Aplicação web para gerenciar vendas utilizando plataforma J2EE

Fabricio Meireles Monteiro

21 de janeiro de 2010

Às mulheres da minha vida,
Marise, Sabrina e Laura...
Ao homem que amo tanto,
Papai Gilberto...
Essa é pra vocês !
— FABRICIO MEIRELES MONTEIRO

Agradecimentos

Agradecer é trazer à mente todas as pessoas que partilharam o momento único de um desafio composto por instantes de fracasso e madrugadas de vitória. Agradecer é potencializar sua capacidade com os nutrientes do próximo. Agradecer é antes de tudo olhar o caminho da sua conquista e perceber que somente seus passos não serão capazes de percorrê-lo.

Agradecer nesta ocasião é perceber que em cada entrelinha por mim aqui escrita existe a entrega fiel e a coragem desbravadora de amigos. Agradecer aqui é ser admirado e se apropriar de tal grandeza para escalar os resultados pretendidos.

Agradecer é prostrar-se diante do Criador e colocar-se como coadjuvante Dele afim de que seus dons estejam sob minha posse.

Agradecer é saborear esse triunfo de engenheiro com você Sá que me ensina converter obstáculos em raras preciosidades.

Agradecer, enfim, é reconhecer que precisamos de pessoas para que os nossos projetos se realizem.

Agradecimento Especial

Agradeço em especial o meu orientador engenheiro Rodrigo Malara por conduzir-me nos caminhos de novas descobertas. Você lembra quando tive medo e fui reprovado na matéria “*Programação orientada a objetos*” ? Pois bem, foi ali que passei a admirar a sua pessoa quando com palavras sinceras você disse que era possível de fato dar a “volta por cima”.

Resumo

O comércio eletrônico (e-commerce) tornou-se popular com o surgimento da Internet. A Internet com os seus serviços básicos, como correio eletrônico e a WWW, têm criado um novo espaço para realização de negócios. Este novo ambiente tem fornecido para os agentes econômicos, tanto para as empresas quanto indivíduos, canais alternativos para realizar transações comerciais e, ainda, trocar informações e transferir diferentes tipos de produtos e serviços. Assim, as empresas estão encontrando no mundo virtual outra alternativa de oferecer seus serviços, tornando-se mais competitivas e com um custo reduzido. Este trabalho tem como objetivo arquitetar, construir e implementar um software que siga os padrões J2EE e que o mesmo esteja disponível vinte e quatro horas por dia nos sete dias da semana. Para tanto, é feito um estudo sobre os tipos de e-commerce, sobre a plataforma J2EE e frameworks para posteriormente especificar o software E-Commerce Filito aqui totalmente implementado. **Palavras–chaves**: e-commerce, J2EE, frameworks.

Sumário

1 Introdução **6**

1.1 Contextualização 6

1.2 Motivação 7

1.3 Objetivo 7

1.4 Organização do Trabalho 8

2 Revisão Bibliográfica **9**

2.1 Processos do Software 9

2.1.1 Desenvolvimento de Software e suas Abordagens 10

2.2 Rational Unified Process 12

2.2.1 Princípios e Práticas do RUP 12

2.2.2 Modelagem Visual da Aplicação 13

2.3 Unified Modeling Language 13

2.3.1 Introdução 13

2.3.2 Visão Geral 14

2.4 Java Enterprise Edition 15

2.4.1 Java 15

2.4.2 Plataforma J2EE 15

2.4.3 Clientes J2EE 16

2.4.4 Componentes Web 16

2.4.5 Componentes de Negócio 17

2.5 Frameworks 18

2.5.1 Framework Spring 18

2.5.2 Framework Hibernate 20

2.5.3 Web Framework 24
2.5.4 Web Framework Velocity 24
2.5.5 Web Framework Struts 24
2.5.6 Framework Validator Struts 25
2.6 Banco de Dados 33
2.6.1 Introdução 33
2.6.2 Evolução 34
2.6.3 Orientação a Objetos 34
2.6.4 SGBD 35
2.6.5 MySQL 35
2.7 Representação Estrutural da aplicação E-Commerce Filito 36
2.8 Softwares utilizados durante desenvolvimento 39

3 Especificação E-Commerce Filito **47**
3.1 Diagrama Casos de Uso 47
3.2 Cenário da aplicação desenvolvida 57

4 Resultados e Trabalhos Futuros **59**
4.1 Contribuições 59
4.2 Resultados Obtidos 60
4.3 Trabalhos Futuros 60

Lista de Figuras

2.1 *Container J2EE* 16
2.2 *Camada Web e aplicacoes J2EE* 17
2.3 *Camada de Negocio e Camada de Banco de Dados* 18
2.4 *Diagrama de classes que exemplifica a questao do grafo de navegacao* . 22
2.5 *Esquema: tarefas de entrada, processamento e saida da arquitetura MVC* 27
2.6 *O MVC e geralmente representado como tres componentes interligados* . 29
2.7 *As camadas da aplicacao web* 29
2.8 *Processo de solicitacao-resposta do Struts. UML de Jean-Michel Garnier* 32
2.9 *Uma classe no Java e na UML* 37
2.10 *Um objeto* 38
2.11 *Uma interface* 38
2.12 *Pacote: ambiente de desenvolvimento, codigo java e notacao UML* 39
2.13 *Tela de edicao do Eclipse* 40
2.14 *Maven: compilando codigos e gerando pacotes* 41
2.15 *Servidor Jetty em execucao* 41
2.16 *Logomarca Software Foundation Apache* 42
2.17 *Interface padrao da ferramenta MySQL Administrator* 43
2.18 *Interface MySQL Administrator para inserir instrucoes SQL* 43
2.19 *Interface padrao da ferramenta Power Architect* 44
2.20 *Interface engenharia reversa Power Architect* 45
2.21 *Construcao de diagrama Caso de Uso com Argo UML* 46

3.1 *Diagrama de Casos de Uso* 48
3.2 *Pagina referente a comprar produto* 49
3.3 *Pagina compra efetuada com sucesso* 50

3.4 *Status do pedido atualizado* 53
3.5 *Lista com pedidos gerados* 55
3.6 *Cenario utilizado no desenvolvimento* 58

Lista de Tabelas

2.1 *Os componentes Struts indexados pela camada* 30

Capítulo 1

Introdução

1.1 Contextualização

A Internet foi desenvolvida há mais de três décadas, financiada pelo Departamento da Defesa dos EUA. Originalmente projetada para conectar os principais sistemas de computadores de cerca de uma dezena de universidades e organizações de pesquisa, a Internet atualmente é acessível a centenas de milhões de computadores no mundo [4].

Na Internet se fundem tecnologias de computação e de comunicações. Ela torna o trabalho mais fácil. Torna informações acessíveis mundialmente de forma instantânea e conveniente. Possibilita que indivíduos e empresas locais de pequeno porte tenham uma exposição mundial. Ela está alterando a forma com o comércio é feito. As pessoas podem procurar os melhores preços de praticamente todos os produtos e serviços. Comunidades de interesse especial podem permanecer em contato entre si. Pesquisadores podem se informar instantaneamente sobre os últimos avanços em qualquer parte do globo. Antigamente, a maioria dos aplicativos de computador rodava em computadores que não se comunicavam entre si. Atualmente, podem-se escrever aplicativos que se comunicam com as centenas de milhões de computadores do mundo [4].

Alguns dos serviços disponíveis na Internet, além da Web, são o acesso remoto a outras máquinas (Telnet e SSH), transferência de arquivos (FTP), correio eletrônico (e-mail, normalmente através dos protocolos POP3 e SMTP), boletins eletrônicos (news ou grupos de notícias), bate-papo on-line (Chat), mensagens instantâneas (ICQ, MSN

Messenger, Blogs), etc. O que estas aplicações têm em comum é o uso da arquitetura do tipo “cliente-servidor”.

O paradigma cliente-servidor é bastante difundido e usado praticamente em todos os processos distribuídos em que a aplicação servidora aguarda mensagens, executa serviços e retorna resultados. A aplicação cliente, pelo contrário, é a que estabelece a ligação, envia mensagens para o servidor e aguarda mensagens de resposta.

1.2 Motivação

A busca por uma fatia do mercado estimula empresas a se adaptarem as novas tecnologias de comunicação e de informação, buscando assim alcançar novos objetivos de negócio.

Com esse propósito a Empresa Trad, presente nos estados de São Paulo e Minas Gerais, atuante no mercado há anos como revendedora do produto chamado **Filito**[1] tem por necessidade a implementação de um novo sistema de vendas.

O sistema de vendas utilizado atualmente pela empresa Trad não atende mais todas as necessidades desejadas pela equipe de vendedores, principalmente no quesito acessibilidade. Esse fato motivou a empresa a buscar um novo sistema que seja capaz de gerenciar as vendas via Web. Assim, somente uma aplicação que funcione via Internet é capaz de suprir os quesitos buscados pela empresa.

1.3 Objetivo

O objetivo desse trabalho é a construção de uma aplicação capaz de realizar todo o gerenciamento das vendas do produto Filito via Web. Para isso, utiliza-se a poderosa plataforma de programação J2EE, Servidores Apache e Jetty, Frameworks e o banco de dados MySQL.

Três fatores agregados à tecnologia Java foram preponderantes para que essa fosse utilizada nesse trabalho. São eles: o baixo custo, total portabilidade entre as plataformas

[1] Filito é uma rocha metamórfica fina. Após passar por um processo de produção, se torna um produto semelhante a “cal” utilizado nas construções civis. Nesse trabalho, no entanto, o Filito é apenas um produto distribuído e vendido no mercado

existentes no mercado hoje e as suas poderosas API's que dinamizam a construção de todo código.

O Apache é um servidor Web configurável, robusto e de alta performance desenvolvido por uma equipe de voluntários, conhecida como Apache Group. Ele é responsável por disponibilizar páginas, fotos ou qualquer outro tipo de objeto ao navegador do cliente.

1.4 Organização do Trabalho

Os capítulos restantes deste trabalho são estruturados como segue. No capítulo 2 são apresentadas as revisões bibliográficas referentes às ferramentas utilizadas para construir toda a aplicação Web. No capítulo 3 é apresentada a metodologia utilizada na modelagem e desenvolvimento. O capítulo 4 apresenta algumas considerações finais em relação ao trabalho desenvolvido.

Capítulo 2

Revisão Bibliográfica

O comércio eletrônico busca automatizar tarefas comerciais, incluindo geração, processamento, coordenação, distribuição e harmonização de transações comerciais, via Internet e outras mídias eletrônicas. Freqüentemente a terminologia associada ao comércio eletrônico pode ser confusa. Quer a denominação seja *Internet commerce*, *e-commerce*, ou mesmo *ec*, trata-se da mesma tecnologia: comércio eletrônico, e todos os outros termos associados, significam comprar ou vender bens ou serviços de uma empresa eletronicamente.

Esta seção permeia um estudo sobre conceitos fundamentais adquiridos antes de arquitetar ou desenvolver o software de comércio eletrônico *E-Commerce Filito*.

2.1 Processos do Software

Um processo de desenvolvimento do Software fornece uma orientação sobre como desenvolvê-lo com sucesso. Tal orientação pode cobrir o espectro inteiro de atividades associadas ao desenvolvimento do software. O processo pode se manifestar na forma de abordagens aprovadas, melhores práticas, regras, técnicas, seqüência, etc [2].

Se for formal ou informal, o processo de desenvolvimento do software basicamente empregado tem um impacto profundo no sucesso de um projeto de software. Uma abordagem organizada poderia funcionar bem em um pequeno projeto, mas poderia levar ao caos um projeto grande e assim causar um impacto no calendário em geral. Do mesmo modo, um processo de desenvolvimento do software burocrático poderá levar a

frustração e sucumbir até a melhor equipe [2].

