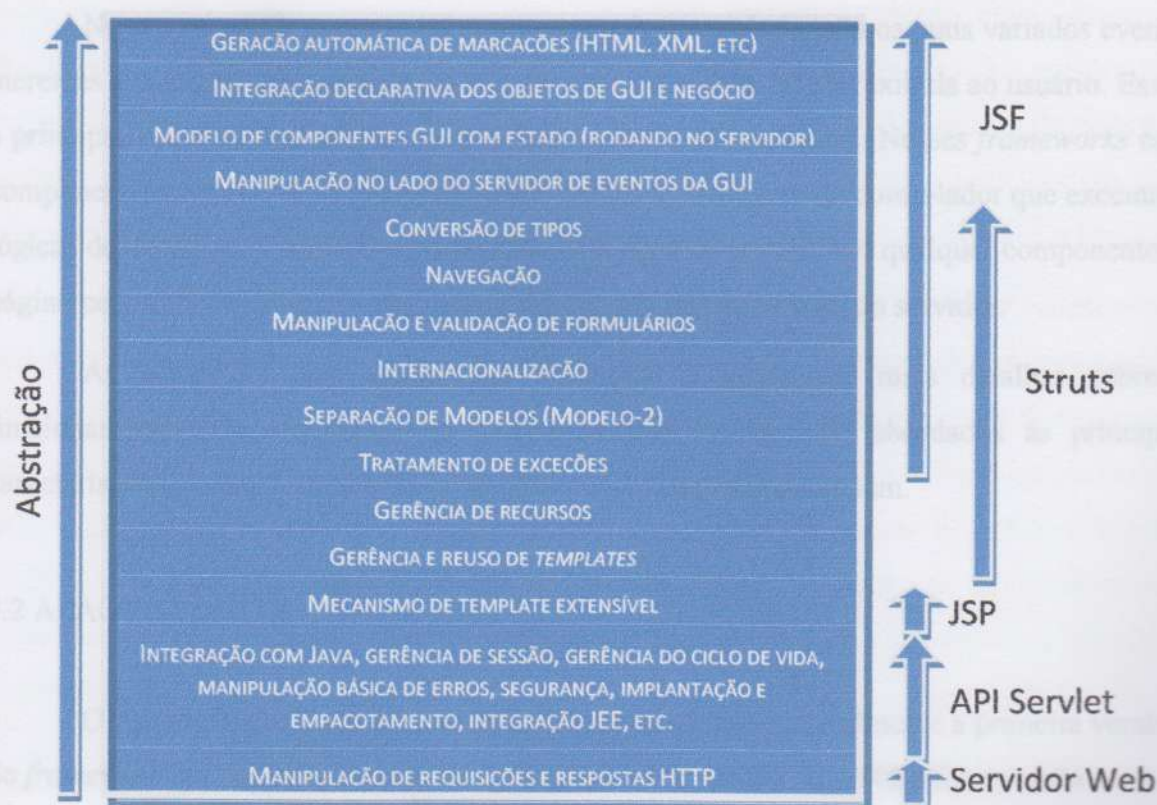


Figura 14. Abstrações de serviços providos pelas tecnologias para aplicações Web.
Fonte: Mann (2005).

Os dois *frameworks* usados na implementação dos estudos de caso dessa monografia são considerados os mais completos e mais utilizados dentre as diversas opções disponíveis no mercado e na comunidade de desenvolvedores. O Struts e o Java Server Faces (JSF) são ambos os *frameworks* aplicáveis em arquiteturas de aplicações Web baseadas no Modelo-2 do MVC, entretanto possuem naturezas diferentes. A principal diferença diz respeito à forma como são manipuladas as requisições geradas a partir da interação do usuário com o conteúdo HTML que lhe é apresentado. A partir desse critério os *frameworks* podem ser baseados em ações ou em componentes.

O Struts é considerado um *framework* baseado em ações devido ao comportamento do controlador com relação ao tratamento das requisições vindas do cliente, seja uma submissão de dados através de um formulário ou um simples redirecionamento de página. Nessa estratégia, toda URL submetida corresponde diretamente à execução de uma ação. No Struts, as URLs são mapeadas (através de um arquivo de configuração XML) para invocar a execução de um método de uma classe. Na implementação desse método se tem acesso aos dados submetidos na requisição e ao objeto de resposta do HTTP, o que permite se utilizar das principais funcionalidades providas pelos *frameworks*, como conversão de tipos por exemplo.

No caso do JSF, as requisições tratadas pelo controlador são os mais variados eventos inerentes a qualquer componente gráfico disposto na página HTML exibida ao usuário. Esse é a principal característica de *frameworks* baseados em componentes. Nesses *frameworks* cada componente da GUI está diretamente ligado a classes auxiliares do controlador que executa as lógicas de negócios. Cada evento disparado pela ação do usuário em qualquer componente da página pode invocar diretamente a execução de uma lógica no lado do servidor.

As próximas duas seções são dedicadas a apresentar mais detalhes sobre o funcionamento e a arquitetura de ambos os *frameworks*. São abordados as principais características, os componentes mais importantes e como se relacionam.

3.2 APACHE STRUTS

O Apache Struts é um projeto open-source que lançou oficialmente a primeira versão do *framework* homônimo em maio de 2000 (TURNER; BEDELL, 2002). Essa versão se

tornou muito popular para desenvolvimento de aplicações Web por muitos anos, chegando a ser considerado um padrão “*de-facto*” para a comunidade enquanto a Sun não respondia a altura com uma API padronizada para a plataforma JEE. O Struts 1.x ainda é responsável por diversos sistemas atualmente em produção implementados ao longo desses anos. Apesar de usar conceitos hoje considerados “ultrapassados”, ainda pode ser considerado uma boa opção para projetistas mais conservadores que optem por aguardar o completo amadurecimento dos *frameworks* modernos lançados recentemente.

A essência do funcionamento do Struts é o mapeamento de URL em ações, desde sua origem até as versões mais recentes. Entretanto a segunda versão mudou consideravelmente a forma de utilizar os recursos do *framework*. Nas primeiras versões seu uso era baseado em inúmeras configurações em arquivos XML e na criação de muitas classes para a implementação dessas ações. Por algum tempo, esses aspectos eram suportáveis, afinal a implementação de toda estrutura do Modelo-2 conseguia ser ainda mais trabalhosa. Entretanto, o crescimento dos sistemas e o surgimento de novos *frameworks* mais simplificados (apesar de menos poderosos) fizeram com que esse excesso de configuração passasse a se tornar burocracia para o desenvolvimento das aplicações.

Com o lançamento da segunda versão no início de 2007, o projeto do Struts foi reestruturado e passou a implementar de forma diferente sua solução baseada em ações. Outro *framework* criado pela comunidade teve um papel importante nessa nova versão. O WebWork, mantido pela OpenSymphony¹, é outro *framework* baseado em ações que disputou por muito tempo a popularidade dos desenvolvedores com o Struts. As equipes de ambos os projetos se uniram e desenvolveram um único *framework* que agregasse o melhor das duas soluções. O resultado dessa parceria é a versão mais recente do projeto da Apache, o Struts2.

Essa versão incorporou recursos importantes que fazem uso de APIs recentemente incluídas na plataforma JSE 5 (SUN MICROSYSTEM, 2007). Alguns deles já considerados imprescindíveis, como o uso opcional de anotações para algumas configurações do *framework* no lugar de extensos arquivos XML. Outros aspectos significantes que foram aplicados têm sido bastante aproveitados em *frameworks* modernos como: suporte a AJAX (CRANE; PASCCARELLO, 2006), uso de classes POJOs (simples classes Java sem uso de *interfaces*, anotações ou super classes), integração com ambientes de execução com suporte a injeção de dependências como Spring, entre outros.

¹ <http://www.opensymphony.com/>

Parece inegável que a evolução desse projeto seja evidente em comparação com as primeiras versões. Apesar de possuir uma porção razoável de adeptos, a nova versão do Struts não se apresenta mais como a solução “dos sonhos” em uma “terra de cego”, como era o ambiente quando lançou sua primeira versão em meio às confusas páginas JSPs ou mesmo trabalhosos controladores Servlets. Atualmente a comunidade já possui diversos *frameworks* Web para os mais diferentes propósitos (VRaptor¹, Mentawai², Tapestry³, Spring MVC⁴, etc.). Como a popularidade de tecnologia algumas vezes transcende a qualidade técnica do produto, seu sucesso ainda é uma incógnita para a comunidade.

3.2.1 Componentes

Embora a nova versão do Struts continue seguindo a mesma idéia de tratar cada requisição vinda do cliente como uma ação a ser processada, a forma com que o *framework* implementa essa solução mudou bastante. A nova implementação passou a usar conceitos do paradigma de Orientação a Aspectos através dos chamados Interceptors e os conteúdos das repostas aos clientes agora podem ser configurados em objetos do tipo ResultType. Esses e outros componentes mudaram radicalmente a estrutura do *framework*, e conseqüentemente a forma de usá-lo. A Figura 15 apresenta uma visão geral dos componentes dessa nova arquitetura e como se comunicam.

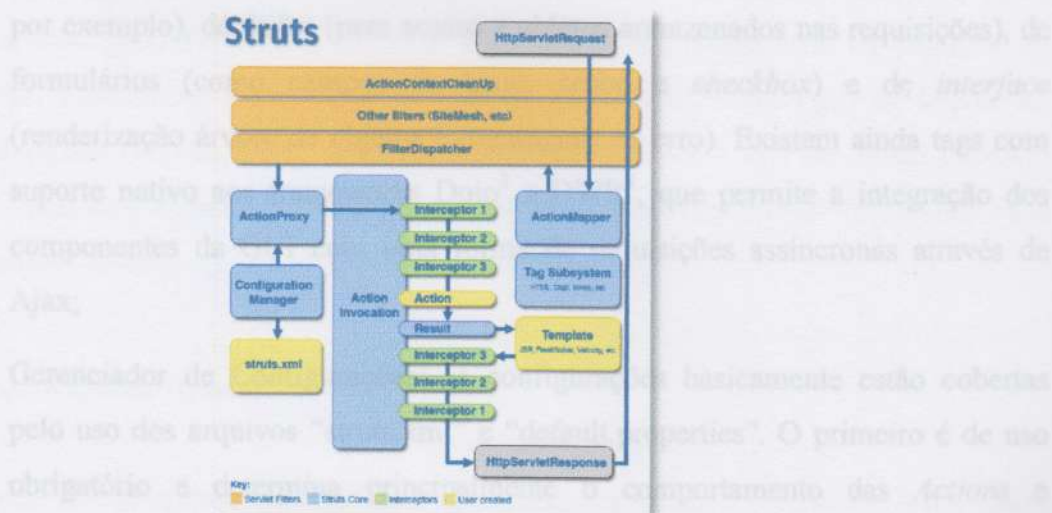


Figura 15. Arquitetura do Struts2.

Fonte: Apache (2007)

¹ <http://www.vraptor.org/pt/index.html>

² <http://www.mentaframework.org/>

³ <http://tapestry.apache.org/>

⁴ <http://www.springframework.org/>

Os principais componentes da arquitetura do Struts2 mostrados na Figura 15 são (ROSEINDIA, 2007):

- *Actions*: as classes de ação são criadas pelo usuário para a implementação das lógicas da aplicação e para acessar os componentes de negócio;
- *Interceptors*: são objetos que processam alguma funcionalidade antes e depois da execução de alguma ação em uma determinada ordem. Muitas funcionalidades adicionais podem ser incorporadas à aplicação através desses objetos como: *upload* de arquivo, tratamento avançado de erros, validações e gerenciamento de fluxo de navegação;
- *Result Types*: são os tipos de resultado de retorno da execução das ações. Eles definem qual tecnologia de *template* irá renderizar o conteúdo a ser exibido ao usuário;
- Tecnologias de visualização (ou *templates*): as quatro tecnologias suportadas por padrão pelo Struts2 são JSP, Velocity¹, FreeMarker² e transformações XLST (HUNTER, et al., 2007).
- Biblioteca de *tags* e suporte a Ajax: são recursos auxiliares para marcação e formatação dos dados a serem apresentados na visão. As *tags* fornecidas estão separadas em tags genéricas de controle (testes condicionais e laços iterativos, por exemplo), de dados (para acesso a objetos armazenados nas requisições), de formulários (como campos de texto, senha e *checkbox*) e de *interface* (renderização árvore de objetos e mensagens de erro). Existem ainda tags com suporte nativo aos frameworks Dojo³ e DWR⁴, que permite a integração dos componentes da GUI com uma forma de requisições assíncronas através de Ajax;
- Gerenciador de Configurações: as configurações basicamente estão cobertas pelo uso dos arquivos “struts.xml” e “default.properties”. O primeiro é de uso obrigatório e determina principalmente o comportamento das *Actions* e *Interceptors*. Enquanto o segundo é opcional, e indica propriedades da aplicação que podem ter seus valores padrão alterados.

¹ <http://velocity.apache.org/>

² <http://freemarker.sourceforge.net/>

³ <http://dojotoolkit.org/>

⁴ <http://getahead.org/dwr>

O funcionamento do *framework* é determinado pelo comportamento desses componentes. O ciclo de vida de uma requisição HTTP sobre a arquitetura do Struts2 faz uso de cada um deles. Esse ciclo de vida está representado na Figura 16, onde mostra os passos tomados pela arquitetura no “processamento” de uma requisição.

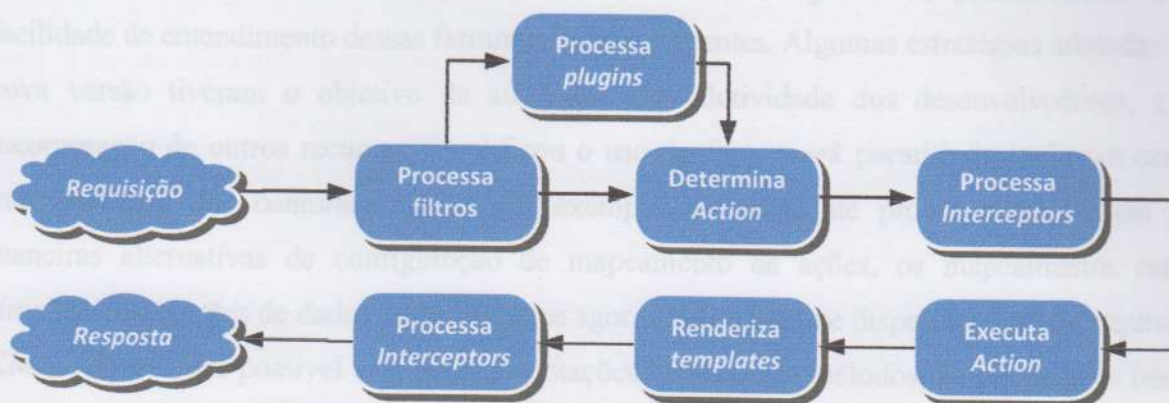


Figura 16. Ciclo de vida de uma requisição no Struts2
Fonte: Dados da Pesquisa (2007)

A cada requisição uma cadeia de filtros (do inglês *filters chain*) é processada, sendo o primeiro deles o *Filter Dispatcher*, que verifica com o *Action Mapper* se há, e qual ação deve ser executada. Outros filtros nessa cadeia possuem papéis opcionais como o “carregamento” de funcionalidades adicionais, ou *plugins*, como uso de tecnologias de padronização de *layout* como SiteMesh¹ e Apache Tiles². Em seguida, o Gerenciador de Configuração é consultado para identificar qual ação deve ser executada. Antes de executar a lógica da aplicação na *Action* solicitada, são processados os *Interceptors* na ordem configurada para essa ação. Finalmente a ação pode ser executada, para em seguida ser renderizada os arquivos de *templates* (JSP, FreeMarker ou Velocity) e gerar a resposta a ser entregue ao cliente no formato configurado (HTML, PDF, entre outros) de acordo com o tipo de *Result* indicado para a ação. Antes de a resposta ser entregue ao cliente, os *Interceptors* são executados mais uma vez na ordem inversa.

3.2.2 Principais Características

Algumas das principais características que surgiram na nova versão do Struts visam combater e solucionar as críticas mais comuns ao uso da primeira versão. A produtividade e o desacoplamento dos principais componentes da estrutura foram os principais focos. Os tópicos a seguir descrevem quais são essas características e como passaram agregar vantagens

¹ <http://www.opensymphony.com/sitemesh/>

² <http://tiles.apache.org/>

ao *framework* (ROUGHLEY, 2006):

Simplicidade no desenvolvimento

O uso das primeiras versões do Struts passou a perder preferência para novos *frameworks* para camada Web, principalmente devido ao ganho de produtividade e a facilidade de entendimento dessas ferramentas mais recentes. Algumas estratégias adotadas na nova versão tiveram o objetivo de aumentar a produtividade dos desenvolvedores, e a incorporação de outros recursos simplificou o uso do *framework* permitindo inclusive certa independência dos *containers* JEE. Um exemplo de ganho de produtividade foram as maneiras alternativas de configuração de mapeamento de ações, os mapeamentos mais simples e validações de dados de formulários agora opcionalmente dispensa o uso de arquivos XML e passa a ser possível usar apenas anotações de classes e métodos. Nas anotações basta informar as URLs de mapeamentos e seus retornos para o caso das ações, e os campos e tipos de validadores no caso das validações.