2.1.1 Desenvolvimento de Software e suas Abordagens

Há vários processos para desenvolver o software. Alguns dos mais predominantes / populares estão brevemente descritos abaixo:

⇨ **Abordagem do tipo simplesmente desenvolva:** é caracterizada por uma falta geral de formalidade e por um processo praticamente inexistente em torno das atividades de desenvolvimento do software. O desenvolvedor do software tem o papel principal, que talvez seja diferenciado pela experiência e especialidade na área. O único foco da equipe de desenvolvimento é completar o projeto do software da melhor maneira possível, usando qualquer meio conseguido pelas tecnologias à disposição [2].

⇨ **Processo em cascata:** abordagem muito usada no passado e continua a ser popular. A idéia é segmentar o desenvolvimento em fases seqüenciais (por exemplo: exigências, análise, construção, implementação, teste). Isso funciona bem para os projetos pequenos e para os projetos em que as exigências são estáveis e relativamente fixas, no qual o domínio do problema é bem compreendido e a solução foi aprovada em projetos similares no passado [2].

⇨ **Processo iterativo:** é baseada no modelo espiral de Boehm. A idéia é reduzir o risco no início do projeto executando a seqüência identificada nas atividades (exigências, análises, construção, etc.) diversas vezes e revelando cada atividade principal de uma maneira planejada. Cada iteração termina com uma versão executável [2].

⇨ **Rational Unified Process:** é uma evolução do processo *Objectory*[1] [6], que foi adquirido pela Rational Software há alguns anos e mesclado com a *Rational Approach*. Foi melhorado com o tempo por meio da incorporação de outros aspectos do desenvolvimento do software, assim como as melhores práticas identificadas pela indústria do software durante os anos [2].

Esse processo é a referência largamente usada para a construção dessa aplicação e o motivo para tal está fundamentado no seguinte: 1. O RUP[2] é um processo aprovado e atualmente está sendo usado com sucesso em muitos projetos. 2. Acredita-se que a

[1] Objectory: método cujo objetivo é tornar a análise compatível com a codificação do software
[2] RUP: **Rational Unified Process** é uma evolução do processo *Objectory*

arquitetura, análise e construção são essenciais para o sucesso em longo prazo de um projeto. 3. O RUP pode ser personalizado para se adequar a necessidades específicas.

⇨ **Processo ICONIX:** esse processo oferece uma abordagem parecida com a da RUP. Ele enfatiza a "análise robusta" e formaliza essa análise em um diagrama robusto. A análise robusta gira em torno da análise dos casos de uso e do estabelecimento de um primeiro corte nos objetos que participam em cada caso de uso. Esses objetos são classificados como objetos de controle, de limite e de entidade. De modo prático, a diferença é uma questão de semântica.

⇨ **Processo Open:** O processo *Object-oriented Process, Environment, and Notation* (OPEN) foi desenvolvido pelo consórcio OPEN. Como o RUP, desenvolveu-se a partir de uma mescla dos esforços anteriores na área. É basicamente para ser usado em um ambiente de desenvolvimento de software baseado em objetos ou componentes [2].

O OPEN é definido como uma estrutura do processo conhecida como *OPEN Process Framework* (OPF). O OPF fornece um conjunto de componentes que são divididos em cinco grupos: *Work Units* (Unidades de Trabalho), *Work Products* (Produtos de Trabalho), *Producers* (Produtores), *Stages* (Estágios) e *Languages* (Linguagens).

Producers são geralmente pessoas. Eles trabalham nas *Work Units* e produzem os *Work Products*. As *Languages*, da *Unified Modeling Language* (UML) para a *Structured Query Language* (SQL), são usadas para criar os *Work Products*. Tudo isso ocorre no contexto dos *Stages*, como as fases, marcos, etc., que fornece a organização para as *Work Units*.

⇨ **Extreme Programming / Feature-Driven Development:** originalmente proposta por Kent Beck, tem merecido muita atenção ultimamente. A XP é, em geral, posicionada como um "processo de desenvolvimento de software leve" e, na verdade, pode ser quase construída com um antiprocesso no sentido tradicional [2].

A principal idéia sob a XP é manter as coisas tão simples quanto o possível para ter o serviço pronto. As atividades XP são organizadas em torno de quatro tarefas maiores: planejamento, construção, codificação e teste.

2.2 Rational Unified Process

O RUP, abreviação de Rational Unified Process (ou Processo Unificado da Rational), é um processo proprietário de Engenharia de Software criado pela Rational Software Corporation, adquirida pela IBM [8].

Esse processo usa a abordagem da orientação a objetos em sua concepção e é projetado e documentado utilizando a notação UML (Unified Modeling Language) para ilustrar os processos em ação. É considerado pesado e preferencialmente aplicável a grandes equipes de desenvolvimento e a grandes projetos, porém o fato de ser amplamente customizável torna possível que seja adaptado para projetos de qualquer escala [8].

2.2.1 Princípios e Práticas do RUP

- *Desenvolver o software de modo iterativo*: o desenvolvimento iterativo segue um processo mais contínuo e cíclico, permitindo um percorrer mais fácil de correções no caminho. Assim, as questões de alto risco podem ser concentradas e o risco pode ser eliminado mais cedo. Os problemas são identificados continuamente e podem ser superados com um custo menor em vez de serem descobertos bem no final do esforço quando podem ameaçar o projeto inteiro [2].
- *Gerenciar as exigências*: um projeto dificilmente começa com todas as exigências já capturadas e descritas. Ao contrário, o processo é uma identificação gradual, compreensão e aprimoramento. Assim, as exigências precisam ser gerenciadas com cuidado para assegurar o sucesso do projeto [2].
- *Usar arquitetura baseada em componentes*: o software baseado em componentes oferece a vantagem de um desenvolvimento realmente modular. Tal desenvolvimento modular leva a uma melhor arquitetura em geral [2].
- *Modelar visualmente o software*: nas palavras de Grady Booch: “Um modelo é uma simplificação da realidade que descreve completamente um sistema de uma perspectiva em particular” [14]. A construção dos modelos leva a uma melhor compreensão do problema e melhora a comunicação sobre ele, tornando assim os sistemas complexos mais gerenciáveis. A modelagem visual é a maneira preferida de fazer a modelagem porque permite trabalhar em um nível mais alto de abstração [2].

➪ *Verificar continuamente a qualidade do software*: estudos provaram que corrigir os problemas depois de o produto ser distribuído é, sempre, várias vezes mais caro do que corrigir. O teste contínuo significa um teste inicial, que pode ter um custo mais baixo. Tal teste contínuo pode também oferecer uma avaliação mais objetiva do status verdadeiro do projeto [2].

➪ *Controlar as alterações do software*: durante a confecção do software muitas alterações são feitas e isso precisa ser gerenciado para que não ocorra um caos entre as versões. Assim, há uma forte necessidade de controlar as alterações para o andamento efetivo do projeto [2].

2.2.2 Modelagem Visual da Aplicação

Abstrair a programação do código e representá-lo usando blocos de construção gráfica constitui-se uma forma efetiva de obter a visão geral da aplicação. Usando esta representação, recursos técnicos podem determinar a melhor forma para implementar um dado conjunto de interdependências lógicas. Isto também constrói uma camada intermediária entre o processo de negócio e o código necessário através da tecnologia da informação. Um modelo neste contexto é uma visualização e ao mesmo tempo uma simplificação de um projeto complexo.

A Linguagem de Modelagem Unificada (UML) é usada nessa aplicação para modelar os Casos de Uso, Diagramas de classes e outros objetos. RUP também discute outras formas para construir estes modelos.

2.3 Unified Modeling Language

2.3.1 Introdução

A Unified Modeling Language (UML) é uma linguagem gráfica para a modelagem e o desenvolvimento de sistemas de software. Fornece um suporte de modelagem e visualização para todas as fases de desenvolvimento do software, desde a análise das exigências até a especificação, construção e distribuição.

A UML tem suas raízes em várias notações baseadas em objetos anteriores[3]. As

[3]A distinção entre notação e metodologia é uma fonte comum de discussão. A UML é uma notação

mais destacadas entre elas estão as notações popularizadas por Booch, Rumbaugh e et al, Jacobson e et al. Portanto, mesmo que a UML tenha sido formalizada há apenas alguns anos, suas antecessoras foram usadas para construir e especificar os sistemas desde o início dos anos 90.

Ainda sob direção da OMG, a UML continua sendo desenvolvida e aprimorada. Por exemplo, as extensões propostas no início da década fornecem notações comuns para a modelagem dos dados, a modelagem da aplicação Web e as construções J2EE de mapeamento para a UML.

2.3.2 Visão Geral

A idéia central ao usar UML é capturar os detalhes significantes sobre o sistema de modo que o problema seja claramente compreendido, a arquitetura da solução seja desenvolvida e a implementação escolhida seja claramente identificada e construída.

Uma notação rica para modelar visualmente a aplicação facilita este exercício. A UML não só fornece a notação para os blocos de construção básicos, como também fornece maneiras para expressar as relações complexas entre os blocos de construção básicos.

As relações podem ser estáticas ou dinâmicas por natureza. As relações estáticas giram basicamente em torno dos aspectos estruturais da aplicação. A relação de herança entre um par de classes, as interfaces implementadas por uma classe e a dependência em outra classe são exemplos de relações estáticas.

As relações dinâmicas, por outro lado, estão preocupadas com o comportamento da aplicação e assim existem durante a execução. As mensagens trocadas em um grupo de classes para cumprir alguma responsabilidade e o fluxo do controle da aplicação, por exemplo, são capturados no contexto das relações dinâmicas que existem na aplicação.

---

que pode ser aplicada usando muitas abordagens diferentes. Essas abordagens são as metodologias.

2.4 Java Enterprise Edition

2.4.1 Java

Em maio de 1995, a Sun anunciou Java formalmente em uma conferência importante. Java gerou interesse imediato na comunidade comercial por causa do fenomenal interesse pela World Wide Web. Java é agora utilizada para criar páginas da Web com conteúdo interativo e dinâmico, para desenvolver aplicativos de grande porte, para aprimorar a funcionalidade de servidores da World Wide Web (os computadores que fornecem o conteúdo que o usuário visualiza no navegador), fornecer aplicativos para dispositivos destinados ao consumidor final (como telefones celulares, pagers e assistentes pessoais digitais) e para muitas outras finalidades.

2.4.2 Plataforma J2EE

A plataforma J2EE[4] define um padrão para o desenvolvimento de aplicações multicamadas, simplificando o seu desenvolvimento através de uma padronização baseada em componentes, fornecendo um conjunto completo de serviços para esses componentes, e da manipulação automática de vários detalhes inerentes ao ambiente corporativo, tais como: controle transacional, ajuste de performance, persistência, segurança, escalabilidade, gerencia de recursos (processos, memória, conexões com banco de dados).

Uma aplicação J2EE é geralmente dividida em quatro camadas conforme mostra Figura 2.1: camada do cliente, camada da Web, camada de negócios e a camada de EIS (*Enterprise Information System*). Essas camadas são geralmente distribuídas em três locais distintos: as máquinas clientes, a máquina do servidor J2EE e as máquinas de banco de dados.

A lógica de aplicação é dividida em componentes de acordo com sua função. Um componente é uma unidade de software funcional independente composta de arquivos e classes relacionadas que se comunica com outros componentes.

A especificação J2EE define os seguintes componentes: clientes, componentes web e componentes de negócios, os quais são descritos a seguir.

[4] A versão J2EE utilizada é a 1.6.0

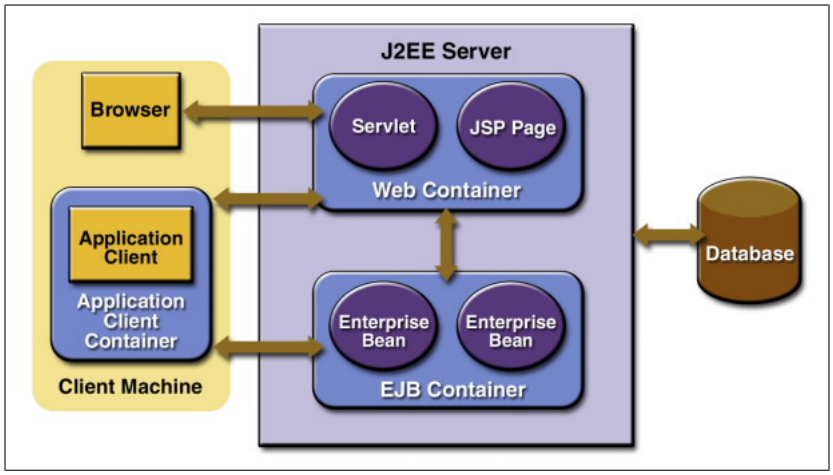

Figura 2.1: *Container J2EE*
[12]

2.4.3 Clientes J2EE

Um cliente J2EE pode ser um cliente web ou um cliente de aplicação. O cliente web é dividido em duas partes: um navegador web e as páginas web dinâmicas por ele recebidas do servidor. Essas páginas podem possuir um applet[5] incorporado, em que o cliente necessitará de um Java plugin e um arquivo de políticas de segurança para que este applet possa ser executado com sucesso pelo navegador web.

2.4.4 Componentes Web

Os componentes web podem ser tanto servlets ou páginas JSP [16].

Servlets são classes da linguagem Java armazenadas no servidor que executam dinamicamente solicitações e constróem respostas [16].

Páginas JSP são arquivos de texto que são executados como servlets mas permitem uma abordagem mais relacionada a criação de conteúdo estático [16].

Páginas estáticas HTML e applets são empacotados junto com os componentes web durante a construção da aplicação, mas não são considerados componentes web pela especificação J2EE.

[5]Applet: é um software aplicativo que é executado no navegador web por um plugin Java Runtime Environment

Como camada cliente e conforme observa-se na Figura 2.2, é possível incluir componentes JavaBeans para administrar os dados inseridos pelo usuário e enviar os mesmos para os beans de negócio que estão sendo executados na camada de negócio para serem processados [16].

2.4.5 Componentes de Negócio

Todo o código que contém a regra de negócio é por padrão empacotado e executado na camada de negócio. A Figura 2.3 mostra como um bean de negócio recebe e processa (se necessário) os dados de programas clientes e depois envia os mesmos para uma a camada responsável por armazenar estas informações (camada *Enterprise Information System* ou Database). Um bean de negócio também pode receber dado de uma base de dados, processá-lo (se necessário) e enviá-lo para um programa cliente.

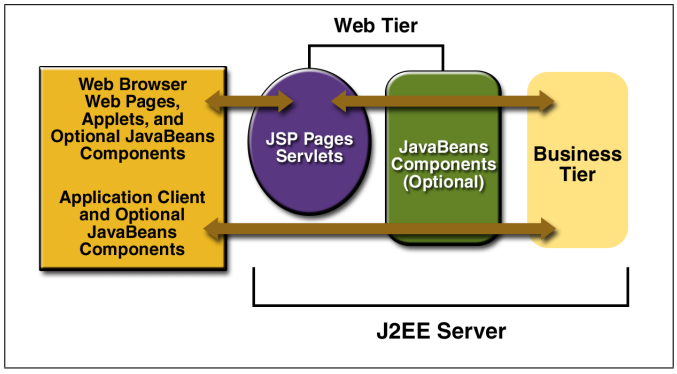

Figura 2.2: *Camada Web e aplicacoes J2EE*
[11]

Existem três tipos de beans de negócio: beans de sessão, beans de entidade de banco de dados e beans de mensagens. Um *bean de sessão* representa um uma comunicação de curta duração com um cliente. Quando um cliente finaliza e execução, o bean de sessão e todas as informações referentes a esse fluxo são excluídos. Ao contrário disso, um *bean de entidade* representa uma linha armazenada em uma tabela de banco de dados. Se o cliente finalizar a execução ou se o servidor for desligado os serviços garantem que o bean de entidade estão salvos [11].

Um *bean de mensagem* combina características de um bean de sessão com serviço de mensagem java (*Java Message Services*), permitindo um componente de negócio

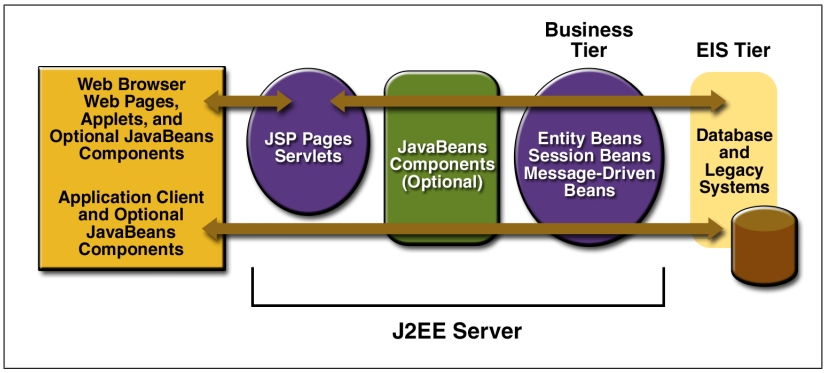

Figura 2.3: *Camada de Negocio e Camada de Banco de Dados* [11]

receber uma mensagem JMS de forma assíncrona [11].

2.5 Frameworks

Uma framework é uma aplicação reutilizável e semicompleta que pode ser especializada para produzir aplicações personalizadas [16] e, segundo Coad [3], um framework é um esqueleto de classes, objetos e relacionamentos agrupados para construir aplicações específicas.

Um framework fornece aos desenvolvedores um conjunto de componentes estruturais que têm as seguintes características:

- São conhecidos por funcionarem bem em outras aplicações.
- Estão prontos para serem usados com o próximo projeto.
- Pode também ser usados por outras equipes na organização.

Na aplicação a ser desenvolvida são utilizados os seguintes frameworks:

- Spring Framework.
- Hibernate Framework.
- Struts Validador Framework.

2.5.1 Framework Spring

É um framework open source criado por Rod Johnson e descrito em seu livro “Expert One-on-One: J2EE Design e Development”. Trata-se de um framework não intrusivo,

baseado nos padrões de projeto inversão de controle (IoC) e injeção de dependência [17].

No Spring[6] o container se encarrega de "instanciar" classes de uma aplicação Java e definir as dependências entre elas através de um arquivo de configuração em formato XML, inferências do framework, o que é chamado de auto-wiring ou ainda anotações nas classes, métodos e propriedades. Dessa forma o Spring permite o baixo acoplamento entre classes de uma aplicação orientada a objetos [17].

Esse framework oferece diversos módulos que podem ser utilizados de acordo com as necessidades do projeto, como módulos voltados para desenvolvimento Web, persistência, acesso remoto e programação orientada a aspectos [17].

Para obter um entendimento sobre as características do Spring, faz-se necessário compreender, inicialmente, o padrão **Inversão de Controle** (*Inversion of Control*), e sua variação denominada **Injeção de Dependência** (*Dependency Injection*).

Inversão de Controle

Inversão de controle (*Inversion of Control ou IoC*, em inglês) é o nome dado ao padrão de desenvolvimento de programas de computadores onde a sequência (controle) de chamadas dos métodos é invertida em relação à programação tradicional, ou seja, ela não é determinada diretamente pelo programador. Este controle é delegado a uma infraestrutura de software muitas vezes chamada de container ou a qualquer outro componente que possa tomar controle sobre a execução [5].

Por definição um fluxo normal de execução acontece quando um determinado programa cria chamadas para outros programas e assim sucessivamente, deixando a criação dos componentes, o início da execução e o fim da execução sob o controle do programador.

A inversão de controle ocorre quando ao invés de se criar explicitamente um código, ou acompanhar todo o ciclo de vida de uma execução, o programador delega alguma dessas funcionalidades para um terceiro.

São exemplos de inversão de controle:

- Programação Orientada a Eventos.
- Injeção de Dependência.

[6] A versão Spring utilizada é a 2.5.2

- Linguagens de Programação Funcional [18].

Injeção de Dependência

Injeção de dependência (*Dependency Injection*, em inglês) é um padrão de desenvolvimento de programas de computadores utilizado quando é necessário manter baixo o nível de acoplamento entre diferentes módulos de um sistema. Nesta solução as dependências entre os módulos não são definidas programaticamente, mas sim pela configuração de uma infraestrutura de software (container) que é responsável por "injetar" em cada componente suas dependências declaradas. A Injeção de dependência se relaciona com o padrão Inversão de controle mas não é um sinônimo deste [18].

2.5.2 Framework Hibernate

O framework Hibernate[7] provê o mapeamento de objetos java em tabelas de banco de dados. Nas seções seguintes são analisadas com mais detalhes as questões decorrentes do mapeamento de objetos em tabelas de banco de dados e como o framework Hibernate trata estas questões.

Mapeando Objetos em Tabelas

Quase todo tipo de aplicação requer que seus dados sejam persistidos de alguma forma. Um sistema de gerenciamento de banco de dados (SGBD) relacional não é específico para Java, tampouco destinado para uma aplicação específica. A tecnologia relacional provê uma forma de compartilhar dados entre aplicações diferentes e entre tecnologias diferentes usadas em uma mesma aplicação; por exemplo, um sistema gerador de relatórios e um sistema de gerenciamento de cadastro. Devido a estas características, os SGBDs são o meio de persistir dados das entidades de negócios mais utilizado em aplicações [9].

Quando a tecnologia envolvida é Java, a forma mais baixo nível que esta plataforma oferece para enviar comandos ao SGBD é através da API Java DataBase Connectivity(JDBC) [1]. Através do JDBC uma aplicação Java é capaz de enviar comandos SQL para um SGBD e assim realizar operações de consulta, atualização, inserção e remoção

[7] A versão Hibernate utilizada é a 3.3.1.GA

de dados. O código JDBC fica responsável por realizar a conversão dos dados provenientes do SGBD em classes orientadas a objeto escrito em Java.

Escrever código de acesso a banco de dados, realizando as conversões necessárias do paradigma relacional para o paradigma OO é uma tarefa onerosa, repetitiva e algumas vezes propensa a erros devido à diferença entre esses paradigmas. Para entender o framework Hibernate, é preciso antes entender o conflito dos paradigmas OO e relacional, que se propõe a mediar.

Conflito de Paradigmas: OO x Relacional

Ao migrar de um paradigma para o outro surgem questões que devem ser resolvidas. Dentre elas, as principais são: a questão dos subtipos, a questão da igualdade, as questões dos relacionamentos e a questão do grafo de navegações [9]. Esta seção não se propõe a apresentar as soluções destas questões, mas apenas apresentá-las. Posteriormente, é mostrado como o Hibernate trata destas questões.

A questão dos subtipos surge uma vez onde o paradigma OO possibilita que classes herdem de outras classes, podendo compor modelos complexos através de herança. Os sistemas de gerenciamento de banco de dados relacionais, exceto PostgreSQL, não oferecem suporte a heranças entre as tabelas.

A questão de igualdade surge quando é necessário verificar se um objeto é igual ao outro. Em Java, há duas maneiras de verificar a igualdade de objetos: através do conceito de identidade de objetos, que define que um objeto é igual ao outro se ambos ocupam o mesmo espaço de memória (é checado com o operador de igualdade “a == b”) e através do conceito de equivalência de objetos, que é determinada através do método equals() quando este é implementado (é checado através de “a.equals(b)”). No paradigma relacional a identidade de uma linha de uma tabela é definida através do valor da chave primária.

Relacionamentos em linguagens OO são expressos através de referências a objetos ou coleções de referências a objetos. Já no paradigma relacional os relacionamentos são expressos na forma de chaves estrangeiras. As questões dos relacionamentos surgem a partir das diferenças destas duas formas de representar relacionamentos.