Outro aspecto prático da nova versão é o uso da estratégia conhecida como “Convenção sobre Configuração” (do inglês “*Convention over Configuration*”). Ela auxilia principalmente o uso de configurações mais simples, mas que em grande quantidade acaba prejudicando a produtividade do desenvolvimento. A partir do princípio apontado pelo nome, o *framework* convenciona uma série de formas de configurações pré-definidas (que geralmente são bastante utilizadas), que se forem adotadas as configurações já estarão implícitas. Um exemplo simples são os arquivos de configuração das validações de dados dos formulários que seguem a convenção de usar o nome da classe da ação concatenado com o trecho “-validation”. Isso implica que todas as ações de submissão de formulários implementadas por essa classe serão validadas pelas regras informadas nesse arquivo de validação, evitando essa associação explícita em algum outro arquivo de configuração.

Outro recurso que há algum tempo vem sendo adotado por diversos *frameworks* da plataforma Java, como na nova versão do EJB (DEMICHIEL; KEITH, 2006), é o uso de classes do tipo POJO (SAM-BODDEN, 2006). A utilização dessas classes para implementar as ações elimina a necessidade de conhecimento de herança e *interfaces* de classes do próprio *framework*. Outra vantagem importante é fato de para serem testadas não são necessárias serem executadas dentro de ambientes dos *containers* JEE, o que facilita bastante os testes unitários.

Menor acoplamento dos componentes

O principal aspecto que torna essa versão mais nova do *framework* tão diferente das primeiras versões é uso extensivo do padrão de Inversão de Controle (IoC) (GAMMA, et al., 1995) através da implementação de Injeção de Dependência (ID). O padrão IoC permite que controle da criação dos objetos da aplicação possa ser delegado a outros componentes responsáveis que não a própria aplicação. Para alcançar esse objetivo o Struts se utiliza de ID para a disponibilização dos objetos através da injeção por métodos Set e por *interfaces* (FOWLER, 2004). O mecanismo de ID usado por padrão é do próprio *framework*, entretanto já existe suporte nativo a outros *frameworks* especializados e mais poderosos como Spring.

O maior benefício dessa técnica é diminuir cada vez mais o acoplamento entre os componentes. Isso permite, por exemplo, que as ações implementadas com classes POJO não precisem manipular nem conhecer os objetos com as informações da requisição e respostas HTTP (`HttpServletRequest` e `HttpServletResponse`). Entretanto se houver essa necessidade, basta configurar a ação para que ela use o *Interceptor* adequado, e o *framework* se encarrega de “injetar” esses objetos na ação.

O uso de *Interceptor* fornece funcionalidades a mais para cada ação através do processamento antes e depois da invocação das ações e principalmente através da injeção de objetos que podem ser manipulados por essas ações. Um exemplo simples é o uso do *interceptor* `ParametersInterceptor` responsável pela manipulação dos parâmetros submetidos em cada requisição. Na execução desse *interceptor* antes da invocação da ação, ele acessa os nomes e valores dos parâmetros submetidos e injeta na ação um mapa com essas informações através da *interface* `ParameterAware`.

O Struts permite a definição de plugins através da criação e reunião de um conjunto de *interceptadores* para prover determinada funcionalidade. Eles devem ser completamente independentes do domínio da aplicação e reusáveis em qualquer aplicação. Basear a implementação da camada Web a partir do uso de plugins permite um baixo nível de acoplamento entre os componentes e possibilita alta flexibilidade na escolha de diferentes soluções para diferentes necessidades.

Uso da linguagem de expressão OGNL

A linguagem de expressão OGNL¹ (Object Graph Navigation Language) é usada como uma forma avançada de acessar propriedades de objetos Java. Na nova versão do Struts, a OGNL é associada uma pilha de valores (do inglês *Value Stack*) que representa a árvore de

¹ <http://www ognl.org/>

objetos disponíveis para serem acessados nas páginas JSPs. Essa árvore segue uma hierarquia de acesso, em que alguns contextos da aplicação Web possuem precedência a outros. Essa ordem de precedência é a mesma que outras tecnologias empregam: parâmetros da requisição, atributos da requisição, atributos da sessão e finalmente atributos no contexto de aplicação.

Essas são as características mais importantes que foram incorporadas ao novo Struts a partir dessas novas funcionalidades. A nova versão ainda possui uma série de outras funcionalidades que já são bastante comuns a *frameworks* do seu tipo, algumas que nem eram contempladas nas primeiras versões (como customização de validação e conversão de dados dos formulários) e outras que já existiam anteriormente (como internacionalização).

As mudanças ocorridas na estrutura e no comportamento do *framework* entre as primeiras e essa segunda versão são muito grandes. Tais alterações foram bastante influenciadas pela associação do projeto da Apache com o WebWork. Algumas das diferenças mais marcantes na nova estrutura vieram do projeto da Open Symphony, como por exemplo, o uso da OGNL como linguagem de expressão e de *Interceptors*, além do uso intenso do padrão Inversão de Controle também trazido por esse projeto.

O Struts2 ainda é uma ferramenta com pouca atuação no mercado devido principalmente ao pouco tempo que possuem as versões estáveis lançadas oficialmente (a primeira versão estável oficial lançada foi em fevereiro de 2007). Entretanto, possui um grande potencial (e uma grande expectativa) para ser mais uma vez líder de preferência para *frameworks* Web orientados a ação.

3.3 JAVASERVER FACES

Assim como o Apache Struts, o JavaServer Faces também é um *framework* Web, entretanto, como já explicado anteriormente, ele é baseado na manipulação dos eventos dos componentes da GUI enquanto o projeto da Apache na execução de ações. A primeira versão do JSF foi lançada no ano de 2004, através da especificação da *Java Specification Request* de número 127 (BURNS; MCCLANAHAN, 2004). Desde a primeira especificação, um dos principais objetivos da JSR era de criar um *framework* baseado no Modelo-2 fortemente voltado para a componentização dos objetos da apresentação. As vantagens mais importantes do JSF são originadas desse foco na camada da visão: uso de ferramentas RAD, terceirização de componentes e independência de dispositivos do cliente.

A criação de componentes de GUI reutilizáveis tornou o uso do *framework* bastante propício para ferramentas que automatizem a inserção desses componentes na aplicação. O uso dessas ferramentas tem o mesmo princípio das ferramentas aplicadas no chamado desenvolvimento rápido de aplicação (ou RAD, do inglês *Rapid Application Development*) (MCCONNELL, 1996). Na plataforma Java, essa idéia já era bastante aplicada na criação de aplicações para desktop com Swing (ELLIOTT, et al., 2002). Nessa forma de desenvolver os componentes são “arrastados” aos *templates* de apresentação da aplicação somente como se estivesse simplesmente desenhando uma tela. Algumas IDEs já possuem suporte a esse poderoso recurso do JSF, inclusive o gratuito NetBeans¹, apresentado na Figura 17.

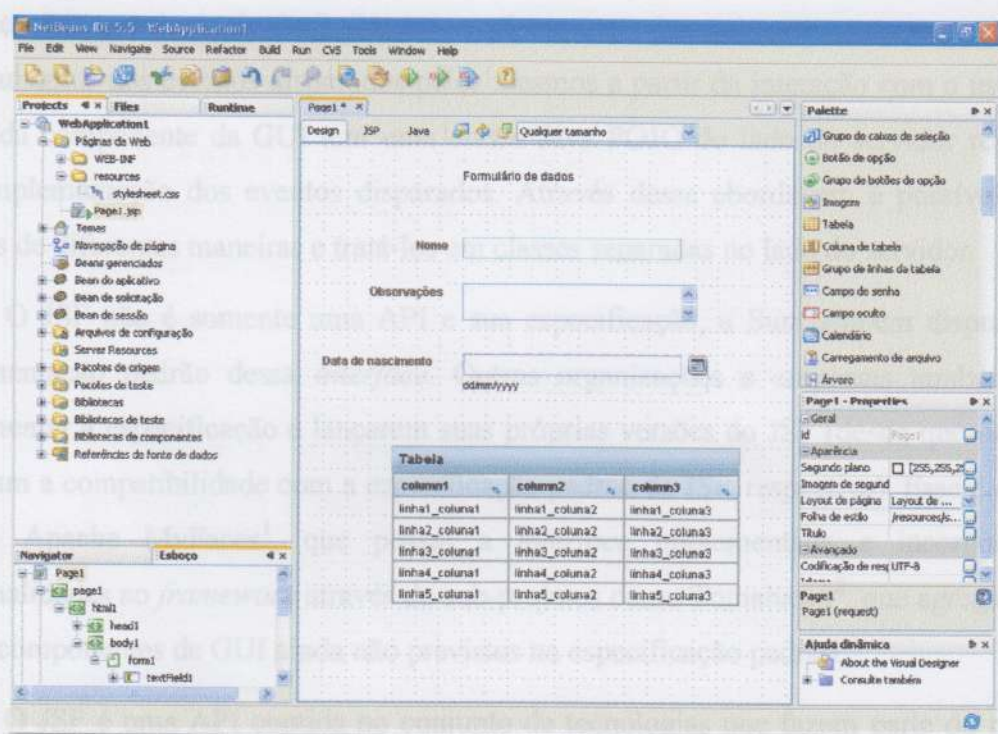


Figura 17. Tela do NetBeans 5.5 com suporte ao RAD de aplicações Web com JSF
Fonte: Dados da Pesquisa (2007)

Ainda devido ao reuso de componentes da GUI, o JSF se diferencia por possibilitar uma modularidade tão expressiva na camada da visão que a terceirização desses componentes já é uma realidade. Atualmente já existem diversas fontes de aquisição de componentes de GUI que podem ser usadas na aplicação com mínima configuração. A biblioteca de componentes Tomahawk da Apache é um exemplo de um projeto *open source* que disponibiliza dezenas de outros componentes não previstos na especificação padrão da API.

Outra grande vantagem oferecida ao desenvolvedor é o verdadeiro desacoplamento entre a camada de visão com, por exemplo, a lógica da aplicação de negócio. Isso permite a

¹ <http://www.netbeans.org/>

independência de qual tipo de dispositivo as requisições do cliente estão sendo enviadas. Através de simples configuração e do uso de outras linguagens de marcação como WML (FOO, 2000) (HTML é a linguagem de marcação padrão usada pelo JSF), é possível formatar conteúdo para outros tipos de dispositivos como aparelhos de telefone celular, sem precisar alterar qualquer outra funcionalidade da aplicação.

Apesar de JSF e Struts, serem ambos considerados implementação do Modelo-2 do padrão MVC, o comportamento do JSF se assemelha muito mais ao do MVC original do que o próprio Struts. Essa característica se deve principalmente à forma como são manipulados os eventos disparados pelos componentes da *interface*, e como interferem diretamente no modelo da aplicação. As principais similaridades envolvem o uso baseado nos componentes da GUI e os tratamentos dos eventos disparados pelos mesmos a partir da interação com o usuário. No JSF cada componente da GUI tem uma classe Java POJO do lado do servidor responsável pela implementação dos eventos disparados. Através dessa abordagem é possível disparar eventos de diferentes maneiras e tratá-los em classes separadas no lado do servidor.

O JSF não é somente uma API e sua especificação, a Sun também disponibiliza a implementação padrão dessa *interface*. Outras organizações e empresas também podem implementar a especificação e lançarem suas próprias versões do JSF (desde que no mínimo cumpram a compatibilidade com a especificação padrão da JSR respectiva). Esse é o caso do projeto Apache MyFaces¹, que possui a *interface* implementada e incorpora novas funcionalidades ao *framework* através de sub-projetos como Tomahawk², que agrega diversos outros componentes de GUI ainda não previstos na especificação padrão.

O JSF é uma API contida no conjunto de tecnologias que fazem parte do padrão de desenvolvimento JEE. Esse fato é uma das principais vantagens apontadas pelos entusiastas da tecnologia. A incorporação da API no JEE a promove ao *status* de *framework* padrão para desenvolvimento de aplicações Web, e essa situação pode motivar os projetistas mais tradicionais a optarem apenas por soluções puramente JEE. Além desse título, outra vantagem é a certeza de compatibilidade das aplicações em qualquer servidor de aplicação JEE, já que eles são requeridos a dar suporte a todas as tecnologias dessa edição.

A versão 1.2 do JavaServer Faces, especificada na JSR 252 (BURNS; KITAIN, 2006), foi lançada no fim de 2006 e possui algumas melhorias em funcionalidades existentes e poucas novidades. A API teve um longo período de amadurecimento até ser considerada

¹ <http://myfaces.apache.org/>

² <http://myfaces.apache.org/tomahawk/>

padrão JEE. JSF já possui uma vasta quantidade de documentações (entre livros, artigos e tutoriais) e sua adoção é auxiliada pela popularidade do uso de ferramentas RAD.

3.3.1 Conceitos Chave

A arquitetura do JSF trabalha com novos conceitos para aplicações Web. Até então, essas aplicações só se baseavam em submissão de URLs e formulários em páginas HTML. Com a proposta de toda a camada da visão ser baseada em componentes, o funcionamento de uma aplicação Web deixou de ser trivial. Os principais conceitos agregados ao JavaServer Faces são:

- **Eventos:** permitem que a aplicação possa abstrair a manipulação das requisições e respostas HTTP, e manipulá-las a partir da geração de disparos ocorridos na interação da *interface* com os usuários. Esses disparos podem ocorrer a partir de qualquer interação com os componentes da página, desde clique em botões até os comandos mais complexo. Entretanto as naturezas desses eventos podem ser quatro: eventos de mudança de valor, de ações, de dados de modelo e de fases;
- **Navegação:** o JSF possui uma configuração especial para determinar toda a navegação de mudanças de páginas. Cada página possui direcionamentos diferentes configuradas em regras de navegação, recurso bastante prático para o gerenciamento da navegação de toda aplicação através de uma ferramenta que auxilie nessa tarefa;
- **Componentes de GUI:** a criação de componentes de GUI reusáveis é importante porque encapsula uma funcionalidade específica que pode ser necessária em vários momentos da aplicação. No caso do conteúdo das respostas ao cliente ser páginas HTML, o uso desses componentes resulta na aplicação de um conjunto de formatações HTML e estilos CSS que, parametrizados, adicionam a mesma funcionalidade em qualquer lugar da aplicação.

3.3.2 Arquitetura

Para implementar essas idéias, a arquitetura do JSF conta com alguns componentes importantes. A Figura 18 ilustra as camadas da Visão e Controlador com seus respectivos

componentes. Assim como o Struts, pode-se perceber que o JSF não define nada sobre como devem ser implementados os acessos ao Modelo. Esses acessos ficam a cargo do desenvolvedor garantir que o controlador não possua dependências graves com essa camada.

3.3.1 Ciclo de Vida

A Figura 18 ilustra o ciclo de vida de uma requisição controlada pelo JSF. O processo começa com uma requisição (request) enviada para o Controlador.

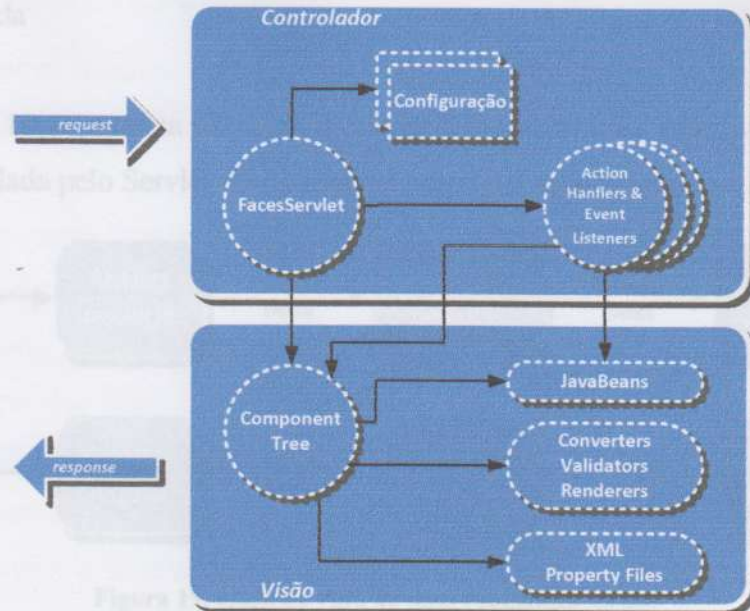


Figura 18. Visão geral da arquitetura do JSF

Fonte: Dados da Pesquisa (2007)

Os principais componentes fornecidos pelo *framework* são:

- *FacesServlet*: esse é o Servlet Controlador, é quem recebe todas as requisições do cliente, prepara, determina a operação na aplicação que será executada e redireciona de volta a resposta;
- *Action Handlers e Event Listeners*: são os manipuladores dos eventos disparados na interface. Eles são os responsáveis por invocar a execução das implementações que representam cada evento;
- *Component Tree*: a representação hierárquica dos componentes da GUI. Cada página renderizada pelo JSF é representada em uma árvore composta por componentes;
- *Converters*: cada componente de GUI pode possuir um conversor de tipos. Eles auxiliam na tradução de tipos complexos para String (e vice-versa).
- *Validators*: os componentes de GUI também possuem validações específicas. O valor associado ao componente é validado de acordo com a configuração do mesmo.
- *Renderers*: são responsáveis pela exibição de conjuntos componentes na

resposta do servidor ao cliente, e pela “tradução” dos mesmos componentes para o controlador no momento na requisição do cliente.