Relacionamentos no paradigma OO são direcionais, ou seja, são definidos de uma classe para a outra. Para criar um relacionamento bidirecional no paradigma OO é

preciso definir dois relacionamentos unidirecionais, um em cada uma das classes envolvidas. A noção de direção em relacionamentos não existe no paradigma relacional. Além disso, classes no paradigma OO podem ser definidas tendo relacionamentos com multiplicidade um-para-um, um-para-muitos ou muitos-para-muitos enquanto que no paradigma relacional os relacionamentos entre tabelas são sempre um-para-muitos, definidos usando chaves estrangeiras, ou um-para-um, definidos usando chaves estrangeiras únicas. Relacionamentos muitos-para-muitos são feitos através de uma tabela extra, chamada tabela de ligação (link table) que não aparece no modelo OO.

Finalmente, há a questão do grafo de navegações. O diagrama UML da Figura 2.4 apresenta duas classes: a classe Message, representando uma mensagem que possui uma propriedade text que armazena o texto da mensagem e a classe User com duas propriedades, username que representa a identificação do usuário e name, que representa o nome do usuário. A classe Message tem um relacionamento unidirecional com multiplicidade um-para-um com a classe User.

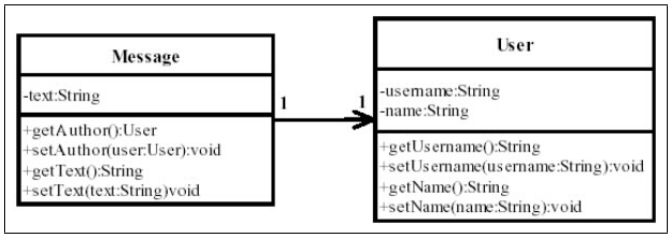

Figura 2.4: *Diagrama de classes que exemplifica a questao do grafo de navegacao*

Em um paradigma orientado a objetos, a forma natural de acessar dados em objetos relacionados é através da técnica onde se navega de um objeto para o outro usando o operador de navegação. Por exemplo, em Java para obter o nome do autor de uma mensagem realiza-se a seguinte operação sobre um objeto do tipo Message: message.getAuthor().getName(). Entretanto, esta não é a forma mais eficiente de recuperar dados em um SGBD relacional e pode levar ao problema conhecido com “n + 1 select problem” [9] onde para cada navegação no grafo de objetos é realizada uma consulta na base de dados. Recuperar o grafo inteiro de objetos de uma só vez implicará em problemas de performance e escalabilidade, que se tornam mais evidentes à medida que a massa de dados da base aumenta e os relacionamentos entre os objetos tornam-se mais complexos. A questão do grafo de navegação envolve então equilibrar esses

fatores, determinando qual porção do grafo de navegação deve ser recuperada imediatamente e qual deve ser recuperada sob demanda.

Muitas questões devem ser consideradas quando é preciso converter um objeto em tabelas. Esta tarefa é repetitiva, propensa a erros e independente do domínio da aplicação; portanto é susceptível à aplicação de frameworks de infra-estrutura.

Frameworks ORM

Mapeamento objeto / relacional (object / relational mapping - ORM) é o nome dado para a persistência automática e transparente de classes de uma aplicação em tabelas de banco de dados relacionais, transformando a representação de dados de um paradigma para outro [9]. Um framework ORM é um framework de infra-estrutura que fornece serviços de ORM.

Mark Fussel, um pequisador no campo da ORM, define quatro níveis de maturidade para soluções ORM: relacional puro (pure relational), mapeamento leve (light object mapping), mapeamento médio (medium object mapping) e mapeamento completo (full object mapping) [2].

Na aplicação a ser desenvolvida é utilizada a solução de mapeamento completo que oferece suporte a modelos complexos de objetos, envolvendo composição e herança. A persistência é transparente, ou seja, as classes persistidas não herdam de uma classe ou implementam uma interface do framework. Além disso, o framework implementa estratégias de otimização como, por exemplo, o cache de entidades.

Uma solução ORM com mapeamento completo como o framework Hibernate consiste de três módulos principais: uma API para realização de operações de consultas, atualizações, inserções e remoção de objetos de classes persistentes; uma linguagem ou API para especificação de consultas referentes a classes e as propriedades de classes; um recurso para especificação de metadados de mapeamento de objetos.

Hibernate é um framework ORM utilizado para persistir os objetos de negócio. Hibernate fornece mapeamento completo [2] e trata de todas as questões relativas ao mapeamento de objetos em tabelas levantadas na seção Conflito de Paradigmas: OO x Relacional. Através da API do Hibernate são realizadas operações de inserção, atualização, remoção e consulta de objetos, e através da linguagem de consulta do Hibernate (Hibernate Query Language - HQL) consultas mais complexas são realizadas. Os metadados

de mapeamento são definidos em arquivos XML.

2.5.3 Web Framework

Um web framework reusa análise, projeto e código. Reusa análise porque descreve os tipos de objetos importantes e como um problema maior pode ser dividido em problemas menores. Reusa projeto porque contém algoritmos abstratos que descrevem a interface que o programador deve implementar e as restrições a serem satisfeitas pela implementação. E reusa código porque torna mais fácil desenvolver uma biblioteca de componentes compatíveis e porque a implementação de novos componentes pode herdar grande parte de seu código das superclasses abstratas [7].

Segundo Johnson, uma característica importante dos web frameworks é que os métodos definidos pelo usuário são chamados de dentro do próprio web framework, ao invés de serem chamados do código de aplicação do usuário [7]. O web framework geralmente faz o papel de programa principal, coordenando e seqüenciado as atividades da aplicação web.

Nas seções/subseções seguintes, são apresentados os web frameworks utilizados na aplicação, bem como o padrão de projeto MVC2 utilizado por estes.

2.5.4 Web Framework Velocity

É outro projeto do Jakarta. É um template similar ao Freemarker, capaz de criar conteúdo dinâmico para web. O Velocity[8] pode gerar SQL, PostScript e XML a partir de templates, e pode também ser usado como utilitário "*stand-alone*"para gerar código fonte e relatórios. Pode ainda ser usado como um componente integrado de outros sistemas. O Velocity também provê serviços templates para potencializar frameworks de aplicações web.

2.5.5 Web Framework Struts

É um framework open source do projeto Jakarta que auxilia a implementação de aplicações para web. É construído em Java e seu "coração" consiste em uma camada de

[8]A versão Velocity utilizada é a 1.5

controle flexível baseada nas tecnologias Java Servlets, Java Beans, ResourcesBundles e XML.

O Struts[9] favorece o desenvolvimento de aplicações seguindo o paradigma MVC2. Fornece um componente Controller e se integra a outras tecnologias para oferecer suporte aos componentes Model (como JDBC, EJB's, etc.) e View (como Velocity, JSP, XSLT, etc.)

2.5.6 Framework Validator Struts

O framework Struts Validator[10] é um acréscimo poderoso ao framework Struts. Permite que as validações sejam gerenciadas em um arquivo de configuração separado, onde poderão ser revisadas e modificadas sem mudar o código Java ou JavaServer. É importante uma vez que, como a localização, as validações são ligadas ao nível de negócio e não devem ser misturadas com o código da apresentação.

O framework vem com diversos validadores que satisfazem a maioria da necessidades de rotina. É possível também adicionar facilmente validadores personalizados para as exigências especiais. Se requerido, o método *validate* do Struts original pode ser usado junto com o Struts Validator para garantir que todas as necessidades de validações possam ser satisfeitas.

Como o framework Struts principal, o Struts Validator é construído para a localização completa. Pode ainda compartilhar o arquivo de recursos da mensagem padrão com a estrutura principal, fornecendo uma solução uniforme para os conversores.

Validar a entrada de dados é um serviço essencial que um framework da aplicação web tem que fornecer e o Struts Validator é uma solução flexível que pode se dimensionar para satisfazer as necessidades até das aplicações mais complexas [16].

Padrões de Projeto

Um padrão de projeto descreve uma solução comprovada para um problema recorrente de arquitetura, evidenciando sempre o contexto no qual o problema foi apresentado para que outros desenvolvedores tenham conhecimento e possam assim implementar suas soluções fundamentado em padrões. Existem muitas definições para

[9] A versão Struts utilizada é a 1.3.8

[10] A versão Struts Validator utilizada é a 1.3.8

padrão de projeto mas todas elas tem em comum à recorrência de um problema/solução para um contexto específico. Algumas características mais comuns dos padrões são [2]:

- Surgem através da experiência adquirida.
- Normalmente são escritos em formato estruturado.
- Evitam reinvenção de algo já consolidado.
- Existem em diferentes níveis de abstração.
- São frequentemente aprimorados.
- São artefatos reutilizáveis.
- Sempre oferecem melhores práticas para os projetos.
- Podem ser utilizados em conjunto para resolver problemas maiores.

Os padrões de projeto são importantes, no entanto, eles somente não garantem o sucesso de uma aplicação. Só é possível determinar se um padrão é aplicável após uma análise cuidadosa de sua descrição e somente depois de aplicado é possível determinar se o padrão tem ajudado ou não no desenvolvimento.

No final dos anos 70, quando as interfaces gráficas do usuário (GUIs) estavam sendo inventadas, arquitetos de software viam as aplicações como tendo três partes maiores: a parte que gerencia os dados, a parte que cria as telas e os relatórios e a parte que lida com as interações entre o usuário e outros subsistemas [Ooram]. No início dos anos 80, o ambiente de programação ObjectWorks/Smalltalk introduziu esse grupo de três como uma estrutura de desenvolvimento. Na linguagem Smalltalk 80, o sistema de dados é denominado *Model*(Modelo), o sistema de apresentação é chamado de *View*(Exibição) e o sistema de interação é o *Controller*(Controlador). Desde então o padrão de projeto MVC se tornou comum, especialmente em sistemas orientados a objetos [2].

A Evolução do MVC

Originalmente o Model/View/Controller era um framework para construir aplicações Smalltalk. O framework suportava três classes representando o estado da aplicação, a apresentação da tela e o fluxo de controle - identificados como Model, View e Controller. Veja Figura 2.5.

A solução Smalltalk MVC é usar um padrão de notificação Observer. Nesse padrão, cada View registra-se como um observador dos dados de Model. Model pode então notificar as Views sobre as alterações enviando uma mensagem para todos os seus

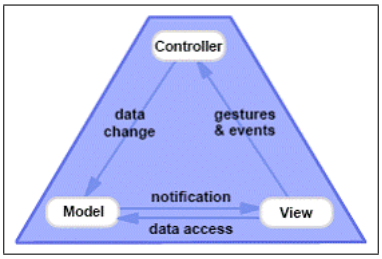

Figura 2.5: *Esquema: tarefas de entrada, processamento e saida da arquitetura MVC* [10]

observadores registrados.

Outras pessoas generalizam a estrutura Smalltalk MVC no paradigma arquitetura Model-View-Controller, que pode ser aplicado na construção de aplicações em qualquer plataforma.

O Início do Modelo 2

As JavaServer Pages são para tornar as páginas web dinâmicas mais fáceis de escrever. Os JSPs foram introduzidos pela primeira vez como uma alternativa para os servlets, assim como para as Active Server Pages da Microsoft. Aos desenvolvedores era oferecida a capacidade dos servlets como páginas do servidor fáceis de criar [16].

No entanto, muitas equipes acharam que, se não tivessem cuidado, um projeto poderia facilmente entrar em colapso sob o peso de páginas entrelaçadas irremediavelmente. Os recursos avançados requeriam o uso de scriptlets complexos. Mas os scriptlets são difíceis de reutilizar - a menos que você esteja transmitindo o código entre as páginas. As páginas utilitárias podem ser incluídas, mas são difíceis de manter organizadas e resultam em árvores de “origem”muito feias. Algo estava errado com essa imagem.

Muitos desenvolvedores logo perceberam que os JSPs e os servlets poderiam ser usados juntos para desenvolver aplicações web. Os servlets podiam lidar com o fluxo do controle; os JSPs podiam se concentrar na tarefa desagradável de escrever a HTML. No devido curso, usar os JSPs e servlets juntos ficou conhecido como o Modelo 2 (usar os JSPs sozinho era referido como modelo 1).

Naturalmente, muitos foram rápidos em mostrar que o Modelo 2 do JSP lembra a clássica arquitetura Model-View-Controller. Em alguns círculos, agora é comum usar os termos Modelo 2 e MVC de modo alternado, embora alguns discutam se uma aplicação

pode ser MVC e não suportar o padrão de notificação Observer clássico. O Model-View-Controller sem o padrão de notificação é algumas vezes chamado de MVC2 ou Web MVC [16].

Camadas da Aplicação - analisando a exibição

Uma razão para o Modelo 2 ser mantido distinto do MVC é que o padrão de notificação Observer não funciona bem em um ambiente web. O HTTP é um protocolo de "extração": o cliente solicita e o servidor responde. Sem solicitação, sem resposta. O padrão Observer requer um protocolo de anexação para a notificação, a fim de que o servidor possa transferir uma mensagem para um cliente quando o modelo mudar. Embora haja maneiras de simular a transferência dos dados para um cliente web, elas vão de encontro ao aspecto granular e poderiam ser consideradas como uma correção [16].

A Figura 2.6 mostra o paradigma Model-View-Controller como é mais demonstrado: um triângulo com três componentes interligados. Pode ser difícil para uma aplicação web manter a parte de "notificação de alteração"desse diagrama. Esse tipo de coisa funciona bem quando todos os recursos estão no mesmo servidor e os clientes têm uma conexão aberta com esse servidor. Não funciona tão bem quando os recursos são distribuídos em diversos servidores e os clientes não mantêm uma conexão aberta com a aplicação.

Muitos arquitetos de sistemas distribuídos, inclusive as aplicações, estremecem diante da idéia da exibição fazendo uma consulta do estado. Muito frequentemente, as aplicações remoteas são designadas em torno do padrão Layers[POSA]. Basicamente, o padrão Layers informa que as classes podem interagir com as classes em sua própria camada ou com as classes em uma camada adjacente. Em uma aplicação complexa, isso impede que as dependências cresçam exponencialmente quando os componentes são adicionados. A camada é um padrão básico na construção das aplicações remotas [16].

De um contexto MVC, introduzir o padrão Layer coloca a responsabilidade das alterações do estado e as consultas do estado no Controller junto com qualquer notificação da alteração.

Como ilustrado na Figura 2.7, as aplicações web em camadas usam uma construção "mais plana"que o MVC convencional. O Controlador é um sanduíche entre a camada

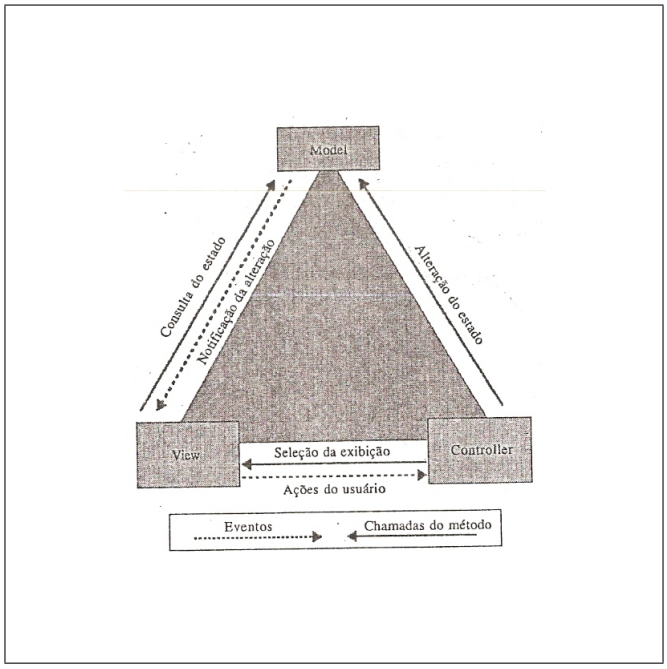

Figura 2.6: *O MVC e geralmente representado como tres componentes interligados* [16]

da apresentação (View) e a lógica da aplicação (Model).

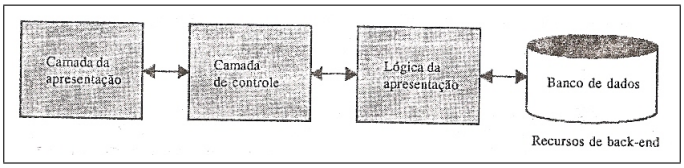

Figura 2.7: *As camadas da aplicacao web* [16]

As maiores responsabilidades de cada componente são inalteradas. O fluxo muda ligeiramente no sentido de que qualquer consulta do estado ou notificação da alteração tem de passar pelo Controller. Outra diferença é que, quando View ou a camada da apresentação mostra o conteúdo dinâmico, usa os dados transmitidos pelo Controller em vez dos dados retornados diretamente por Model. Essa alteração desliga View de Model, permitindo que o Controller selecione os dados e que View exiba os dados [16].

Implementação Struts

O Struts implementa a arquitetura Modelo 2 fornecendo um servlet controlador que pode ser usado para gerenciar o fluxo entre as páginas JSP e outros dispositivos da camada de apresentação. O Struts implementa o padrão MVC/Layers usando ActionForwards e ActionMappings para manter as decisões do fluxo do controle fora da camada de apresentação. Os JSPs podem se referir aos destinos lógicos. Os componentes Controller fornecem os URIs reais durante a execução [16].

A Tabela 2.1 indexa os componentes Struts pela camada.

Tabela 2.1: *Os componentes Struts indexados pela camada*

Camada View	**Camada Controller**	**Camada Model**
Extensões da tag JSP	ActionForwards Classes ActionForm ActionMappings ActionServlet Classes Action ActionErrors MessageResources	GenericDataSource
JavaServer Pages, templates Velocity e outros veículos de apresentação fornecidos pelo desenvolvedor	Várias classes utilitárias, como Commons-Digester e Commons-BeansUtils	Outros serviços de dados e APIs fornecidos pelo desenvolvedor

Os componentes ActionServlet, ActionForm, Action e ActionMapping são componentes fundamentais da arquitetura Struts, sendo descritos da seguinte forma:

O ***ActionServlet*** (org.apache.action.ActionServlet) é o controlador no framework Struts. Ele é responsável por criar e interagir com os objetos Action e utilizar o ActionMapping para carregar os mapeamentos.

O ***ActionForm*** (org.apache.action.ActionForm) é responsável por tratar os formulários. Suas instâncias são preenchidas com os dados do formulário e validados (se necessário) pela invocação do método validate antes de um objeto Action ser executado.

O ***Action*** (org. apache.action.Action) chama a lógica de negócio, que em geral, se encontra separada fisicamente em outra camada, com o objetivo de manter a capacidade de reutilização de quaisquer componentes de negócios.

O ***ActionMapping*** (org.apache.action.ActionMapping) contém todos os mapeamentos utilizados pelo controlador para encaminhar as solicitações para os respectivos ob-

jetos Action. Os objetos ActionMapping são carregados em memória a partir de um arquivo XML chamado struts-config.xml [16].

Note que segundo o padrão Layers (veja Seção 2.5.4), os componentes devem interagir apenas com os outros componentes em sua coluna ou coluna adjacente. Para saber, os componentes Model não devem interagir diretamente como os componentes View.

Na prática, Controller e View interagem através dos contextos da solicitação, sessão e aplicação fornecidos pela plataforma do servlet. Controller e Model interagem através do sistema de arquivos e de memória (no caso de carregar documentos XML ou arquivos Properties) ou através de outros serviços, como o TCP, para criar uma conexão com um banco de dados JDBC.

Fluxo de Controle do Struts

Como as aplicações web são dinâmicas, é difícil representar um fluxo de controle verdadeiro. Dependendo das circunstâncias, muitas coisas diferentes podem ocorrer de maneiras diversas - especialmente nas aplicações web. Mas há ainda uma ordem geral para as coisas [16].

Nos frameworks da aplicação ou mesmo nas aplicações web, “essa ordem geral das coisas” ou esse processo pode parecer difícil de seguir à primeira vista.

A Figura 2.8 mostra o processo de solicitação-resposta do Struts em uma sequência visual. Abaixo segue uma descrição da solicitação-resposta. Os números entre parênteses referem-se à Figura 2.8 onde for apropriado [16]:

- Um cliente solicita um caminho que coincide com o padrão Action URI (1).
- O contêiner transmite a solicitação para o ActionServlet.
- Se for uma aplicação modular, o ActionServlet selecionará o devido módulo.
- O ActionServlet pesquisa o mapeamento para obter o caminho.
- Se o mapeamento especificar um componente do formulário, o ActionServlet verá se já existe um, ou criará um (1.1.1).
- Se um componente do formulário estiver em cena, o ActionServlet irá redefini-lo e preenchê-lo a partir da solicitação HTTP (1.1.2).
- Se o mapeamento tiver a propriedade “validate” definida para “true”; chamará validate no componente do formulário (1.1.3).

➪ Se falhar, o servlet enviará para o caminho especificado pela propriedade “input” e esse fluxo do controle terminará.

➪ Se o mapeamento especificar um tipo Action, será reutilizado se já existir ou instanciado (1.1.4).

➪ O método “perform” ou “execute” do Action é chamado e transmitido para o componente do formulário instanciado (ou null).

➪ O Action pode preencher o componente do formulário, chamar os objetos de negócios e fazer o que mais for necessário (1.1.4.1-1.1.4.3).

➪ O Action retorna um ActionForward para o ActionServlet (1.1.4.4).

➪ Se o ActionForward for para outro Action URI, o fluxo começa de novo; do contrário, será enviado para uma página de exibição ou algum outro recurso. Muito frequentemente, será um JSP, no caso um Jasper ou um equivalente (não Struts) que apresentará a página (2,3) [16].

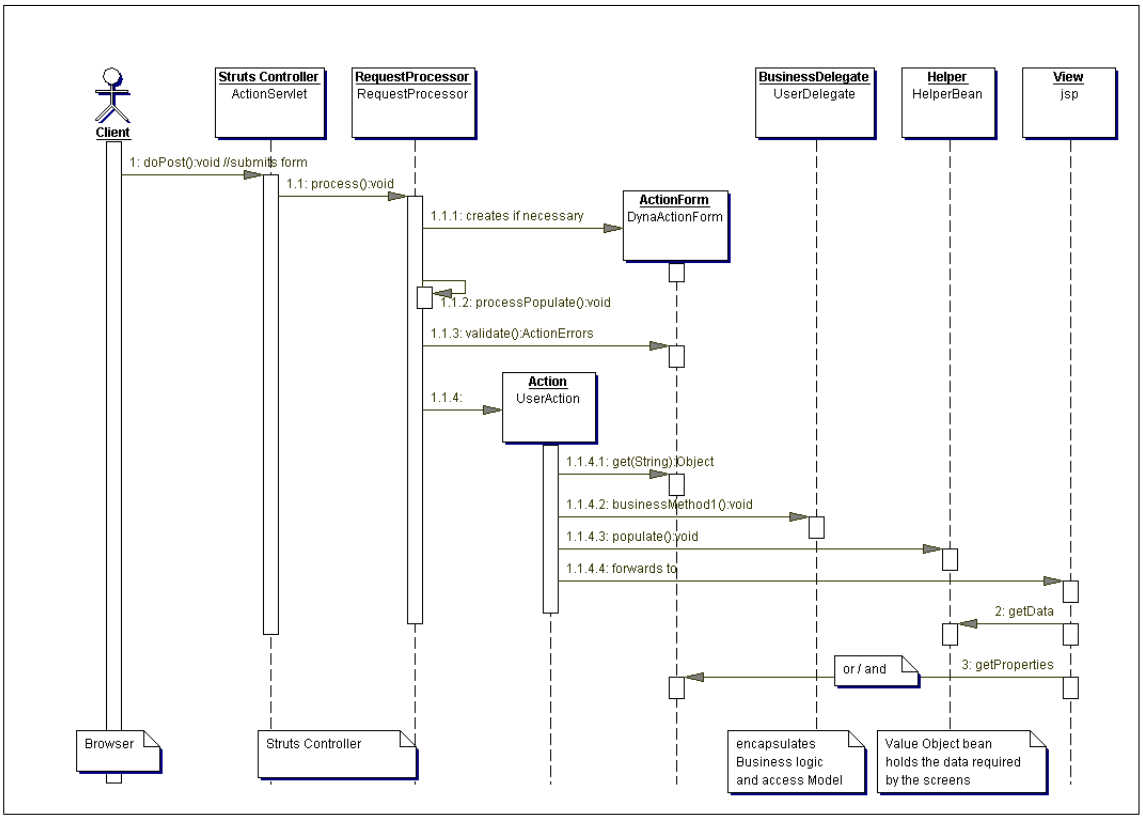

Figura 2.8: *Processo de solicitacao-resposta do Struts. UML de Jean-Michel Garnier*

➪ Se o JSP usar as tags Struts HTML e elas virem o ActionForm correto na solicitação (1.1), preencherão seus controles a partir do ActionForm. Do contrário, a tag - html:form - criará um. Desde o Struts 1.1, a tag do formulário também chamará a redefinição no ActionForm se criar o objeto em si [16].