3.3.1 Ciclo de Vida

A Figura 19 representa as fases do ciclo de vida de uma requisição no JSF. Toda requisição controlada pelo Servlet do *framework* passa por essas seis fases.

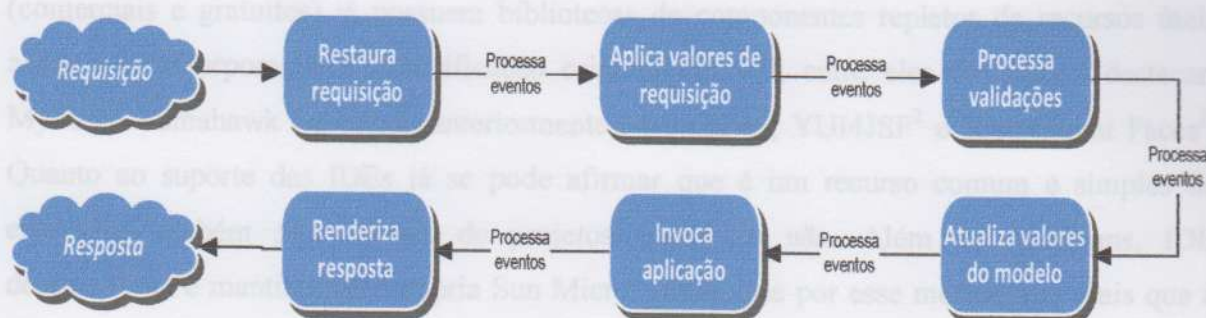


Figura 19. Ciclo de vida de uma requisição no JSF

Fonte: Dados da Pesquisa (2007)

No primeiro momento, a árvore de componentes é criada na restauração da requisição. Em seguida, os valores da requisição são aplicados (e convertidos) nos componentes através dos conversores, para então cada componente ser validado (caso tenha sido configurado para tal). Nos últimos passos, os valores dos objetos de modelo são convertidos e atualizados, para finalmente ser executada a implementação do evento disparado e ser decidido qual será a próxima visão a ser exibida (HIGHTOWER, 2005).

É importante ressaltar que nesse ciclo de vida, ao ocorrerem erros de validação ou conversão de dados a renderização da resposta é automaticamente chamada e os passos seguintes são desconsiderados. Por exemplo, ao ocorrer um erro de conversão de tipos na atualização dos valores do modelo o passo de invocação da aplicação é sumariamente ignorado e a resposta de retorno é renderizada com uma mensagem de erro.

O JSF manipula ações e eventos de forma diferente dentro desse ciclo de vida. Ações e os chamados Eventos de Ações (do inglês *Action Events*) são manipulados pelos *Action Handlers* durante a fase “Invoca aplicação” e representam sempre uma possível mudança de navegação. Enquanto os eventos propriamente ditos, são manipulados pelos *Event Listeners* durante a fase “Processa validações”.

A manipulação de eventos do JSF é um recurso nativo do *framework* que simula um comportamento assíncrono da resposta. Isso porque os eventos são usados para invocar a

aplicação, mas sempre permanecer na mesma página, ou seja, não há renderização de novos componentes e a árvore de componentes não precisa ser re-criada. Dessa forma, a resposta do servidor parece tão mais rápida que uma navegação normal, que o usuário tem a impressão de que não precisou submeter a página ao servidor.

Esses e outros recursos do *framework* têm tornado o JSF cada vez mais popular. Sua crescente popularidade está vinculada aos seus dois recursos mais importantes: componentes da GUI e suporte ao desenvolvimento rápido nas IDEs. Inúmeros projetos de software (comerciais e gratuitos) já possuem bibliotecas de componentes repletos de recursos úteis ainda não incorporados à especificação original da API, entre eles é possível destacar: MyFaces Tomahawk (já citado anteriormente), QuipuKit¹, YUI4JSF² e Rich Client Faces³. Quanto ao suporte das IDEs já se pode afirmar que é um recurso comum e simples de encontrar também seja através de projetos pagos ou não. Além do NetBeans, IDE desenvolvida e mantida pela própria Sun Microsystems, que por esse motivo tem mais que a obrigação de suportar o desenvolvimento com esse *framework*, outras IDEs já consagradas para a plataforma também já possuem esse suporte de forma simplificada como em: MyEclipse⁴, JBuilder⁵, JDeveloper⁶ e diversos plugins do próprio Eclipse⁷.

O uso de *frameworks* de aplicação, como o JSF e o Struts, traz um aumento considerável de produtividade ao desenvolvimento de softwares em comparação com o desenvolvimento baseado no uso de CGI, ou mesmo na combinação JSP e Servlets. Entretanto, a natureza diferente desses dois *frameworks* pode resultar em ganhos diferentes de produtividade no desenvolvimento das aplicações. A avaliação dessa diferença de produtividade realizada nesse trabalho é apresentada a seguir, mostrando sua metodologia e objetivos.

¹ <http://www.teamdev.com/quipukit/>

² <http://yui4jsf.sourceforge.net/>

³ <http://www.rcfaces.org/>

⁴ <http://www.myeclipseide.com/>

⁵ <http://www.codegear.com/br/products/jbuilder>

⁶ <http://www.oracle.com/technology/products/jdev/>

⁷ <http://www.eclipse.org/>

4 APRESENTAÇÃO DA AVALIAÇÃO

Os dois capítulos anteriores tiveram o objetivo de apresentar e descrever os conceitos e o funcionamento das principais tecnologias de desenvolvimento de aplicações Web na plataforma Java. O trabalho dessa monografia consiste em uma avaliação da produtividade do desenvolvimento de aplicações baseadas especificamente nos dois *frameworks* Web apresentados no capítulo anterior.

A avaliação é baseada na análise de aplicações desenvolvidas como estudos de caso. Foram criadas duas aplicações, uma para cada tipo de *framework*, onde o desenvolvimento delas foi focado em solucionar problemas clássicos e funcionalidades típicas de aplicações de aplicações Web de sistemas de grande porte. A análise foi realizada com base em comparações de como cada *framework* permite solucionar e implementar cada um desses problemas e funcionalidades.

Esse capítulo é dedicado a descrever com mais detalhes a avaliação realizada nesse trabalho. O primeiro tópico é destinado a descrever a metodologia adotada para realizar a avaliação. Em seguida, são destacadas as principais diretrizes que guiaram o desenvolvimento das aplicações que serviram de experimento para basear a avaliação de produtividade dos *frameworks*.

4.1 METODOLOGIA

Para atingir os objetivos da avaliação dos *frameworks* foi necessária a elaboração de estudos de caso através do desenvolvimento de pequenas aplicações baseadas em *frameworks* orientados a ação e a componentes. Os *frameworks* Apache Struts e JavaServer Faces foram escolhidos para serem utilizados nessas aplicações. Foram escolhidos os *frameworks* mais importantes das suas categorias (orientados a ações e a componentes) de acordo com o que foi apresentado no Capítulo 4.

Essas aplicações são simples sistemas de informação que simulam o controle acadêmico. Foram desenvolvidas a partir de funcionalidades básicas de Sistemas de Informação (Sis), como autenticação de usuários e tratamento de erros, além de seguir premissas importantes para o desenvolvimento desse tipo de aplicação como separação de interesses. Maiores detalhes sobre esses e outros aspectos dessas aplicações são encontradas no próximo tópico e no Capítulo 5.

Como mencionado anteriormente, a avaliação de produtividade do uso dos *frameworks* Web escolhidos foi baseada na análise de como eles são implementadas soluções para problemas comuns de aplicações Web de grande porte. Esses problemas comuns foram identificados a partir da experiência profissional do autor dessa monografia com desenvolvimento de aplicações Web.

4.1.1 Problemas e Funcionalidades

O desenvolvimento de aplicações Web possui distinções a outros tipos de aplicações em razão de algumas características típicas. Por exemplo, a usabilidade da interface gráfica pode ser comprometida nessas aplicações devido à comunicação remota com o servidor, que pode ser prejudicada pelo desempenho de uma conexão lenta. Serviços e funcionalidades providos por *frameworks* Web devem prever a solução ou pelo menos fornecer subsídios suficientes para suprir tanto os problemas quanto às funcionalidades básicas para esse tipo de aplicação.

Os requisitos funcionais de sistemas de grande porte independem da plataforma de desenvolvimento ou tipo de aplicação a ser desenvolvida. Isso implica que, os mesmos requisitos a serem atendidos no desenvolvimento de aplicações para desktop, também devem ser atendidos para aplicações Web. O que muda é que a implementação de cada funcionalidade varia de acordo com o ambiente de execução da aplicação.

As funcionalidades identificadas nesse trabalho a serem atendidas pelas aplicações desenvolvidas com os *frameworks* escolhidos são bastante comuns em sistemas de informação de grande porte. A implementação de cada uma delas nas aplicações deve fornecer experiências de usabilidade parecidas para o usuário. Com isso torna a comparação mais justa, e torna o uso da aplicação transparente ao usuário sobre qual tecnologia foi utilizada. As principais funcionalidades atendidas pelas aplicações e avaliadas nesse trabalho estão listadas abaixo:

- Cadastro “simples”: as operações de “cadastro simples” correspondem à inserção de dados através de formulários de uma só tela que geralmente é padronizado para diversas entidades do sistema. As operações abrangem também a remoção, atualização e listagem desses dados;
- Cadastro “complexo”: tem finalidade idêntica às operações de cadastro simples,

entretanto possuem um conjunto de informações mais complexas que acaba refletindo na complexidade dos formulários de aquisição dos dados. No caso de aplicações Web, os “cadastros complexos” precisam de mais de uma tela com formulários para o usuário informar os dados, onde cada tela é um passo completado para a conclusão da operação (essa operações também podem ser chamadas de “passo-a-passo”);

- *Upload* de arquivo: o carregamento (ou *upload*) de arquivo é uma funcionalidade bastante comum em grandes sistemas de informação. Principalmente para sistemas que lidam com diversas informações complementares que podem ser importantes ao usuário, mas não para o sistema. Através do carregamento e persistência do arquivo no lado do servidor, o acesso a essas informações podem ficar centralizadas e sempre disponíveis;
- Paginação e classificação de tabelas: a exibição de listas de informações em formato tabular é bastante comum em aplicações Web. Entretanto a usabilidade do sistema pode ser comprometida se essa listagem for tão grande que a exibição da página demore mais que um tempo aceitável. Para casos assim, é preciso que essa tabela seja dividida em páginas e o carregamento das informações seja feito sob demanda do usuário no pedido de visualização de cada página. Além de permitir a classificação dessas informações a partir dos dados de cada coluna da tabela;
- Autenticação de usuário: autenticar a entrada do usuário no sistema é uma das funcionalidades mais básica para grandes sistemas. A segurança e privacidade das informações dos usuários dependem da eficiência do atendimento a esse requisito. Para aplicações Web existe ainda uma dificuldade maior que é o fato das URLs das páginas e operações geralmente são exibidas nos navegadores. Dessa forma, cada operação e cada acesso do usuário devem ser verificados se ele já se identificou para então poder verificar se ele possui permissões suficientes para o acesso.

Aplicações Web não possuem requisitos de funcionalidades muito diferentes em comparação com, por exemplo, aplicações de desktop. Entretanto, devido às suas características por ser uma aplicação distribuída principalmente por se basear na comunicação sem estado do protocolo HTTP, as aplicações Web possuem problemas específicos que

precisam ser solucionados no desenvolvimento. Entre os principais problemas enfrentados no desenvolvimento de aplicações Web, a avaliação de produtividade dos *frameworks* escolhidos analisou o problema da sessão aberta na visão, do inglês *Open Session In View*.

Open Session In View

O problema da sessão aberta na camada de visão ocorre quando a arquitetura da aplicação se utiliza de alguma solução de persistência de dados que precisa manter a sessão da conexão com o banco de dados aberta durante a renderização dos componentes da visão. Isso acontece, por exemplo, com *frameworks* de mapeamento objeto-relacional como Hibernate e a API JPA que se utilizam de estratégias chamadas de *Lazy* e *Eager* para pré-carregar objetos de composição no momento dos objetos maiores.

Caso um relacionamento de um-para-muitos (1-M), como o relacionamento entre Aluno e MatriculaAluno no modelo de domínio ilustrado na Figura 21, esteja mapeado com a estratégia *Eager*, significa que ao carregar um objeto de Aluno, automaticamente os objetos de MatriculaAluno relacionados com esse aluno serão carregados na coleção de matrículas que esse objeto possui. Em muitos casos como esse, em que o relacionamento pode significar o carregamento de dezenas ou centenas de outros objetos automaticamente, essa solução não é considerada escalável para sistemas de alta disponibilidade.

Uma solução é mapear esse relacionamento como *Lazy*, no qual esses objetos somente serão carregados do banco de dados se forem efetivamente invocados pela aplicação. Entretanto, a sessão da conexão com o banco precisa permanecer aberta para que o *framework* reconheça o objeto original. No caso de aplicações Web, as sessões de persistência precisam ser fechadas a cada fim de requisição do cliente para garantir o desempenho do SGBD (Sistema de Gerenciamento de Banco de Dados), mas ao mesmo tempo é necessário que essa sessão permaneça aberta até o final da renderização do conteúdo de resposta ao cliente para que possíveis relacionamentos *Lazy* do domínio possam ser carregados.

4.2 EXPERIMENTOS

Foram desenvolvidas duas aplicações Web que serviram como experimentos para serem extraídos os dados experimentais para a avaliação de produtividade. Cada uma foi criada utilizando um dos *frameworks* Web escolhidos. Foi priorizado o uso das versões mais recentes (até a elaboração desse documento) de cada *framework*: o Struts 2 e o JSF 1.2. Esse

tópico tem o intuito de apresentar os objetivos, requisitos funcionais e domínio dessas aplicações. Mais detalhes sobre a implementação das aplicações podem ser encontrados no Capítulo 6.

O objetivo das aplicações é de cobrir funcionalidades importantes para aplicações Web. As operações realizadas pelos usuários são relativas a um pequeno sistema acadêmico, no qual existem os usuários (professores e alunos) com diferentes níveis de acesso. Os usuários podem pertencer a três diferentes perfis que determinam sua visibilidade de informações e acesso a operações: aluno, professor e administrador. A partir desses perfis, os usuários possuem um conjunto limitado de operações: administradores são responsáveis pelo cadastro dos dados básicos como disciplinas, cursos, alunos, matricular alunos em turmas entre outros; os professores podem acessar e modificar dados de suas turmas como cadastro de material, de notas e faltas e envio de mensagens; e os alunos só podem visualizar as informações de suas respectivas turmas.

Um diagrama de casos de uso está ilustrado na Figura 20, representando alguns dos requisitos funcionais citados no parágrafo anterior. Os casos de uso relacionados com os atores Aluno, Professor e Administrador demonstram a distinção dos níveis de acesso das operações realizadas pelos usuários. Apesar do pequeno número de casos de uso, eles foram escolhidos por serem considerados representativos de acordo com tipos de casos de uso comuns para sistemas de informação.

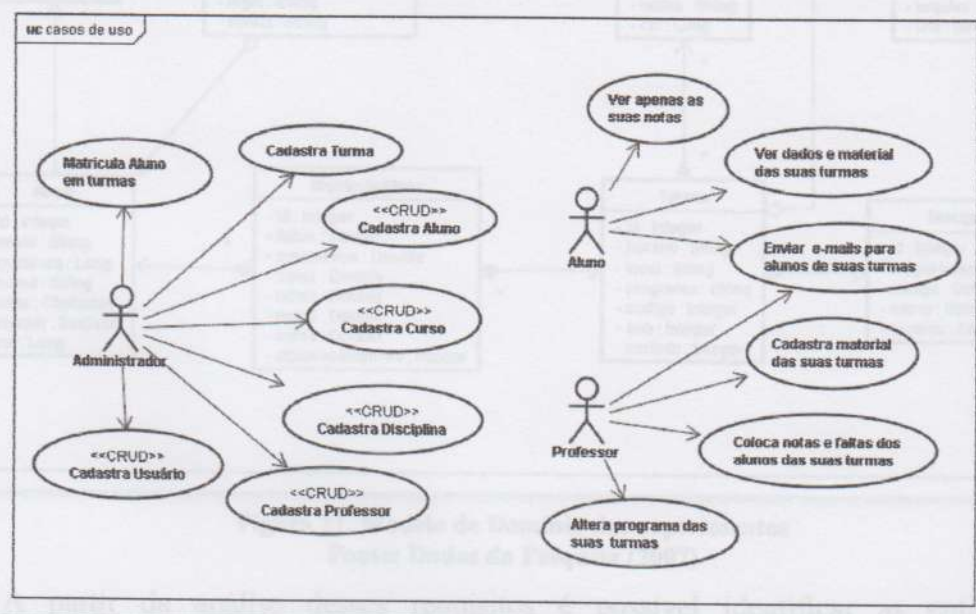


Figura 20. Diagrama de Casos de Uso dos experimentos

Fonte: Dados da Pesquisa (2007)

Os casos de uso identificados com o estereótipo CRUD representam as quatro operações básicas de persistência em base de dados: Cadastro (*Create*), Consulta (*Retrieve*), Atualização (*Update*) e Remoção (*Delete*). Nesses casos de uso, o comportamento dos componentes da arquitetura do sistema é praticamente o mesmo, sugerindo que sua implementação seja generalizada na própria arquitetura. Esses cadastros não possuem uma lógica de negócio diferenciada, e navegação de páginas e componentes de GUI são praticamente os mesmos. Outros casos de uso de cadastro não tão simplificados são: “Matrícula de Alunos em Turmas” e o “Cadastro de Turmas”. Neles, a navegação de páginas é mais complexa e são requeridas regras de negócio como proibir matrícula de alunos em turmas que o aluno já tenha sido aprovado ou a validação de choque de horários entre as turmas dos professores relacionados.