2.6 Banco de Dados

2.6.1 Introdução

Igualmente a muitas tecnologias na computação industrial, os fundamentos de banco de dados relacionais surgiram na empresa IBM, nas décadas de 1960 e 1970, através de pesquisas de funções de automação de escritório. Foi durante um período da história na qual empresas descobriram que estava muito custoso empregar um número grande de pessoas para fazer trabalhos como armazenar e indexar (organizar) arquivos. Por este motivo, fez-se necessário esforços e investimentos para pesquisar um meio mais barato e ter uma solução mecânica eficiente [15].

Muitas pesquisas foram conduzidas durante este período, cujos modelos hierárquicos, de redes, relacionais e outros foram descobertos [15].

Em 1970 um pesquisador da IBM – Ted Codd – publicou o primeiro artigo sobre banco de dados relacionais. Este artigo tratava sobre o uso de cálculo e álgebra relacional para permitir que usuários não técnicos armazenassem e recuperassem grande quantidade de informações. Codd visionava um sistema onde o usuário seria capaz de acessar as informações através de comandos em inglês, onde as informações estariam armazenadas em tabelas [15].

Devido à natureza técnica deste artigo e a relativa complicação matemática, o significado e proposição do artigo não foram prontamente realizados. Entretanto, ele levou a IBM a montar um grupo de pesquisa conhecido como System R (Sistema R) [15].

O projeto do Sistema R era criar um sistema de banco de dados relacional o qual eventualmente se tornaria um produto. Os primeiros protótipos foram utilizados por muitas organizações, tais como MIT Sloan School of Management (uma renomada escola norte-americana). Novas versões foram testadas por empresas e tinham o propósito de rastrear o manufaturamento de estoque. Naturalmente, o Sistema R evolui para SQL/DS, o qual posteriormente tornou-se o DB2. A linguagem criada pelo grupo do Sis-

tema R foi a Structured Query Language (SQL) – Linguagem de consulta estruturada. Esta linguagem tornou-se um padrão na indústria para os bancos de dados relacionais e hoje em dia é um padrão ISO (International Organization for Standardiazation) [15].

2.6.2 Evolução

Durante a década de 80, usuários que utilizavam o sistema passaram a contribuir de forma indireta na evolução do sistema, ou seja, esses usuários davam o “feedback” (retorno) e o software de banco de dados relacionais foi sendo refinado [15].

Desde a sua chegada, os bancos de dados têm tido aumento nos dados de armazenamento, desde os oito megabytes até centenas de terabytes de dados em lista de e-mail, informações sobre consumidores, sobre produtos, vídeos, informações geográficas, etc. Com este aumento de volume de dados, os sistemas de banco de dados em operação também sofrem constantemente um aumento de tamanho [15].

Juntamente com esses aprimoramentos ao longo dos anos, o padrão SQL passou da IBM para a ANSI (American National Standards Institute) e para a ISO, os quais formaram um grupo de trabalho para continuar o desenvolvimento. Este desenvolvimento ainda acontece com outras novas versões dos padrões definidos [15].

2.6.3 Orientação a Objetos

Em meados da década de 80 tornou-se óbvio que existiam várias áreas onde banco de dados relacionais não eram aplicáveis, por causa do tipo de dados envolvidos. Estas áreas incluíam medicina, multimídia e física de energia elevada, todas com necessidade de flexibilidade em como os dados seriam representados e acessados [15].

Este fato levou ao início de pesquisas em banco de dados orientados a objetos, os quais os usuários poderiam definir seus próprios métodos de acesso aos dados e como estes seriam representados a acessados. Ao mesmo tempo, linguagens de programação orientadas a objetos (Object Oriented Programming - POO), tais como C++, começaram a surgir na indústria [15].

No início de 1990, tem-se o surgimento do primeiro Sistema de Gerenciamento de Banco de Dados Orientado a Objetos, através da companhia Objectivity. Isso permitiu aos usuários criarem sistemas de banco de dados para armazenar resultados de

pesquisas como o CERN (maior laboratório do ramo de partículas físicas em pesquisas nucleares) e SLAC (Centro de Aceleração Nuclear), para mapeamento de rede de provedores de telecomunicações e para armazenar registros médicos de pacientes em hospitais, consultórios e laboratórios, respectivamente [15].

2.6.4 SGBD

Um sistema de gerenciamento de banco de dados SGBD consiste em uma coleção de dados inter-relacionados e um conjunto de programas para acessá-los. Um conjunto de dados, normalmente referenciado como banco de dados, contém informações sobre uma empresa particular, nesse caso, de estoque do produto Filito, fornecedores do produto, consumidores do produto. O principal objetivo de um SGBD é prover um ambiente que seja adequado e eficiente para recuperar e armazenar informações em um banco de dados [15].

Os sistemas de banco de dados são projetados para gerenciar grandes grupos de informações. O gerenciamento de dados envolve a definição de estruturas para armazenamento de informação e o fornecimento de mecanismos para manipulá-los. Além disso, o sistema de banco de dados precisa fornecer segurança às informações armazenadas, caso o sistema dê problema, ou contra tentativas de acesso não autorizado. Se os dados devem ser divididos entre diversos usuários, o sistema precisa evitar possíveis resultados anômalos [15].

A importância das informações na maioria das organizações e o conseqüente valor dos bancos de dados têm orientado o desenvolvimento de grande parte dos conceitos e técnicas para o gerenciamento eficiente dos dados [15].

2.6.5 MySQL

O Gerenciador de Banco de Dados MySQL[11] é um sistema cliente / servidor que consiste de um servidor SQL multitarefa que suporta acessos diferentes, diversos programas clientes e bibliotecas, ferramentas administrativas e diversas interfaces de programação (API's) [13].

[11] A versão MySQL utilizada é a 5.1

Também se concede o Servidor MySQL como uma biblioteca multitarefa capaz de se conectar à aplicação com o intuito de obter um produto mais rápido, menor e mais facilmente gerenciável [13].

Sabe-se que um sistema de gerenciamento de banco de dados relacional armazena dados em tabelas separadas em vez de colocar todos os dados em um só local. Isso proporciona velocidade e flexibilidade [13].

A confiabilidade, juntamente com a sua capacidade de expansão, foram realidades comprovadas durante a leitura de bibliografias ao redor do mundo e que agregaram o interesse em utilizar o MySQL nessa aplicação.

2.7 Representação Estrutural da aplicação E-Commerce Filito

Os conceitos estruturais, como a classe e a interface, são fundamentais para o Java e a UML. Esta seção mostra como alguns conceitos do código Java implementado mapeiam-se no na UML [2].

- **Classe, Variáveis e Métodos**

  Na UML, uma **classe** Java é representada através de um retângulo com compartimentos. Três compartimentos horizontais são usados:

  ➪ *Compartimento do nome:* mostra o nome da Classe Java

  ➪ *Compartimento do atributo:* lista as variáveis definidas na classe, se houver

  ➪ *Compartimento das operações:* mostra os métodos definidos na classe, se houver

  As **variáveis** Java podem se manisfestar de várias maneiras na UML. A forma mais simples de declaração da variável é listá-la no compartimento de atributo de uma classe. Sublinhar o atributo indica a natureza *estática* da variável. A visibilidade de um atributo pode ser indicada antecedendo-o com o operador matemático de **adição** para público, **#** para protegido e o operador matemático **subtração** para privado [2].

Os **métodos** são o equivalente das operações em uma classe na UML. Eles são mostrados no terceiro compartimento para uma classe. O escopo da visibilidade das operações UML é definido usando a mesma convenção utilizada para os atributos da classe. Sublinhar o nome da operação é usado para diferenciar um método estático. Listar a operação em *itálico* no compartimento da operação mostra que o método é abstrato. Naturalmente é possível ocultar ou exibir os detalhes dependendo da importância do detalhe [2].

A Figura 2.9 mostra uma classe Java implementada na aplicação E-Commerce e sua equivalência na notação UML.

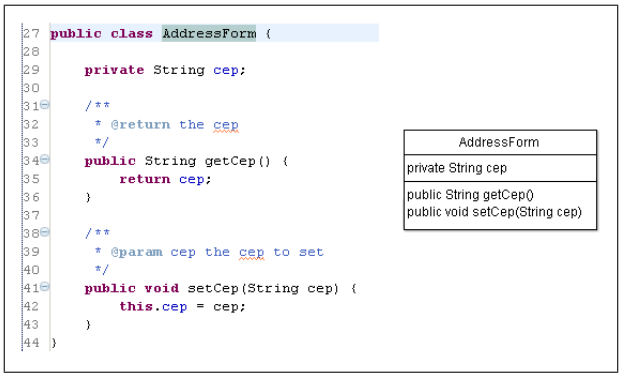

Figura 2.9: *Uma classe no Java e na UML*

- **Objeto**

  Embora o Java e a UML tenham o conceito de objeto, não há nenhum mapeamento direto entre um objeto UML e o código Java. É porque os objetos são entidades dinâmicas, baseadas nas definições da classe. As aplicações Java são escritas em termos de classes Java que resultam na criaçãode objetos Java quando a aplicação é de fato executada [2].

  Na UML, os objetos são usados para modelar os aspectos dinâmicos do sistema através de diagramas de interação que não será abordado neste trabalho. Um retângulo com um nome do objeto e/ou um nome da classe é usado como a notação para um objeto. Algumas vezes é desejável mostrar os valores do atributo para o objeto em cada situação. Isso pode ser feito usando um retângulo com duas partições mostrando os atributos da classe. A Figura 2.10 mostra um exemplo prático utilizado na aplicação E-Commerce.

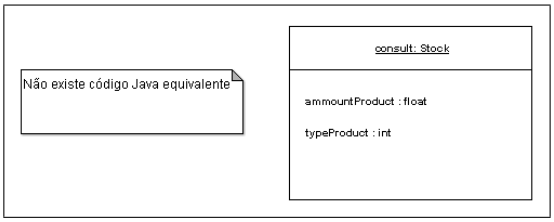

Figura 2.10: *Um objeto*

- **Interface**

  Na UML, uma interface Java é representada como uma classe com estereótipo com **interface**. As classes com estereótipo podem ter opcionalmente ícones associados. No caso de uma interface, a representação de ícones da UML é um pequeno círculo. Essa representação é comumente usada para representar as interfaces Java ao modelar na UML.

  A Figura 2.11 mostra a representação da interface padrão utilizada na aplicação E-Commerce.

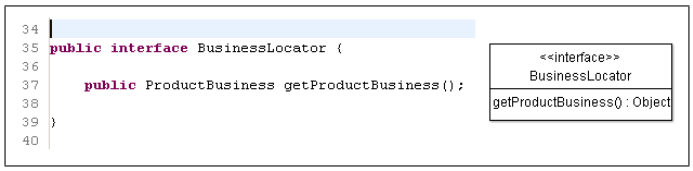

Figura 2.11: *Uma interface*

- **Pacote**

  Um pacote Java mapeia um pacote UML. Os pacotes podem ser lógicos, significando que é possível utilizá-los apenas como mecanismo de agrupamento. Os pacotes também podem ser físicos, significando que resultam em um diretório físico no sistema de arquivos [2].

  O pacote é representado como uma pasta, como mostrado na Figura 2.12 que ilustra a divisão utilizada na aplicação E-Commerce.

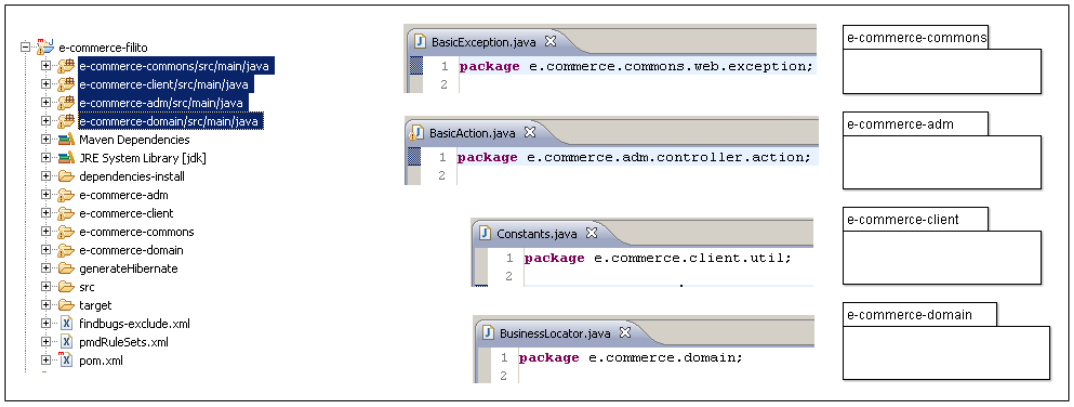

Figura 2.12: *Pacote: ambiente de desenvolvimento, codigo java e notacao UML*

2.8 Softwares utilizados durante desenvolvimento

Plataforma Eclipse

Eclipse[12] é uma plataforma (IDE) focado no desenvolvimento de ferramentas e aplicações de software. Possui como características marcantes o uso da SWT e não do Swing como biblioteca gráfica, a forte orientação ao desenvolvimento baseado em plug-ins e o amplo suporte ao desenvolvedor com centenas de plug-ins que procuram atender as diferentes necessidades de diferentes programadores. A Figura 2.13 ilustra a tela de edição de código do *Eclipse*.

Apache Maven

Apache Maven, ou simplesmente *Maven*[13], é uma ferramenta para gerenciamento e automação de projetos em Java. Possui um modelo de configuração baseado no formato XML. *Maven* é um projeto da Apache Software Foundation.

Maven utiliza uma construção conhecida como Project Object Model (POM). Ela descreve todo o processo de construção de um projeto de software, suas dependências em outros módulos e componentes e a sua sequência de construção. O *Maven* contém tarefas pré-definidas que realizam funções bem conhecidas como compilação e empacotamento de código.

[12] A versão Eclipse utilizada é a 3.5.1 - Galileo
[13] A versão Maven utilizada é a 2.0.10

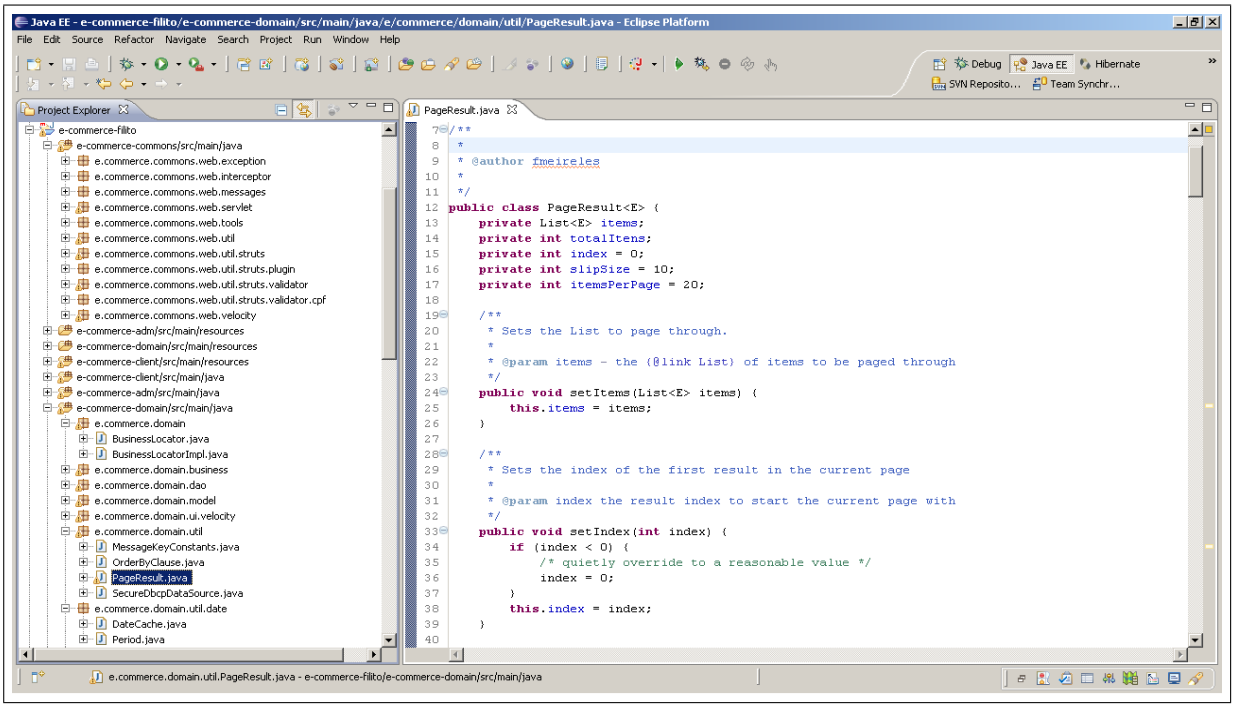

Figura 2.13: *Tela de edicao do Eclipse*

Uma característica chave do *Maven* é que ele é construído para trabalhar em rede. O núcleo da ferramenta pode baixar plugins de um repositório. (o mesmo repositório utilizado pelos outros projetos Java do Apache e outras organizações). O *Maven* disponibiliza suporte nativo para a recuperação de arquivos deste repositório, e para a inclusão dos artefatos resultantes no final do processo. Um cache de artefatos atua como ponto de sincronização dos artefatos de um projeto local. A Figura 2.14 mostra o *Maven* gerando os arquivos ".class"e pacotes da aplicação E-Commerce Filito.

Servidor Jetty

Jetty é um servidor HTTP Servlet de código aberto totalmente escrito em Java. Foi desenhado para ser leve, de alta performance, embebível, extensível e flexível, assim criando uma plataforma ideal para servir pedidos HTTP dinâmicos a partir de qualquer aplicação Java. A Figura 2.15 mostra a aplicação E-Commerce Filito sendo executada pelo *Jetty*[14].

[14]A versão Jetty utilizada é a 5.1.6

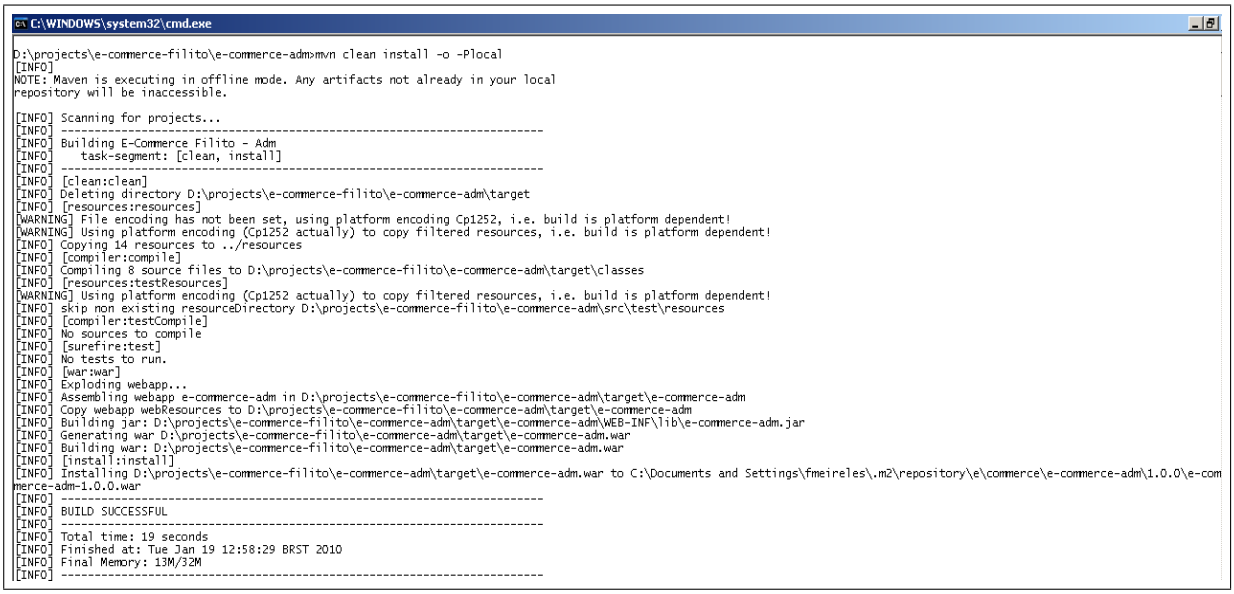

Figura 2.14: *Maven: compilando codigos e gerando pacotes*

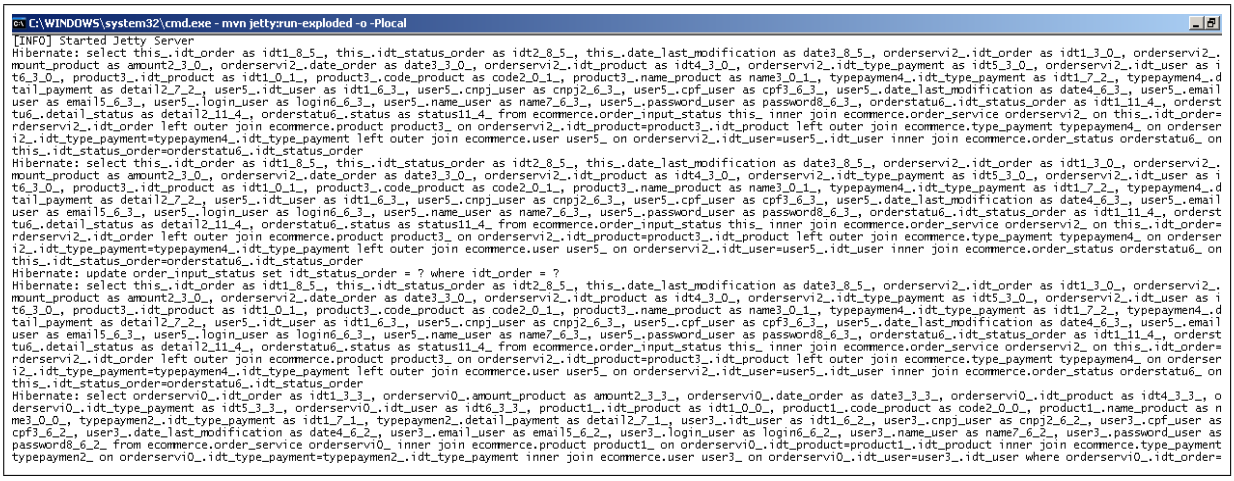

Figura 2.15: *Servidor Jetty em execucao*

Servidor Apache

O servidor *Apache* (ou Servidor HTTP Apache, em inglês: Apache HTTP Server, ou simplesmente: Apache) é o mais bem sucedido servidor web livre. Foi criado em 1995 por Rob McCool, então funcionário do NCSA (National Center for Supercomputing Applications).