Outros casos de uso que envolve manipulação das informações das turmas de professores e alunos possuem regras e navegações próprias. Atividades como “Cadastro de Material” e “Alterar Programa” podem ser realizadas numa página central com as informações da turma e sua disciplina, enquanto as operações de cadastro de notas e envio de mensagens dos alunos da turma podem ser realizadas em páginas separadas.

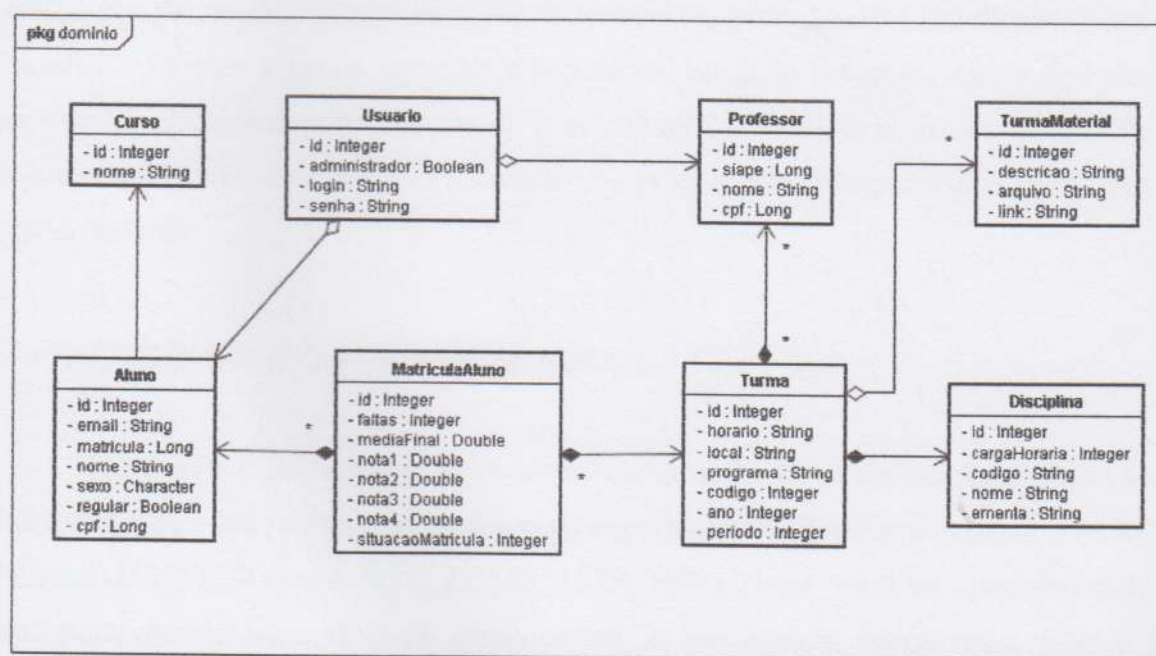


Figura 21. Modelo de Domínio dos experimentos

Fonte: Dados da Pesquisa (2007)

A partir da análise desses requisitos é possível identificar as entidades que representam o domínio da aplicação e como elas se relacionam. A Figura 21 ilustra o modelo do domínio através de um diagrama de classes. Nele, pode-se perceber os relacionamentos de

composição de *MatriculaAluno* com as entidades de *Turma* e *Aluno*. Essa entidade representa o vínculo de cada aluno com uma turma específica e registra ainda informações desse vínculo como número de faltas, notas parciais e média final. Outros relacionamentos importantes são os de agregação entre *Usuario*, *Aluno* e *Professor*, eles indicam que um usuário do sistema pode está vinculado tanto a aluno quanto a um professor.

Com o intuito de compensar o tamanho das aplicações, as operações cobertas por elas devem ser compatíveis com sistemas de grande porte. Ambas as aplicações provêem funcionalidades importantes para aplicações Web, como *upload* de arquivo, classificação e ordenação de tabelas, envio de e-mails e autenticação de usuários. Caso algum dos *frameworks* não suporte alguma dessas funcionalidades, componentes específicos devem ser desenvolvidos. No próximo capítulo serão apresentados mais detalhes de como foram implementadas ambas as aplicações.

O principal objetivo desse capítulo é demonstrar que a forma como as aplicações dos experimentos foram desenvolvidas foi focada em recursos que prezam pela produtividade. Como o uso de recursos oferecidos pelos *frameworks* que auxiliam no seu desenvolvimento específico. O tópico a seguir apresenta a arquitetura usada no desenvolvimento de todos os casos de uso independentemente do *framework* Web utilizado. Os tópicos seguintes são descrições de como cada *framework* permitiu a solução dos problemas e funcionalidades apontados no capítulo anterior.

5.1 ARQUITETURA E CAMADA DE MODELO

Basear todo o processo de desenvolvimento de software na elaboração e validação de uma arquitetura é hoje o foco de um dos mais importantes e utilizados processos, o Processo Unificado (JACOBSON, GRADY, RUMBAUGH, 1999). Nesse processo, a reusabilidade da arquitetura, ou de parte de seus componentes, é um aspecto fundamental e deve ser preocupação dos profissionais responsáveis como arquiteto e/ou engenheiros de software. As aplicações desenvolvidas nesse trabalho foram implementadas utilizando serviços de componentes de uma arquitetura única.

Outro aspecto importante sobre o desenvolvimento dessas aplicações é que os *frameworks* Web utilizados apenas fornecem suporte ao desenvolvimento de componentes nas

5 ESTUDOS DE CASO

A avaliação de produtividade de aplicações implementadas com *frameworks* Web baseados em ações e componentes foi realizada a partir de experimentos desenvolvidos com os *frameworks* Struts e JSF. O Capítulo 4 apresentou os principais objetivos traçados e restrições impostas a essas aplicações para que pudessem servir de exemplos compatíveis com sistemas de informações reais.

Neste capítulo, são apresentados maiores detalhes de como foram implementadas tais aplicações para satisfazerem os requisitos e restrições apresentadas no capítulo anterior. As descrições das soluções serão baseadas na análise de diagramas de classes UML (BOOCH, et al., 2005) que representam em alto nível a implementação dessas soluções. O desenvolvimento das aplicações foi baseado em conceitos aplicados a processos iterativos de desenvolvimento (LARMAN, 2003) em que baseiam a implementação dos casos de uso na utilização de recursos oferecidos pela arquitetura.

O principal objetivo desse capítulo é demonstrar que a forma com que as aplicações dos experimentos foram desenvolvidas foi focada em recursos que prezam pela produtividade. Como o uso de recursos oferecidos pelos *frameworks* que auxiliam no seu desenvolvimento específico. O tópico a seguir apresenta a arquitetura usada no desenvolvimento de todos os casos de uso independente do *framework* Web utilizado. Os tópicos seguintes são descrições de como cada *framework* permitiu a solução dos problemas e funcionalidades apontados no capítulo anterior.

5.1 ARQUITETURA E CAMADA DE MODELO

Basear todo o processo de desenvolvimento de software na elaboração e validação da uma arquitetura é hoje o foco de um dos mais importantes e utilizados processos, o Processo Unificado (JACOBSON; GRADY; RUMBAUGH, 1999). Nesse processo, a reusabilidade da arquitetura, ou de parte de seus componentes, é um aspecto fundamental e deve ser preocupação dos profissionais responsáveis como arquiteto e/ou engenheiros de software. As aplicações desenvolvidas nesse trabalho foram implementadas utilizando serviços de componentes de uma arquitetura única.

Outro aspecto importante sobre o desenvolvimento dessas aplicações é que os *frameworks* Web utilizados apenas fornecem suporte ao desenvolvimento de componentes nas

camadas de visão e controlador. Todos os serviços da camada de modelo são os mesmos para ambas as aplicações. Para alcançar o reuso dos serviços básicos da arquitetura e da camada de modelo específicos do domínio da aplicação, é necessário total independência entre esses elementos e os componentes implementados nas camadas suportadas pelos *frameworks*.

De acordo com o princípio da separação de interesses, os componentes relacionados com funcionalidades similares da arquitetura são agrupados em camadas. Em razão da arquitetura ser baseada no padrão MVC e os *frameworks* já darem suporte às camadas de visão e controlador resta à arquitetura fornecer recursos para a camada de modelo. Na camada de modelo os principais serviços oferecidos são divididos também nas camadas de Negócio (CN) e Acesso a Dados (CAD). A Figura 22 ilustra a distinção dessas duas camadas na arquitetura através do uso de pacotes.

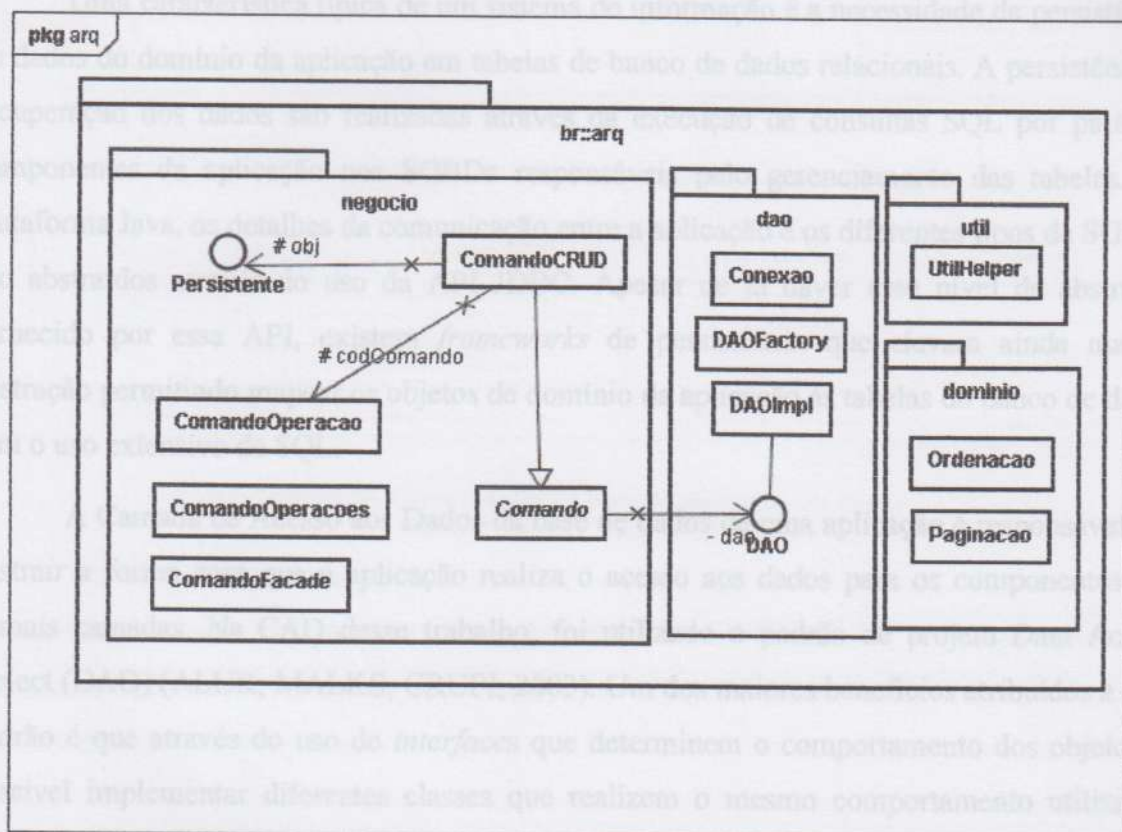


Figura 22. Camadas de negócio e acesso a dados

Fonte: Dados da Pesquisa (2007)

Componentes desenvolvidos para essas camadas possuem comportamentos genéricos que irão atender a maioria das funcionalidades de aplicações independente do seu domínio. Um exemplo disso é o uso de componentes na CAD para consulta de informações numa base de dados. Em componentes mais genéricos as consultas buscam todos os registros ou

parametrizam apenas por chave-primária devido ao fato de não poder depender do domínio de aplicação alguma. Fica a cargo dos componentes da CAD das aplicações específicas estenderem esse comportamento genérico e especializar consultas de acordo com o domínio da aplicação.

Os próximos dois tópicos detalham sobre a implementação dos serviços oferecidos pela camada de modelo da arquitetura para as camadas de visão e de controlador das aplicações. Em seguida os dois últimos tópicos descrevem a estrutura e o comportamento das duas camadas suportadas pelos *frameworks* em cada uma das aplicações.

5.1.1 Camada de Acesso a Dados

Uma característica típica de um sistema de informação é a necessidade da persistência de dados do domínio da aplicação em tabelas de banco de dados relacionais. A persistência e recuperação dos dados são realizadas através da execução de consultas SQL por parte de componentes da aplicação nos SGBDs responsáveis pelo gerenciamento das tabelas. Na plataforma Java, os detalhes da comunicação entre a aplicação e os diferentes tipos de SGBDs são abstraídos através do uso da API JDBC. Apesar de já haver esse nível de abstração fornecido por essa API, existem *frameworks* de persistência que elevam ainda mais a abstração permitindo mapear os objetos de domínio da aplicação às tabelas do banco de dados sem o uso extensivo de SQL.

A Camada de Acesso aos Dados da base de dados de uma aplicação é responsável por abstrair a forma com que a aplicação realiza o acesso aos dados para os componentes das demais camadas. Na CAD desse trabalho, foi utilizado o padrão de projeto Data Access Object (DAO) (ALUR; MALKS; CRUPI, 2003). Um dos maiores benefícios atribuídos a esse padrão é que através do uso de *interfaces* que determinem o comportamento dos objetos, é possível implementar diferentes classes que realizem o mesmo comportamento utilizando diferentes formas de fazê-lo. Um exemplo clássico seria a criação de uma família de classes DAOs concretas que implementem o acesso ao banco de dados através de puras consultas SQL, e outra família de classes que realize as mesmas operações através do uso de um *framework* como Hibernate. A abstração desse acesso é possível pelo fato de que os componentes das outras camadas apenas conhecerão as assinaturas dos métodos das *interfaces*.

A Figura 23 ilustra o uso do DAO baseado em *interfaces* e a generalização de

operações básicas como criação, atualização e listagem de registros reunidos em um objeto genérico. Objetos que necessitem de comportamentos específicos, como uma busca parametrizada de acordo com o domínio da aplicação, deve apenas estender as operações básicas para manter a compatibilidade dos objetos. Esse mecanismo de herança de operações genéricas também pode ser percebida na Figura 23 a partir das *interfaces* TurmaDAO, UsuarioDAO e AlunoDAO.

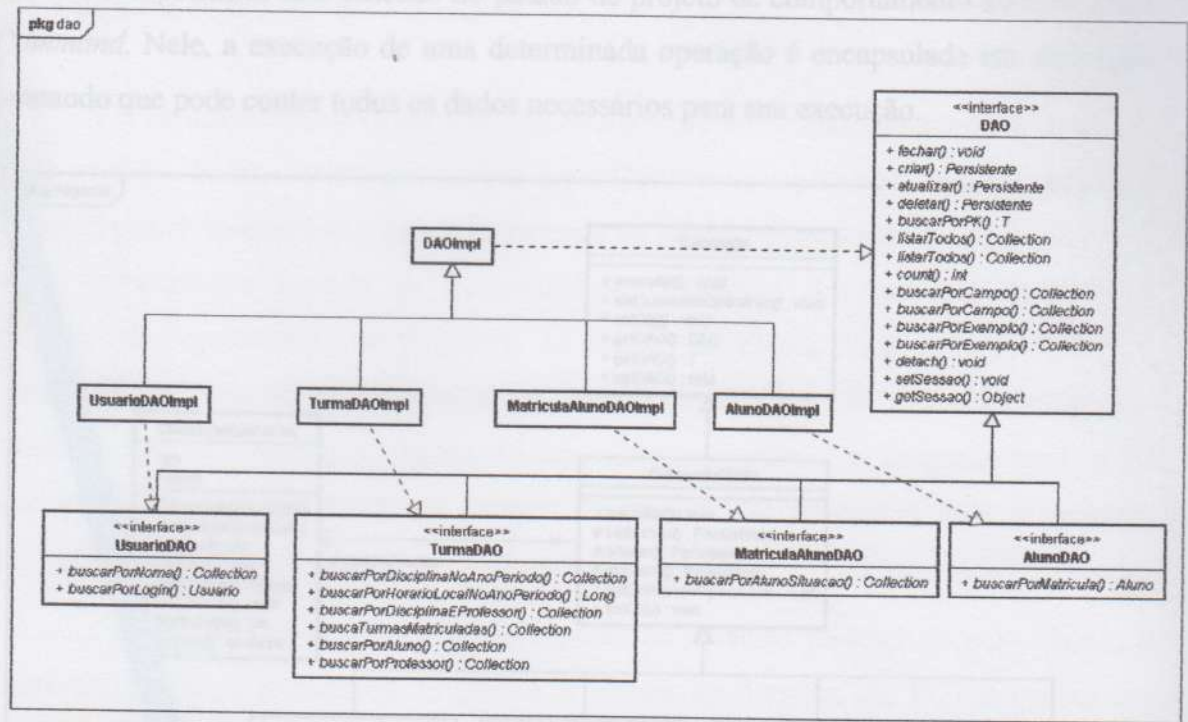


Figura 23. Componentes da camada de acesso aos dados
Fonte: Dados da Pesquisa (2007)

No diagrama de classes da Figura 22 pode-se perceber a classe DAOFactory, outro componente muito importante na implementação da CAD. Ele é responsável pela implementação do padrão de projeto de criação de objetos AbstractFactory reunidos pelo GoF (GAMMA, et al., 1995). Nessa arquitetura, a fábrica de DAOs tem o objetivo de abstrair a criação dos objetos que implementam as *interfaces* de DAOs. Nessa fábrica ainda é usado o Singleton, outro simples padrão de criação do GoF responsável por permitir a criação de um único objeto da fábrica e de garantir que ele seja facilmente acessível de qualquer ponto da aplicação.

5.1.2 Camada de Negócio

A camada de negócio é a outra camada importante da arquitetura utilizada. Ela é responsável por fornecer operações de negócio generalizadas, como as básicas de persistência como cadastro, alteração e remoção de registro entidades. A estrutura dessa camada está representada no diagrama de classes da Figura 24. É possível perceber pelo diagrama que o uso dos componentes está baseado no padrão de projeto de comportamento do GoF chamado *Command*. Nele, a execução de uma determinada operação é encapsulada em um objeto de comando que pode conter todos os dados necessários para sua execução.

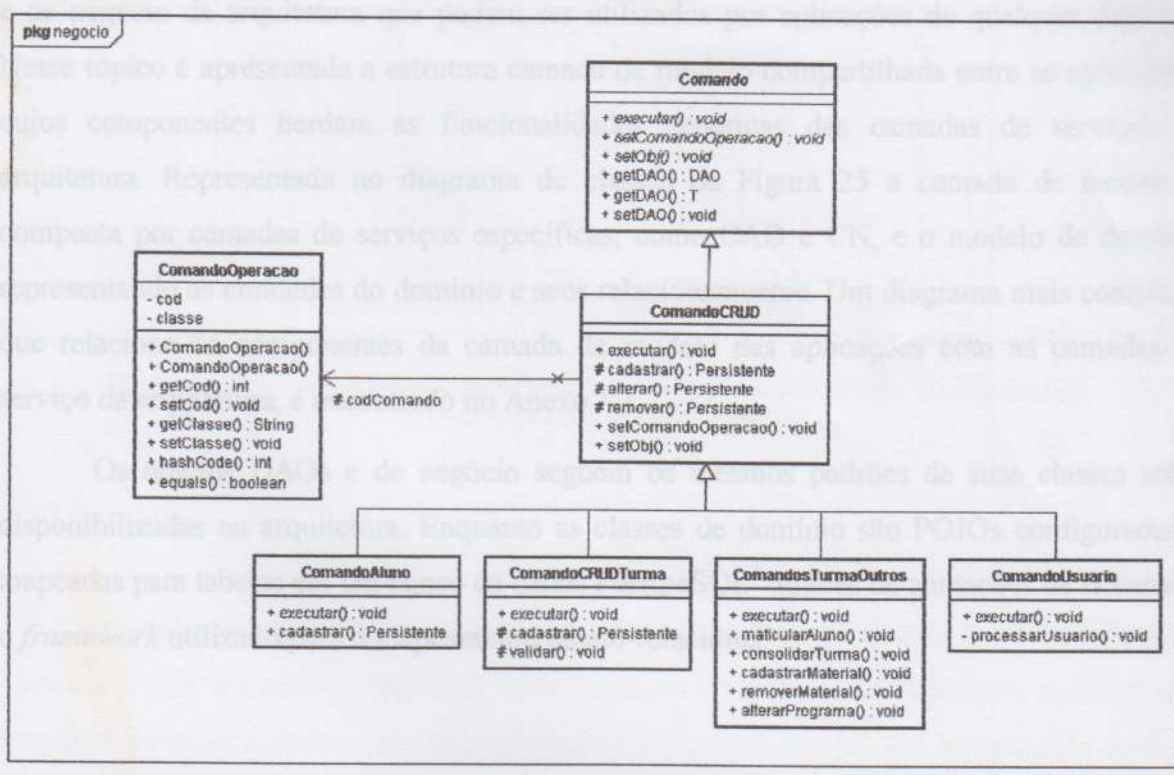


Figura 24. Componentes da camada de negócio

Fonte: Dados da Pesquisa (2007)

Cada comando do sistema deve estender atributos e operações da classe abstrata **Comando**, dessa forma toda camada de negócio pode abstrair a lógica de negócio implementada nos objetos do comando. O principal comando do sistema é representado pela classe **ComandoCRUD** que implementa as operações CRUD já mencionadas em capítulos anteriores. Os comandos específicos das aplicações, os que as regras a serem implementadas dependem do domínio, herdam o comportamento e a estrutura dos comandos generalizados da arquitetura.

A implementação da invocação de cada comando no sistema foi simplificada através

do uso de classes POJO com acesso direto aos objetos DAOs da CAD. Em abordagens mais sofisticadas baseadas na plataforma Java, os componentes de negócio podem ser implementados como componentes EJBs. Componentes dessa tecnologia agregam valores importantes a aplicações de grande porte, como suporte transparente de transacionalidade de operações executadas pelos componentes e processamento distribuído dos mesmos.

5.1.3 Camada de Modelo das Aplicações

Foram apresentados os recursos genéricos oferecidos pelas camadas de acesso a dados e de negócio da arquitetura que podem ser utilizados por aplicações de qualquer domínio. Nesse tópico é apresentada a estrutura camada de modelo compartilhada entre as aplicações, cujos componentes herdam as funcionalidades genéricas das camadas de serviços da arquitetura. Representada no diagrama de classes da Figura 25 a camada de modelo é composta por camadas de serviços específicas, como CAD e CN, e o modelo de domínio representando as entidades do domínio e seus relacionamentos. Um diagrama mais completo, que relacione os componentes da camada de modelo das aplicações com as camadas de serviço da arquitetura, é encontrado no Anexo 1.1.

Os objetos DAOs e de negócio seguem os mesmos padrões de suas classes mães disponibilizadas na arquitetura. Enquanto as classes de domínio são POJOs configuradas e mapeadas para tabelas em um banco de dados PostgreSQL¹ através de anotações de Hibernate o *framework* utilizado para o mapeamento objeto-relacional.

¹ <http://www.postgresql.org/>

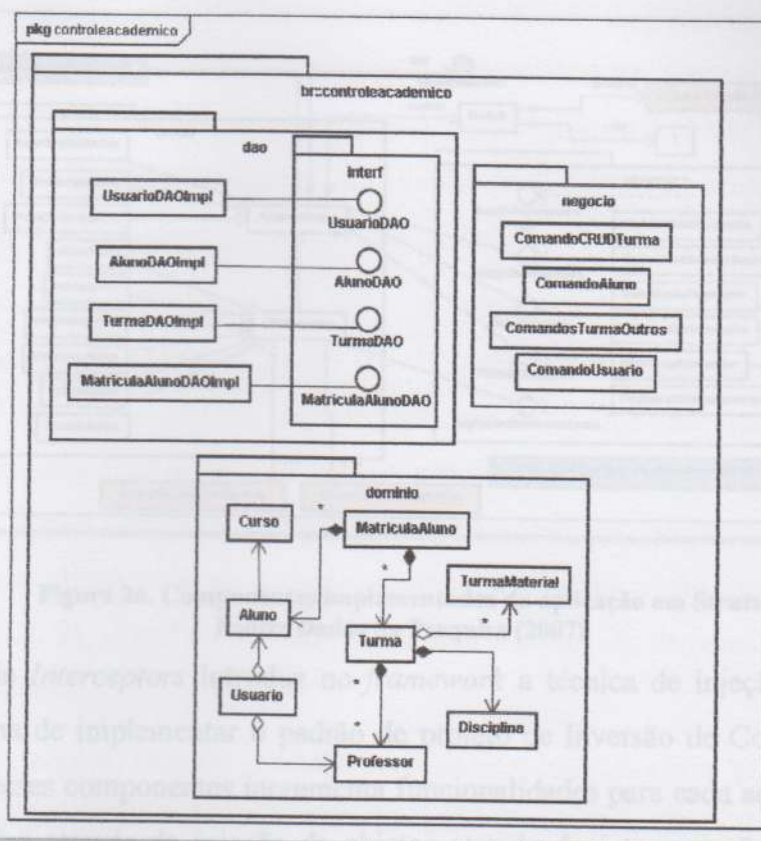


Figura 25. Camada de dados compartilhada pelas implementações das duas aplicações

Fonte: Dados da Pesquisa (2007)

5.2 IMPLEMENTAÇÃO EM STRUTS

O *framework* Apache Struts sofreu grandes alterações na evolução das suas primeiras versões 1.x para sua mais recente 2.x. A incorporação de recursos do WebWork modificaram consideravelmente a forma com que aplicações são desenvolvidas em comparação com as primeiras versões do Struts. No Capítulo 4, esses e outros aspectos técnicos da arquitetura desse *framework* são discutidos com mais detalhes. O tópico corrente tem o objetivo de apresentar o uso do Struts para implementar as aplicações que serviram como estudos de caso para o trabalho dessa monografia.

O diagrama de classes da Figura 26 apresenta uma visão geral dos principais componentes implementados nas aplicações (o Anexo 1.2 desse documento apresenta uma versão mais detalhada desse diagrama). Todo o desenvolvimento da nova versão do Struts é baseado no uso de dois tipos de componentes: as ações (do inglês *Actions*) e os interceptadores (do inglês *Interceptors*). É possível verificar no diagrama que tais componentes implementados estão distribuídos nos pacotes de mesmo nome.

compõem a plataforma JavaEE 5. Como *framework* “oficial” da plataforma, essa API possui diversas implementações distribuídas gratuita e comercialmente. Com o intuito de analisar apenas os recursos especificados na API, para o desenvolvimento da aplicação em JSF foi utilizada a implementação de referência da própria Sun. Com isso a compatibilidade com qualquer outra implementação é completamente garantida.

No caso das aplicações do JSF, os principais componentes que regem o desenvolvimento são os *Managed Beans* (ou *MBeans*) e os *UIComponents* (ilustrados na Figura 27). Os tipos mais comuns são definitivamente os *MBeans*. As operações desses componentes são invocadas na fase de acionamento da aplicação durante o ciclo de vida de uma requisição. Assim como no Struts, esses componentes são divididos entre *MBeans* de cadastro e *MBeans* simples. Os de cadastro se limitam a herdar as funcionalidades da classe abstrata *MBeanCRUD*, a qual é responsável por toda a invocação da camada de negócio para efetivar as operações.

Foi necessária a criação de três componentes de *interface* personalizados para incorporar novas funcionalidades não cobertas pelo *framework*. Esses componentes foram responsáveis pela gerência de paginação e ordenação de tabelas de listagem e outro para satisfazer o requisito de *upload* de arquivo.

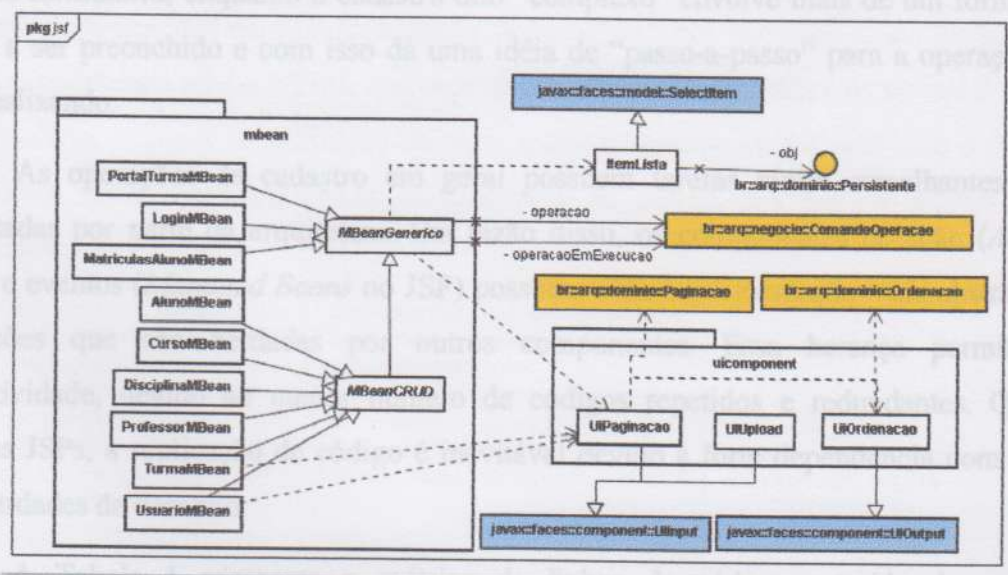


Figura 27. Componentes implementados da aplicação em JSF
Fonte: Dados da Pesquisa (2007)

6 AVALIAÇÃO DE PRODUTIVIDADE

A avaliação de produtividade no desenvolvimento de aplicações baseadas nos *frameworks* Struts e JSF foi realizada a partir da análise dos estudos de caso apresentados no capítulo anterior. Como já discutido, o desenvolvimento das aplicações de caráter experimental teve requisitos funcionais típicos do desenvolvimento de aplicações Web para tornar a utilidade das aplicações compatível com sistemas de informação de grande porte.

Neste capítulo são apresentadas cada análise feita sobre as funcionalidades destacadas em capítulos anteriores. Para cada funcionalidade é comparada a métrica de total de linhas de código (LOC – do inglês *Lines Of Code*) necessárias para sua implementação e também é realizada a indicação de qual framework melhor soluciona o problema.

6.1 OPERAÇÕES DE CADASTRO

Como apresentado nos capítulos anteriores, os casos de uso de uso relativos às operações de cadastro podem ser classificadas em simples e complexa. No cadastro simples a aquisição dos dados é realizada através da submissão dos dados por parte do usuário em um simples formulário, enquanto o cadastro dito “complexo” envolve mais de um formulário de dados a ser preenchido e com isso dá uma idéia de “passo-a-passo” para a operação que se está realizando.

As operações de cadastro em geral possuem tarefas muito semelhantes a serem executadas por parte da arquitetura. Em razão disso, os componentes de ação (*Actions* no Struts) e eventos (*Managed Beans* no JSF) possuem uma implementação mais abstrata dessas operações que são herdadas por outros componentes. Essa herança permite maior produtividade, devido ao menor número de códigos repetidos e redundantes. Quanto às páginas JSPs, a replicação de código é inevitável devido à forte dependência com os dados das entidades de domínio.

A Tabela 1 apresenta a métrica de linhas de código extraída das aplicações desenvolvidas em Struts e JSF especificamente sobre os arquivos responsáveis pela implementação dos cadastros simples e complexos. A tabela informa também o número de arquivos necessários para a implementação dessa funcionalidade e divide os resultados pelos tipos de arquivos (Java, JSP e XML) relacionados diretamente com o framework referente.

Tabela 1. Total de linhas de código para as operações de cadastro

Tipo	Struts		JSF	
	Arquivos	LOC	Arquivos	LOC
Classes Java	12	885	11	844
Páginas JSP	17	441	19	911
Configurações XML	6	203	2	209
Total	35	1529	32	1964

Os valores apresentados na tabela indicam menor produtividade na produção dos arquivos da aplicação JSF. O total de linhas de código de classes Java e arquivos de configuração são bem próximos em ambas as aplicações, por outro lado total de linhas das páginas JSPs da aplicação do Struts é menos da metade do valor na aplicação em JSF. Isso implica que a mesma formatação dos mesmos tipos de dados se torna mais trabalhoso usando os recursos do JSF do que os do Struts.

Um simples exemplo dessa diferença pode ser notado em trechos dos formulários de cadastro da entidade Disciplina ilustrados nas figuras 28 e 29. Na Figura 28 é apresentado um do código do arquivo JSP responsável pela formatação do formulário para cadastro de disciplina através das tags fornecidas pelo Struts2. Uma grande vantagem do uso dessas tags é o recurso de temas, que permite formatação “automática”, por exemplo, nesse formulário não é preciso codificar a tabela HTML que organiza os campos de entrada e os nomes dos campos do formulário. A partir da definição do tema usado o framework formata todo o formulário como uma tabela, já organizando os campos e os nomes dos campos (através da tag `s:textfield`).

```

3 <s:form action="admin/Disciplina/submeterDadosForm.action"
4   method="post" cssClass="form" theme="ajax" >
5   <caption>Cadastro de Disciplina</caption>
6   <s:textfield label="Nome" name="obj.nome">/s:textfield>
7   <s:textfield label="Código" name="obj.codigo">/s:textfield>
8   <s:textfield label="CH" name="obj.cargaHoraria">/s:textfield>
9   <s:textarea cols="40" rows="4" label="Ementa" name="obj.ementa">/s:textarea>
10  <tfoot><tr>
11    <td colspan="2">
12      <s:submit value="{botaoConfirmacao}" theme="simple">/s:submit>
13      <s:submit value="CANCELAR" method="cancelar" theme="simple">/s:submit>
14    </td>
15  </tr></tfoot>
16</s:form>

```

Figura 28. Trecho do código do formulário JSP usando Struts para cadastro de disciplina

Fonte: Dados da Pesquisa (2007)

Enquanto a Figura 28 consegue apresentar em 14 linhas todas tags necessárias para

formatar o formulário de cadastro com quatro campos da entidade Disciplina, a Figura 29 ilustra apenas um trecho do código para o mesmo formulário com tags de JSF. Em 18 linhas só é possível visualizar a formatação para dois campos da entidade. A diferença de total de linhas de código entre os arquivos JSPs é muito grande, muito em razão do excesso de tags e recursos oferecidos pelo JSF.

```

5 <h:form id="form">
6   <h:panelGrid columns="2" styleClass="form">
7     <f:facet name="caption">
8       <h:outputText value="CADASTRO DE DISCIPLINA"/></h:outputText>
9     </f:facet>
10    <h:outputText value="Código: "/></h:outputText>
11    <h:panelGroup layout="block">
12      <h:inputText id="codigo" size="7" value="#{disciplinaBean.obj.codigo}"
13        required="true" label="Código"/></h:inputText>
14      <h:message for="codigo" errorClass="erros"/></h:message>
15    </h:panelGroup>
16
17    <h:outputText value="Nome: "/></h:outputText>
18    <h:panelGroup layout="block">
19      <h:inputText id="nome" size="40" value="#{disciplinaBean.obj.nome}"
20        required="true" label="Nome"/></h:inputText>
21      <h:message for="nome" errorClass="erros"/></h:message>
22    </h:panelGroup>

```

Figura 29. Trecho do código do formulário JSP usando JSF para cadastro de disciplina
Fonte: Dados da Pesquisa (2007)

A diferença total dos valores de LOC para as duas aplicações pode ser considerada importante. Com uma diferença maior que 30% entre os dois valores, é possível indicar que usar Struts para o desenvolvimento de casos de uso de cadastro simples se apresenta mais produtivo que usar JSF. Vale ressaltar que essa análise é baseada no uso “comum” das tecnologias, ou seja, muitos recursos das tecnologias foram utilizados (como validação automática dos dados do formulário), entretanto não foram necessárias abordagens mais complexas. Dessa forma tentou-se simular o desenvolvimento mais próximo do que é mais usado pelo mercado.

6.2 UPLOAD DE ARQUIVO

Para o carregamento remoto de arquivos através de aplicações Web a arquitetura deve prover serviços especializados para adquirir o conteúdo desse arquivo e armazená-lo no servidor, seja em sistema de arquivos ou persisti-lo em uma base de dados. Essa é uma funcionalidade cada vez mais comum em aplicações Web e se faz necessário que *frameworks* para essas aplicações forneçam recursos para facilitar esses casos de uso.

A implementação dessa funcionalidade é bastante diferente nos frameworks dessa

avaliação. No Struts o suporte para fazer upload de arquivo é nativo do *framework*, se faz necessário apenas criar atributos na classe de ação para ter acesso ao nome e conteúdo do arquivo submetido pelo usuário. No caso do JSF, essa funcionalidade tem não suporte nativo na versão mais recente do *framework* e obriga o desenvolvedor a criar componentes de GUI que abstraia esse carregamento para os *Managed Beans*. Essa abstração é conseguida através da implementação de componente de entrada de dados para renderizar o componente HTML, uma tag para formatar esse componente em uma página JSP e um filtro responsável pelo carregamento e persistência do conteúdo do arquivo.

Todo esse custo imposto pelo JSF é refletido nos valores totais de linhas de código nas duas aplicações para implementar essa funcionalidade. A Tabela 2 apresenta esses valores e demonstra a grande diferença entre os valores totais. Enquanto no Struts o desenvolvedor só precisa acessar o serviço já oferecido pelo *framework*, no JSF ele é obrigado a implementar esse serviço.

Tabela 2. Total de linhas de código para as operações de upload de arquivo

Tipo	Struts	JSF
	LOC	LOC
Classes Java	5	272
Configurações XML	1	31
Total	6	303

A Figura 30 apresenta o trecho de código da aplicação desenvolvida em Struts que representa as cinco linhas de código Java presentes em um método de uma classe de ação responsável pelo upload de arquivo na aplicação. Todo processamento de conversão do conteúdo do arquivo submetido do lado do cliente em um objeto do tipo File da API de manipulação de arquivos da plataforma Java foi abstraído. Enquanto no JSF, o desenvolvedor precisa implementar esse e outros processamentos para atender a esse requisito.

```

131 ServletContext servletContext = (ServletContext) ActionContext.
132     getContext().get(StrutsStatics.SERVLET_CONTEXT);
133 String realPath = servletContext.getRealPath("upload/" + material.getArquivo());
134 File theFile = new File(realPath);
135 FileUtils.copyFile(arquivo, theFile);

```

Figura 30. Trecho de código usando Struts para carregar arquivo

Fonte: Dados da Pesquisa (2007)

No caso do atendimento ao requisito da funcionalidade de upload de arquivo, o uso do *framework* Struts se mostrou muito mais produtivo devido ao seu suporte nativo à funcionalidade. A diferença entre o total de linhas de código produzidos para prover o mesmo serviço é enorme, e a não ser que sejam utilizados componentes JSF de terceiros (que

garantam a compatibilidade entre servidores de aplicação), o uso do Struts para essa funcionalidade é muito mais recomendável.

6.3 PAGINAÇÃO E CLASSIFICAÇÃO DE TABELAS

Listagem de dados em páginas HTML normalmente são renderizadas em um formato tabular através das marcações de tabela do próprio HTML. Entretanto o carregamento de uma listagem muito grande pode comprometer a usabilidade da aplicação devido a um baixo desempenho ou uma baixa capacidade da conexão utilizada. Dessa forma, listagens muito grande a serem exibidas em tabelas já devem ser apresentadas ao usuário com a opção do usuário classificar os dados pelas colunas das tabelas de forma ascendente e descendente. E mais importante, os resultados a serem exibidos devem ser decompostos em “páginas” e serem exibidos por partes, permitindo ao usuário “navegar” entre essas páginas para visualizar os outros resultados.

A implementação e classificação em tabelas depende do suporte à renderização de tabelas dinâmicas a partir de coleções de objetos JavaBeans, um recurso a ser disponibilizado pelo *framework*. No caso do JSF esse suporte é nativo do *framework*, facilitou em apenas incorporar a funcionalidade de paginação e classificação. Por outro lado, no Struts não possui nenhum suporte nesse sentido, e foi preciso a criação de uma tag que renderizasse a tabela a partir de uma coleção de objetos, e ainda permitisse as funcionalidades de paginação e classificação.

A Tabela 3 apresenta o total de linhas de código produzidas para atender às funcionalidades de paginação e classificação de tabelas em ambas as aplicações. É possível perceber que a diferença entre os totais de linhas para os arquivos de configuração são bem semelhantes e os totais das linhas produzidas em classes Java também não são tão diferentes, entretanto é a maneira de usar essa funcionalidade nas páginas JSPs que faz a diferença. O total de linhas nas JSPs do JSF é mais que o dobro do valor das JSPs com Struts.

Tabela 3. Total de linhas de código para as operações de paginação e classificação de tabelas

Tipo	Struts	JSF
	LOC	LOC
Classes Java	544	486
Páginas JSP	146	337
Configurações XML	101	112
Total	791	935

A diferença de uso desse serviço implementado pelas classes Java nas páginas JSPs pode ser observado nas figuras 31 e 32. Na Figura 31 é possível observar que a renderização da tabela e as funcionalidades de classificação e paginação das colunas são exclusivamente usadas a partir da criação da tag Tabela. Onde a sua sub-tag Coluna fica responsável por acessar os atributos dos objetos em cada iteração na coleção e renderizar os links para classificação de cada uma. E ainda no final de cada tabela são renderizadas as páginas possíveis de serem visualizadas a partir da quantidade de registros retornados pela consulta.

```

14 <tag:tabela formId="buscaAlunos" colecao="{resultadosBusca}" tamanhoPagina="4"
15   caption="Alunos Encontrados" linhaImpar="linhaImpar" linhaPar="linhaPar" width="90%">
16   <tag:coluna titulo="ID" propriedade="id" width="5%" />
17   <tag:coluna titulo="MATRICULA" propriedade="matricula" />
18   <tag:coluna titulo="NOME" propriedade="nome" />
19   <tag:coluna titulo="CURSO" propriedade="curso.nome" />
20   <tag:coluna action="detalhar.action?paramObjId={id}" width="10%">Detalhar</tag:coluna>
21   <tag:coluna action="atualizar.action?paramObjId={id}" width="2%">
22     
23   </tag:coluna>
24   <tag:coluna action="remover.action?paramObjId={id}" width="2%">
25     
26   </tag:coluna>
27 </tag:tabela>

```

Figura 31. Trecho de código da formatação de uma tabela em Struts

Fonte: Dados da Pesquisa (2007)

Na implementação em JSF, a implementação da renderização de tabelas a partir de coleções de objetos já provido pelo framework através do componente de GUI DataTable. Entretanto, esse componente não suporta a classificação e paginação dos seus dados nativamente, para isso foi criada as tags Paginacao e Ordenacao. A tag responsável pela paginação renderiza a lista páginas no fim da tabela permitindo ao usuário a navegação entre as páginas. Enquanto a tag de ordenação renderiza um link no título de cada coluna da tabela permitindo a classificação tanto ascendente quanto descendente dos registros da tabela.

```

42 <h:dataTable rowClasses="linhaImpar, linhaPar" width="60%" var="item"
43 value="#{alunoBean.resultadoBusca}" rendered="#{not empty alunoBean.resultadoBusca}"
44 <f:facet name="caption">
45 <h:outputText value="Resultado da Busca de Alunos"/>
46 </f:facet>
47 <h:column>
48 <f:facet name="header">
49 <tag:ordenacao propriedade="nome" titulo="ALUNO"/>
50 </f:facet>
51 <h:outputText value="#{item}" />
52 </h:column>
53 <h:column>
54 <f:facet name="header">
55 <tag:ordenacao propriedade="curso.nome" titulo="CURSO"/>
56 </f:facet>
57 <h:outputText value="#{item.curso}" />
58 </h:column>
59 <h:column>
60 [<h:commandLink action="#{alunoBean.detallar}">detallar</h:commandLink>]
61 </h:column>
62 <h:column>
63 [<h:commandLink action="#{alunoBean.atualizar}">atualizar</h:commandLink>]
64 </h:column>
65 <h:column>
66 [<h:commandLink action="#{alunoBean.remover}">remover</h:commandLink>]
67 </h:column>
68 </h:dataTable>
69 <tag:paginacao tamanhoPagina="10" totalPaginasExibicao="5" acao="#{alunoBean.buscar}" />

```

Figura 32. Trecho de código da formatação de uma tabela em JSF
Fonte: Dados da Pesquisa (2007)

A diferença total entre as linhas de código exigidas por cada *framework* é de apenas 15%. E apesar dos resultados dessa métrica serem desfavoráveis à aplicação baseada em JSF, o uso do recurso nativo do componente DataTable pode ser um diferencial no quesito de renderização de resultados em formato tabular. A produtividade alcançada com a criação de simples tags com Struts se limita à utilidade desse renderização, entretanto mesmo o uso de DataTable indique uma produtividade um pouco menor nesse caso, seu uso ainda pode ser mais indicado em razão de ser um recurso muito poderoso e é totalmente integrado à outras funcionalidades do *framework*.

6.4 OPEN SESSION IN VIEW

O problema da sessão de conexão com a base de dados aberta ainda na renderização dos componentes da visão pode ser solucionado na camada Web da aplicação através de configurações específicas dos *frameworks*. Esse problema é identificado caso a conexão com o banco de dados seja fechada ainda na execução da operação, e não esteja mais disponível no momento da renderização da visão se algum objeto precisar acessar o banco para recarregar outro objeto acoplado a ele. Por esse motivo, esse problema é normalmente resolvido em aplicações Web através de operações para abrir e fechar a sessão, respectivamente no início e no final do ciclo de vida de uma requisição.

Solucionar o problema da sessão aberta na visão acaba sendo tarefa simples para

aplicações baseadas tanto em Struts quanto em JSF. Ambos os *frameworks* disponibilizam recursos para o desenvolvedor executar operações customizadas em qualquer momento do ciclo de vida da requisição HTTP. No Struts isso é possível através do processamento dos Interceptors, que são executados antes da invocação da ação requisitada e são re-executados depois da renderização da resposta. Enquanto no JSF o componente responsável pelo acesso ao ciclo de vida da aplicação é o PhaseListener, nele é possível executar uma operação antes e depois de qualquer fase do ciclo do JSF.

Os totais de linhas produzidas nas aplicações e apresentados na Tabela 4, indicam as implementações dos Interceptors e dos PhaseListeners. Estes são os componentes responsáveis por abrir e fechar a sessão de conexão com o banco antes da invocação da operação desejada e depois da renderização da visão em cada requisição.

Tabela 4. Total de linhas de código para solucionar o problema de Open Session In View

Tipo	Struts	JSF
	LOC	LOC
Classes Java	77	81
Configurações XML	3	4
Total	80	85

A pequena diferença entre os valores totais indica que em ambos os *frameworks* a solução para o problema da sessão aberta na visão são compatíveis no que diz respeito à produtividade. Dessa forma, nenhuma das soluções obteve impacto considerável maior que a outra na produtividade do desenvolvimento das aplicações.

- Operações de Cadastro: a produtividade do Struts é superior em meio da formatação das páginas JSPs. A diferença de 30% a menos no valor total de linhas de código para a aplicação baseada no Struts se deve principalmente à simplicidade do uso das tags de formatação do *framework*.
- Upload de Arquivo: atender a este requisito com o Struts foi bem mais produtivo devido ao suporte nativo a essa funcionalidade, enquanto no JSF foi preciso desenvolver componentes GUI para suportá-la.
- Paginação e Classificação de Tabelas: os resultados das métricas apontam uma pequena vantagem na produtividade do uso do Struts, estrutural e suporte nativo

7 CONCLUSÕES

O desenvolvimento de aplicações Web é comumente baseado no uso de *frameworks* de aplicação que fornecem uma infra-estrutura básica de serviços semi-prontos baseados no padrão arquitetural MVC. Esses *frameworks* fornecem recursos que auxiliam a criação de componentes específicos para as camadas de visão, e principalmente, do controlador. Ao adotar o uso de algum desses *frameworks* certamente impacta na produtividade do desenvolvimento da aplicação em comparação com projetos que preferem criar tudo personalizado. Entretanto o funcionamento de *frameworks* de naturezas diferentes pode refletir em produtividade ainda maior.

O trabalho dessa monografia propõe uma avaliação da produtividade no desenvolvimento de aplicações baseadas nos dois principais *frameworks* Web da plataforma Java usados no mercado, Apache Struts e JavaServer Faces. As análises foram realizadas sobre duas aplicações desenvolvidas em caráter experimental com ambos os frameworks. O desenvolvimento dessas aplicações foi guiado pelo atendimento a quatro dos principais requisitos tipicamente definidos para aplicações Web em sistemas de informação: Operações de Cadastro, Upload de Arquivo, Paginação e Classificação de Tabelas e solução para o problema do *Open Session In View*.

O Capítulo 6 apresentou o resultado da métrica de linhas de código extraída das aplicações desenvolvidas como estudo de caso. A avaliação foi realizada a partir das análises desses resultados e trechos de código apresentados no capítulo anterior. A comparação entre as tecnologias foi feita sobre cada um dos requisitos definidos. A seguir essas análises sobre cada requisito são resumidas:

- Operações de Cadastro: a produtividade do Struts é superior em razão da formatação das páginas JSPs. A diferença de 30% a menos no valor total de linhas de código para a aplicação baseada no Struts se deve principalmente à simplicidade do uso das tags de formatação do *framework*;
- Upload de Arquivo: atender a esse requisito com o Struts foi bem mais produtivo devido ao suporte nativo a essa funcionalidade, enquanto no JSF foi preciso desenvolver componentes GUI para suportá-la;
- Paginação e Classificação de Tabelas: os resultados das métricas apontam uma pequena vantagem na produtividade do uso do Struts, entretanto o suporte nativo

do JSF à representação tabular através do DataTable se torna muito mais poderoso do que o uso de simples tags de JSP;

- *Open Session In View*: para solucionar o problema da sessão da conexão do banco aberta ainda na visão, a produtividade com ambas as tecnologias foi equivalente.

A Tabela 5 reúne os valores gerais de linhas de código das aplicações baseadas em Struts e JSF. As últimas colunas da tabela apresentam os valores totais das linhas de código para classes Java, páginas JSP, arquivos de configuração de XML e formatações de CSS. É possível perceber que a grande diferença entre as duas aplicações está no total de linhas das páginas JSPs. Isso pode ser explicado pela grande quantidade de recursos que o JSF oferece através de suas poderosas tags, que ao mesmo tempo proporciona um desenvolvimento rico da interface com usuário, mas também diminui a produtividade da codificação dos arquivos JSPs.

Tabela 5. Resultado geral de linhas de código das aplicações baseadas no Struts e JSF

Tipo	Arquivos		Branças		Comentários		Código		Total	
	Struts	JSF	Struts	JSF	Struts	JSF	Struts	JSF	Struts	JSF
Java	64	66	943	936	1041	1316	3486	3545	5470	5797
JSP	34	33	72	89	3	9	799	1338	874	1436
XML	10	5	66	49	0	9	338	313	404	371
CSS	1	1	14	14	0	0	137	137	151	151
Total	109	105	1095	1088	1044	1334	4760	5333	6899	7755

A partir das análises realizadas o desenvolvimento da aplicação baseada em Struts se mostrou mais produtivo do que o da aplicação em JSF. O maior fator de diferença, como já citado, foi o uso das tags nas páginas JSPs. Os gráficos da Figura 33 indicam as pequenas diferença entre os valores do total das linhas de código para arquivos Java e XML, e a diferença de 40% entre os valores de total de linhas para arquivos JSPs. Entretanto, o resultado dessa avaliação não pode desmerecer o valor das diversas funcionalidades fornecidas pelo JavaServer Faces. Um dos maiores motivos para essa verbosidade das páginas JSPs nas aplicações JSF se deve ao fato de que originalmente ele foi criado para auxiliar o desenvolvimento de aplicações Web baseado no uso de ferramentas RAD, e justamente por isso é preciso tanto controle e tanta “burocracia” nas formatações. Porque normalmente esses códigos devem ser criados e entendidos por ferramentas, e não por pessoas.

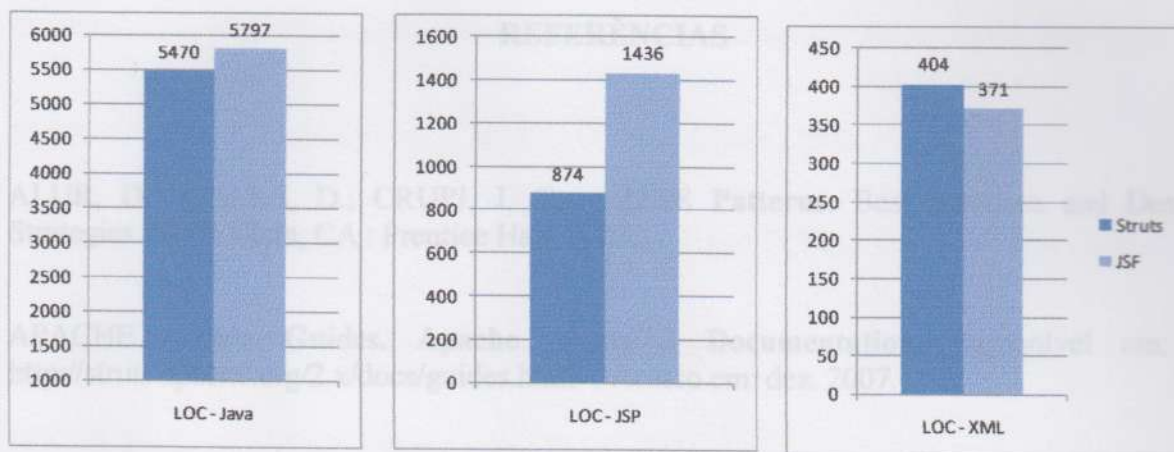


Figura 33. Gráficos de comparação entre os valores de LOC para os arquivos Java, JSP e XML
 Fonte: Dados da Pesquisa (2007)

A avaliação realizada e discutida no trabalho dessa monografia atingiu seu principal objetivo, e permitiu que as diferenças entre os *frameworks* Struts e JSF pudessem ser analisadas em níveis mais profundos. A avaliação possui a contribuição importante que é uma análise objetiva sobre um fator muito importante para o desenvolvimento de aplicações, a produtividade.

Mesmo com os bons resultados apresentados, a avaliação realizada nesse trabalho deve fazer parte de uma avaliação maior que analise outros aspectos além das características de qualidade. Trabalhos futuros podem ter o objetivo de fazer o mesmo tipo de avaliação para outras características como manutenibilidade e confiabilidade, ou então analisar outros *frameworks* Web. A partir da existência de mais avaliações como esta, mais valor será agregado à comparação entre os dois *framework*, dessa forma contribuindo cada vez mais para fundamentar a escolha entre eles.

CHRISTENSEN, H. Frameworks: putting design patterns into perspective. Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education, 2004.

CRANE, D.; PASCCARELLO, E. Ajax in Action. Greenwich : Manning, 2006.

DEMICHIEL, L.; KRITH, M. JSR 220: Enterprise JavaBeans 3.0 Specification. Java Specification Requests. Disponível em: <<http://jcp.org/en/jsr/detail?id=220>>. Acesso em: Out 2007.

DOEDERLEIN, O. 39. Rio de Janeiro : DevMedia, 2007. Java Magazine, Vol. 4.

DOEDERLEIN, O. 1. Rio de Janeiro : DevMedia, 2004. Java Magazine, Vol. 1.

REFERÊNCIAS

- ALUR, D., MALKS, D.; CRUPI, J. **Core J2EE Patterns: Best Practices and Design Strategies**. Santa Clara, CA : Prentice Hall, 2003.
- APACHE. **Struts Guides. Apache Struts 2 Documentation**. Disponível em: <<http://struts.apache.org/2.x/docs/guides.html>>. Acesso em: dez. 2007.
- BALL, J.; et al. **The Java EE 5 Tutorial**. Santa Clara : Sun Microsystems, 2006.
- BOOCH, G.; RUMBAUGH, J. e JACOBSON, I. **UML: Guia do Usuário**. Rio de Janeiro : Campus, 2005.
- BURNS, E.; KITAIN, R. **JSR 252: JavaServer Faces 1.2 Specification**. Java Specification Requests. Disponível em: <<http://jcp.org/en/jsr/detail?id=252>>. Acesso em: dez 2007.
- BURNS, E.; MCCLANAHAN, C. **JSR 127: JavaServer Faces Specification**. Java Specification Requests. Disponível em: <<http://jcp.org/en/jsr/detail?id=127>>. Acesso em: dez 2007.
- CASTRO, E. **Perl and CGI for the World Wide Web**. Berkeley, CA : Peachpit Press, 1998.
- CHOI, W.; et al. **Beginning PHP4 (Programmer to Programmer)**. Indianapolis : Wrox, 2000.
- CHRISTENSEN, H. **Frameworks: putting design patterns into perspective**. Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education. 2004.
- CRANE, D.; PASCCARELLO, E. **Ajax in Action**. Greenwich : Manning, 2006.
- DEMICHIEL, L.; KEITH, M. **JSR 220: Enterprise JavaBeans 3.0 Specification**. Java Specification Requests. Disponível em: <<http://jcp.org/en/jsr/detail?id=220>>. Acesso em: dez 2007.
- DOEDERLEIN, O. 39, Rio de Janeiro : DevMedia, 2007, Java Magazine, Vol. 4.
- DOEDERLEIN, O. 1, Rio de Janeiro : DevMedia, 2004, Java Magazine, Vol. 1.

DUDNEY, B.; et al. **Mastering JavaServer Faces**. Indianápolis : Wiley Publishing, 2004.

ELLIOTT, J.; et al. **Java Swing**. 2ª ed. s.l. : O'Reilly Media, 2002.

EVJEN, B.; et al. **CGI: Internet Programming in C++ and C**. New York : Prentice Hall, 1997.

FLANAGAN, D. **Java O Guia Essencial**. Porto Alegre : Bookman, 2006.
Disponível em: <<http://jcp.org/en/jar/detail?id=134>>. Acesso em: dez 2007.

FOO, S. **Beginning WAP**. Indianapolis : Wrox, 2000.

PERLIGRI-LLOPART, R.; ROTH, M. JSR 152: JavaServer Pages 2.0 Specification. Java Specification Requests. Disponível em: <<http://jcp.org/en/jsr/detail?id=152>>. Acesso em: dez 2007.

FOWLER, M. **Padrões de Arquitetura de Aplicações Corporativas**. Porto Alegre : Bookman, 2003.

FOWLER, M. **Inversion of Control Containers and the Dependency Injection Pattern**. Disponível em: <<http://martinfowler.com/articles/injection.html>>. Acesso em: dez 2007.

ROSEINDIA. Rosetalia. Disponível em: <<http://www.roseindia.net/struts/struts2/>>. Acesso em: dez 2007.

GAMMA, E.; et al. **Design Patterns**. Massachussetts : Addison-Wesley, 1995.

GOERZEN, J. **Foundations of Python Network Programming**. Berkley, CA : Apress, 2004.

SAM-BODIN, B. **Regulating PYTHON: Lightweight Java Web Development Using Plain Old Java Objects**. Spring, Libraries and Tutorials. Berkeley : Apress, 2006.

GUNDAVARAM, S. **CGI Programming on the World Wide Web**. Sebastopol, CA : O'Reilly, 1996.

SOARES, L.; LEMOS, G.; COLCHER, S. **Redes de Computadores: das LANs, MANs e WANs até Internet**. Rio de Janeiro : Campus, 1995.

HIGHTOWER, R. **JSF for nonbelievers: The JSF application lifecycle**. IBM Developer Works. Disponível em: <<http://www-128.ibm.com/developerworks/library/j-jsf2/>>. Acesso em: dez 2007.

LEE, I. **Software Engineering**. 2ª ed. s.l. : Addison-Wesley, 2006.

HUNTER, D.; et al. **The Unified Software Development**. Massachussetts : Addison-Wesley, 1999.

LARMAN, C. **Utilizando UML e Padrões**. Porto Alegre : Bookman, 2003.

LERDORF, R.; TATROE, K.; MACINTYRE, P. **Programming PHP**. s.l. : O'Reilly, 2006.

LIGHTBODY, P. e CARREIRA, J. **WebWork in Action**. Greenwich : Manning, 2005.

MANN, K. **JavaServer Faces In Action**. Greenwich : Manning, 2005.

MCCONNELL, S. **Rapid Development**. s.l. : Microsoft Press, 1996.

MENDES, A. **Arquitetura de Software**. Rio de Janeiro : Campus, 2002.

MINTER, D.; LINWOOD, J. **Pro Hibernate 3**. Berkeley : Apress, 2005.

MORDANI, R. **JSR 154**: Java Servlet 2.4 Specification. Java Specification Requests. Disponível em: <<http://jcp.org/en/jsr/detail?id=154>>. Acesso em: dez 2007.

PELEGRI-LLOPART, E.; ROTH, M. **JSR 152**: JavaServer Pages 2.0 Specification. Java Specification Requests. Disponível em: <<http://jcp.org/en/jsr/detail?id=152>>. Acesso em: dez 2007.

PRESSMAN, R. **Engenharia de Software**. 6ª ed. São Paulo : Makron Books, 2006.

ROSEINDIA. **RoseIndia**. Disponível em: <<http://www.roseindia.net/struts/struts2/>>. Acesso em: dez 2007.

ROUGHLEY, I. **Starting Struts2**. s.l. : C4Media, 2006.

SAM-BODDEN, B. **Beginning POJOs**: Lightweight Java Web Development Using Plain Old Java Objects in Spring, Hibernate, and Tapestry. Berkeley : Apress, 2006.

SOARES, L.; LEMOS, G.; COLCHER, S. **Redes de Computadores**: das LANs, MANs e WANs às Redes ATM. Rio de Janeiro : Campus, 1995.

SOMMERVILLE, I. **Software Engineering**. 8ª ed. s.l. : Addison Wesley, 2006.

SUN MICROSYSTEM. **The Java Tutorials**. Disponível em: <<http://java.sun.com/docs/books/tutorial/>>. Acesso em: dez 2007.

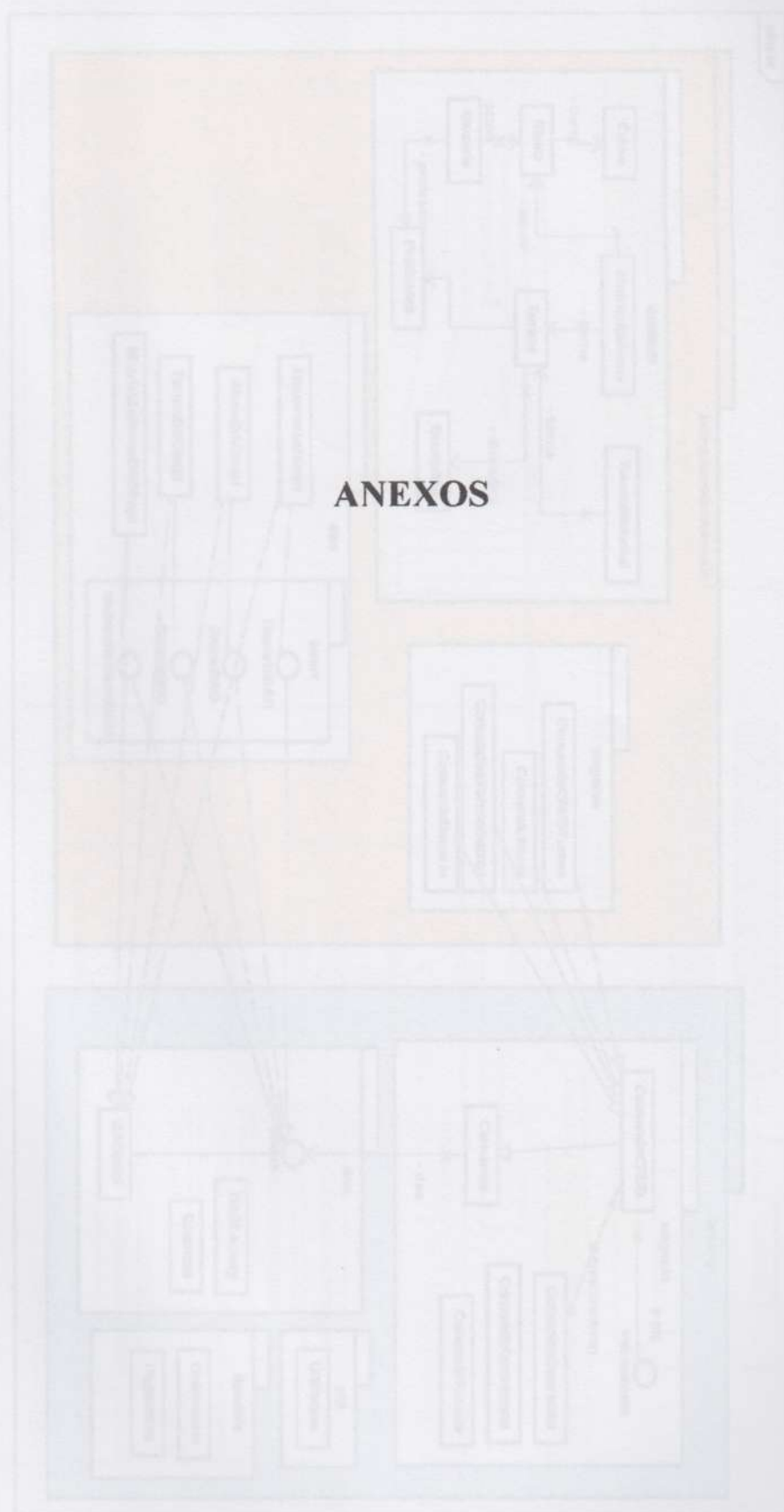
TURNER, J.; BEDELL, K. **Struts Kick Start**. s.l. : Sams Publishing, 2002.

WALL, L.; CHRISTIANSEN, T.; ORWANT, J. **Programming Perl**. s.l. : O'Reilly, 2005.

WALLS, C.; BREIDENBACH, R. **Spring in Action**. Greenwich : Manning, 2005.

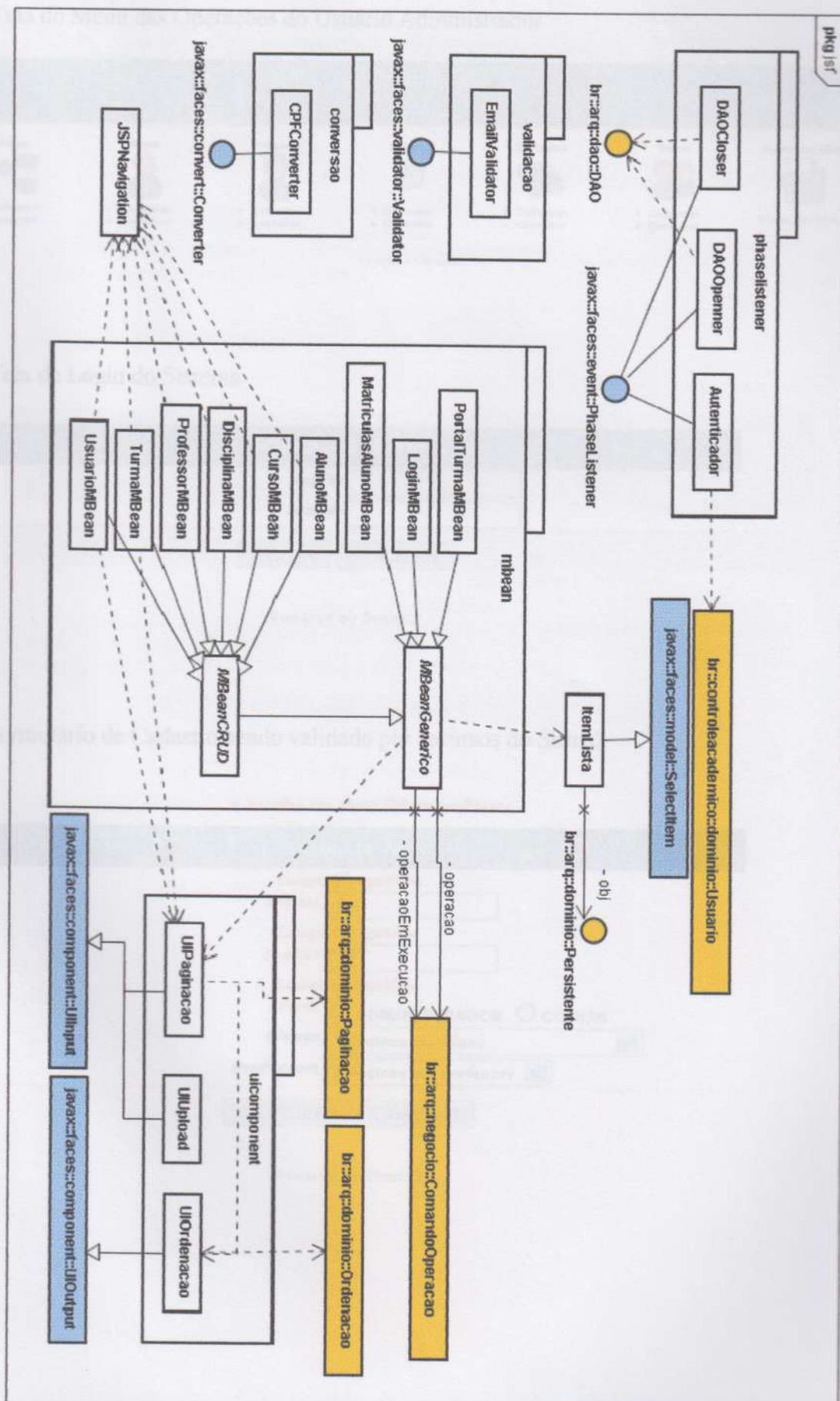
ANEXOS A – Diagramas De Classes

A-1 Arquitetura da Aplicação – Camada de Dados



[illegible]

A-3 Arquitetura da Aplicação JSF



ANEXOS B – Telas Das Aplicações

B-1 Tela do Menu das Operações do Usuário Administrador

BENJAMIN BEDREGAL - adm

PRINCIPAL SAIR

Curso
• Cadastrar
• Consultar

Aluno
• Cadastrar
• Consultar

Professor
• Cadastrar
• Consultar

Usuário
• Cadastrar
• Consultar

Disciplina
• Cadastrar
• Consultar

Turma
• Cadastrar
• Consultar

Matricular Aluno

Powered by Struts2

B-2 Tela de Login do Sistema

ENTRAR NO SISTEMA

Login:

Senha:

ENTRAR CANCELAR

Powered by Struts2

B-3 Formulário de Cadastro sendo validado por recursos do Struts2

• Escolha um aluno OU um professor

CADASTRO DE ALUNO

Campo Obrigatório
Login:

Campo Obrigatório
Senha:

Campo Obrigatório
Tipo: ☐ ADMINISTRADOR ☐ COMUM

Aluno:

Professor:

CADASTRAR CANCELAR

Powered by Struts2

B-4 Telas da Operação de Matrícula *visualizada por um professor*

BENJAMIN BEDREGAL - adm

PRINCIPAL SAIR

MATRICULAR ALUNO

Matrícula: 20070000

Aluno: ANDRE M DANTAS

Turma: DIM0032 - DESENVOLV.DE SOFTWARE DE CONTROLE EM TEMPO-REAL(60 hrs) - IVAN SARAIVA, 

ADICIONAR TURMA 

Turmas Escolhidas

DIM0326 - ALGORITMOS E ESTRUTURAS DE DADOS I(60 hrs) - ANDRE MAURICIO, PAULO PIRES, 

DIM0440 - INTELIGENCIA ARTIFICIAL(60 hrs) - ANDRE MAURICIO, 

MATRICULAR ALUNO CANCELAR

Powered by Struts2

BENJAMIN BEDREGAL - adm

PRINCIPAL SAIR

CONFIRMAÇÃO DE MATRÍCULAS DO ALUNO

Aluno: 20070000 - ANDRE M DANTAS

TURMAS MATRICULADAS

DIM0326 - ALGORITMOS E ESTRUTURAS DE DADOS I(60 hrs) - ANDRE MAURICIO, PAULO PIRES,

DIM0440 - INTELIGENCIA ARTIFICIAL(60 hrs) - ANDRE MAURICIO,

CONFIRMAR MATRÍCULAS  CANCELAR OUTRAS TURMAS

Powered by Struts2

ANEXOS C – Total De Linhas De Código

C-1 LOC da aplicação baseada em Struts

Arquivo	Bra	Com	Cod	Total
/arq/br/arq/dao/Conexao.java	10	7	34	34
/arq/br/arq/dao/DAO.java	20	7	29	56
/arq/br/arq/dao/DAOFactory.java	7	0	30	37
/arq/br/arq/dao/DAOImpl.java	37	7	190	234
/arq/br/arq/dao/DAOImplClass.java	4	5	10	19
/arq/br/arq/dao/hibernate.cfg.xml	3	0	36	39
/arq/br/arq/dominio/Ordenacao.java	12	13	38	63
/arq/br/arq/dominio/Paginacao.java	19	35	48	102
/arq/br/arq/dominio/Persistente.java	4	7	6	17
/arq/br/arq/excecoes/Excecao.java	6	20	15	41
/arq/br/arq/excecoes/ExcecaoDAO.java	5	16	13	34
/arq/br/arq/excecoes/ExcecaoService.java	4	13	10	27
/arq/br/arq/negocio/Comando.java	11	5	25	41
/arq/br/arq/negocio/ComandoCRUD.java	13	0	37	50
/arq/br/arq/negocio/ComandoFacade.java	7	0	20	27
/arq/br/arq/negocio/ComandoOperacao.java	12	5	43	60
/arq/br/arq/negocio/ComandoOperacoes.java	8	7	13	28
/arq/br/arq/negocio/OperacaoDescricao.java	4	7	10	21
/arq/br/arq/util/UtilHelper.java	7	7	40	54
/controleacademico/br/controleacademico/ComandoOperacoesCA.java	14	0	56	70
/controleacademico/br/controleacademico/dao/AlunoDAOImpl.java	4	7	9	20
/controleacademico/br/controleacademico/dao/interf/AlunoDAO.java	4	5	8	17
/controleacademico/br/controleacademico/dao/interf/MatriculaAlunoDAO.java	5	5	10	20
/controleacademico/br/controleacademico/dao/interf/TurmaDAO.java	9	5	20	34
/controleacademico/br/controleacademico/dao/interf/UsuarioDAO.java	6	5	12	23
/controleacademico/br/controleacademico/dao/MatriculaAlunoDAOImpl.java	5	7	17	29
/controleacademico/br/controleacademico/dao/TurmaDAOImpl.java	11	7	76	94
/controleacademico/br/controleacademico/dao/UsuarioDAOImpl.java	9	7	38	54
/controleacademico/br/controleacademico/dominio/Aluno.java	41	105	138	284
/controleacademico/br/controleacademico/dominio/Curso.java	14	43	62	119
/controleacademico/br/controleacademico/dominio/Disciplina.java	24	68	88	180
/controleacademico/br/controleacademico/dominio/MatriculaAluno.java	46	108	163	317
/controleacademico/br/controleacademico/dominio/Professor.java	21	60	80	161
/controleacademico/br/controleacademico/dominio/Turma.java	47	112	188	347
/controleacademico/br/controleacademico/dominio/TurmaMaterial.java	30	78	110	218
/controleacademico/br/controleacademico/dominio/Usuario.java	32	77	122	231
/controleacademico/br/controleacademico/negocio/ComandoAluno.java	7	7	24	38
/controleacademico/br/controleacademico/negocio/ComandoCRUDTurma.java	8	7	38	53
/controleacademico/br/controleacademico/negocio/ComandosTurmaOutros.java	13	5	66	84
/controleacademico/br/controleacademico/negocio/ComandoUsuario.java	5	7	20	32
/struts/br/controleacademico/struts/actions/AbstractAction.java	37	8	110	155
/struts/br/controleacademico/struts/actions/AlunoAction.java	12	7	43	62
/struts/br/controleacademico/struts/actions/AlunoAction-validation.xml	12	0	44	56
/struts/br/controleacademico/struts/actions/AutenticacaoAction.java	14	7	54	75
/struts/br/controleacademico/struts/actions/AutenticacaoAction-validation.xml	4	0	14	18
/struts/br/controleacademico/struts/actions/CRUDActions.java	37	7	148	192
/struts/br/controleacademico/struts/actions/CursoAction.java	4	7	10	21
/struts/br/controleacademico/struts/actions/CursoAction-validation.xml	3	0	9	12
/struts/br/controleacademico/struts/actions/DisciplinaAction.java	4	7	10	21
/struts/br/controleacademico/struts/actions/DisciplinaAction-validation.xml	6	0	33	39
/struts/br/controleacademico/struts/actions/MatriculasAction.java	23	7	126	156
/struts/br/controleacademico/struts/actions/PortalTurmaActions.java	47	7	214	268
/struts/br/controleacademico/struts/actions/ProfessorAction.java	4	7	10	21
/struts/br/controleacademico/struts/actions/ProfessorAction-validation.xml	5	0	19	24
/struts/br/controleacademico/struts/actions/TurmaAction.java	30	7	126	163
/struts/br/controleacademico/struts/actions/TurmaAction-validation.xml	6	0	33	39
/struts/br/controleacademico/struts/actions/UsuarioAction.java	20	7	85	112
/struts/br/controleacademico/struts/actions/UsuarioAction-validation.xml	5	0	28	33
/struts/br/controleacademico/struts/interceptors/AutenticadorInterceptor.java	8	7	25	40
/struts/br/controleacademico/struts/interceptors/DAOCloserInterceptor.java	4	7	18	29
/struts/br/controleacademico/struts/interceptors/DAOOpennerInterceptor.java	5	7	18	30

C-2 LOC da aplicação baseada em JSF

Arquivo	Bra	Com	Cod	Total
/arq/br/arq/dao/Conexao.java	10	7	34	51
/arq/br/arq/dao/DAO.java	20	7	29	56
/arq/br/arq/dao/DAOFactory.java	7	0	30	37
/arq/br/arq/dao/DAOImpl.java	37	7	190	234
/arq/br/arq/dao/DAOImplClass.java	4	5	10	19
/arq/br/arq/dao/hibernate.cfg.xml	3	0	36	39
/arq/br/arq/dominio/Ordenacao.java	12	13	38	63
/arq/br/arq/dominio/Paginacao.java	19	35	48	102
/arq/br/arq/dominio/Persistente.java	4	7	6	17
/arq/br/arq/excecoes/Excecao.java	6	20	15	41
/arq/br/arq/excecoes/ExcecaoDAO.java	5	16	13	34
/arq/br/arq/excecoes/ExcecaoService.java	4	13	10	27
/arq/br/arq/negocio/Comando.java	11	5	25	41
/arq/br/arq/negocio/ComandoCRUD.java	13	0	37	50
/arq/br/arq/negocio/ComandoFacade.java	7	0	20	27
/arq/br/arq/negocio/ComandoOperacao.java	12	5	43	60
/arq/br/arq/negocio/ComandoOperacoes.java	8	7	13	28
/arq/br/arq/negocio/OperacaoDescricao.java	4	7	10	21
/arq/br/arq/util/UtilHelper.java	7	7	40	54
/controleacademico/br/controleacademico/ComandoOperacoesCA.java	14	0	56	70
/controleacademico/br/controleacademico/dao/AlunoDAOImpl.java	4	7	9	20
/controleacademico/br/controleacademico/dao/interf/AlunoDAO.java	4	5	8	17
/controleacademico/br/controleacademico/dao/interf/MatriculaAlunoDAO.java	5	5	10	20
/controleacademico/br/controleacademico/dao/interf/TurmaDAO.java	9	5	20	34
/controleacademico/br/controleacademico/dao/interf/UsuarioDAO.java	6	5	12	23
/controleacademico/br/controleacademico/dao/MatriculaAlunoDAOImpl.java	5	7	17	29
/controleacademico/br/controleacademico/dao/TurmaDAOImpl.java	11	7	76	94
/controleacademico/br/controleacademico/dao/UsuarioDAOImpl.java	9	7	38	54
/controleacademico/br/controleacademico/dominio/Aluno.java	41	105	138	284
/controleacademico/br/controleacademico/dominio/Curso.java	14	43	62	119
/controleacademico/br/controleacademico/dominio/Disciplina.java	24	68	88	180
/controleacademico/br/controleacademico/dominio/MatriculaAluno.java	46	108	163	317
/controleacademico/br/controleacademico/dominio/Professor.java	21	60	80	161
/controleacademico/br/controleacademico/dominio/Turma.java	47	112	188	347
/controleacademico/br/controleacademico/dominio/TurmaMaterial.java	30	78	110	218
/controleacademico/br/controleacademico/dominio/Usuario.java	32	77	122	231
/controleacademico/br/controleacademico/negocio/ComandoAluno.java	7	7	24	38
/controleacademico/br/controleacademico/negocio/ComandoCRUDTurma.java	8	7	38	53
/controleacademico/br/controleacademico/negocio/ComandosTurmaOutros.java	13	5	66	84
/controleacademico/br/controleacademico/negocio/ComandoUsuario.java	5	7	20	32
/jsf/br/controleacademico/jsf/conversao/CPFConverter.java	9	19	43	71
/jsf/br/controleacademico/jsf/conversao/NotaConverter.java	9	27	45	81
/jsf/br/controleacademico/jsf/ItemLista.java	15	12	39	66
/jsf/br/controleacademico/jsf/ItemListaUtil.java	7	7	23	37
/jsf/br/controleacademico/jsf/JSPNavigation.java	5	7	12	24
/jsf/br/controleacademico/jsf/mbean/AlunoMBean.java	5	7	25	37
/jsf/br/controleacademico/jsf/mbean/CursoMBean.java	4	7	10	21
/jsf/br/controleacademico/jsf/mbean/DisciplinaMBean.java	4	7	10	21
/jsf/br/controleacademico/jsf/mbean/LoginMBean.java	7	7	26	40
/jsf/br/controleacademico/jsf/mbean/MatriculasAlunoMBean.java	28	28	123	179
/jsf/br/controleacademico/jsf/mbean/MBeanCRUD.java	31	14	132	177
/jsf/br/controleacademico/jsf/mbean/MBeanGenerico.java	45	14	162	221
/jsf/br/controleacademico/jsf/mbean/PortalTurmaMBean.java	41	49	173	263
/jsf/br/controleacademico/jsf/mbean/ProfessorMBean.java	4	7	10	21
/jsf/br/controleacademico/jsf/mbean/TurmaMBean.java	20	14	97	131
/jsf/br/controleacademico/jsf/mbean/UsuarioMBean.java	13	8	67	88
/jsf/br/controleacademico/jsf/phaselistener/Autenticador.java	5	22	28	55
/jsf/br/controleacademico/jsf/phaselistener/DAOCloser.java	8	7	25	40
/jsf/br/controleacademico/jsf/phaselistener/DAOOpenner.java	9	7	25	41
/jsf/br/controleacademico/jsf/uicomponent/ordenacao/OrdenacaoTag.java	13	21	40	74
/jsf/br/controleacademico/jsf/uicomponent/ordenacao/UIOrdenacao.java	14	14	66	94