É a principal tecnologia da Apache Software Foundation, responsável por mais de

uma dezena de projetos envolvendo tecnologias de transmissão via web, processamento de dados e execução de aplicativos distribuídos.

Figura 2.16: *Logomarca Software Foundation Apache*

O servidor é compatível com o protocolo HTTP versão 1.1. Suas funcionalidades são mantidas através de uma estrutura de módulos, permitindo inclusive que o usuário escreva seus próprios módulos — utilizando a API do software.

O servidor é configurado por um arquivo mestre nomeado httpd.conf e opcionalmente pode haver configurações para cada diretório utilizando arquivos com o nome .htaccess.

Na aplicação E-Commerce Filito o servidor Apache[15] é o reponsável pelo gerenciamento dos templates, arquivos javascript ".js" e arquivos de estilos ".css".

MySQL Administrator

MySQL Administrator[16] é um programa para executar operações administrativas, como configuração, monitoramento e início e parada de um servidor MySQL, gerenciamento de usuários e conexões, executar backups, e um grande número de outras tarefas administrativas. A Figura 2.17 mostra a interface padrão da ferramenta.

Já a Figura 2.18 mostra a interface de uma das funcionalidades que *MySQL Administrator* oferece, onde é possível executar instruções SQL.

Power Architect

Power Architect[17], foi outra ferramenta de banco de dados utilizado para desenvolver a aplicação E-Commerce Filito pois apresenta boas e úteis funcionalidades para o

[15] A versão Apache utilizada é a 2.2
[16] A versão MySQL Administrator utilizada é a 1.2.17
[17] A versão Power Architect utilizada é a 0.9.12

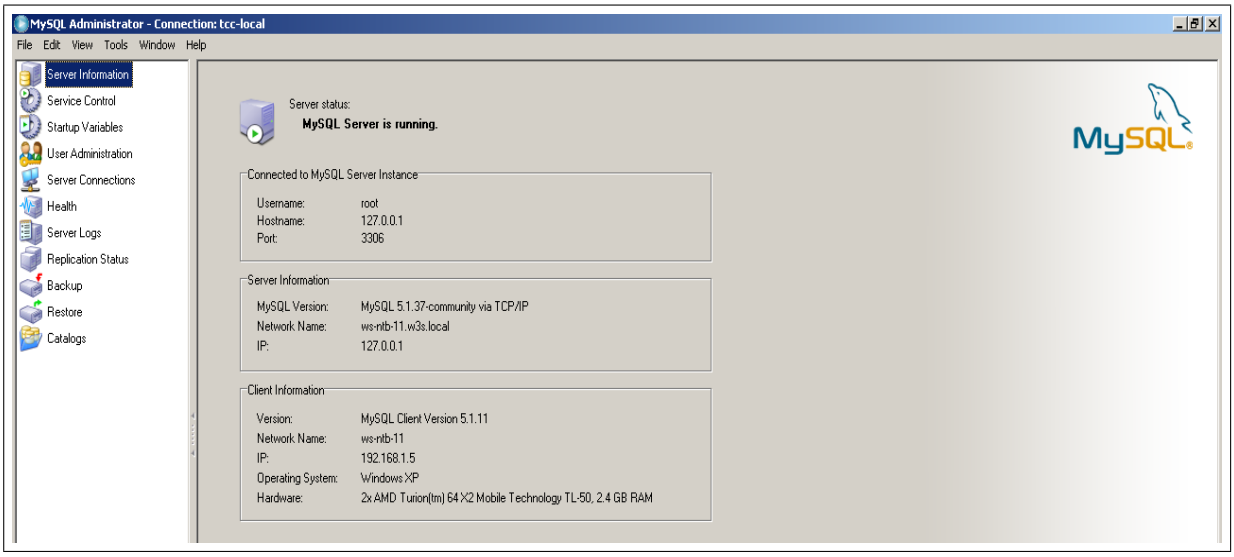

Figura 2.17: *Interface padrao da ferramenta MySQL Administrator*

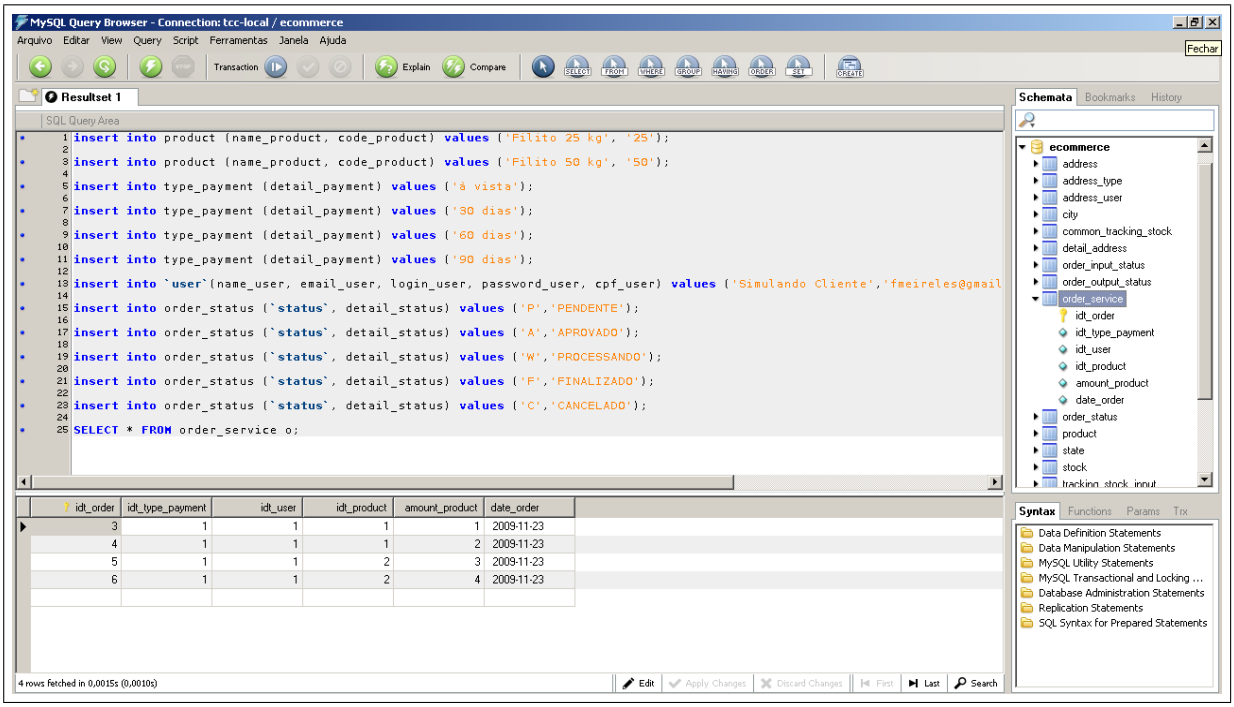

Figura 2.18: *Interface MySQL Administrator para inserir instrucoes SQL*

desenvolvimento de aplicações que tenham um certo grau de complexidade. É uma ferramenta livre e de código aberto. A Figura 2.19 exibe a interface padrão dessa ferra-

menta onde é possível por exemplo iniciar um novo projeto de banco de dados.

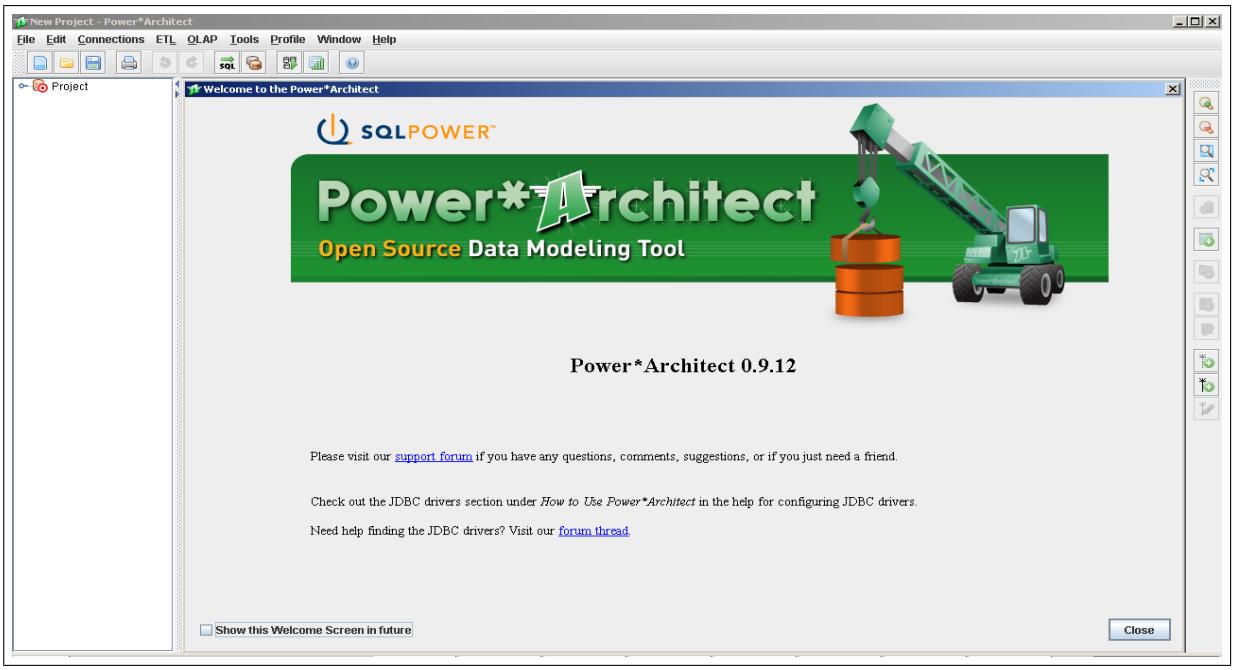

Figura 2.19: *Interface padrao da ferramenta Power Architect*

As suas principais funcionalidades são:

1. Boa interface visual com diagramas de ER e OLAP.
2. Acesso a bancos de dados via JDBC.
3. Conexão a múltiplas fontes de dados ao mesmo tempo.
4. Comparação de modelos de dados com identificação de diferenças e geração de scripts de sincronização.
5. Suporte a “Drag-and-drop”de tabelas de um modelo para o playpen (diagrama).
6. Geração de banco de dados através do projeto (Forward-engineers).
7. Engenharia reversa para a construção dos projetos.
8. Salvamento da estrutura de dados em um projeto que pode ser trabalhado remotemante.

9. Dados do projeto armazenados em um formato XML de fácil navegação.

10. Licença livre GPL (version 3).

11. Suporte rudimentar a ETL - Extração, Transformação e Carga de Dados.

12. Suporte a esquemas OLAP para projeto de Data Warehouses.

As funcionalidades largamente exploradas no desenvovimento do E-Commerce Filito foram: 2, 4, 6, 8 e 9.

A Figura 2.20 exibe a interface que foi utilizada para executar engenharia reversa do banco de dados da aplicação E-Commerce Filito, ou seja, a partir das construções realizadas graficamente a ferramenta gerou todos os scripts necessários e os executou para que a base de dados pudesse existir. Nada de trabalho braçal nesta parte de desenvolvimento !

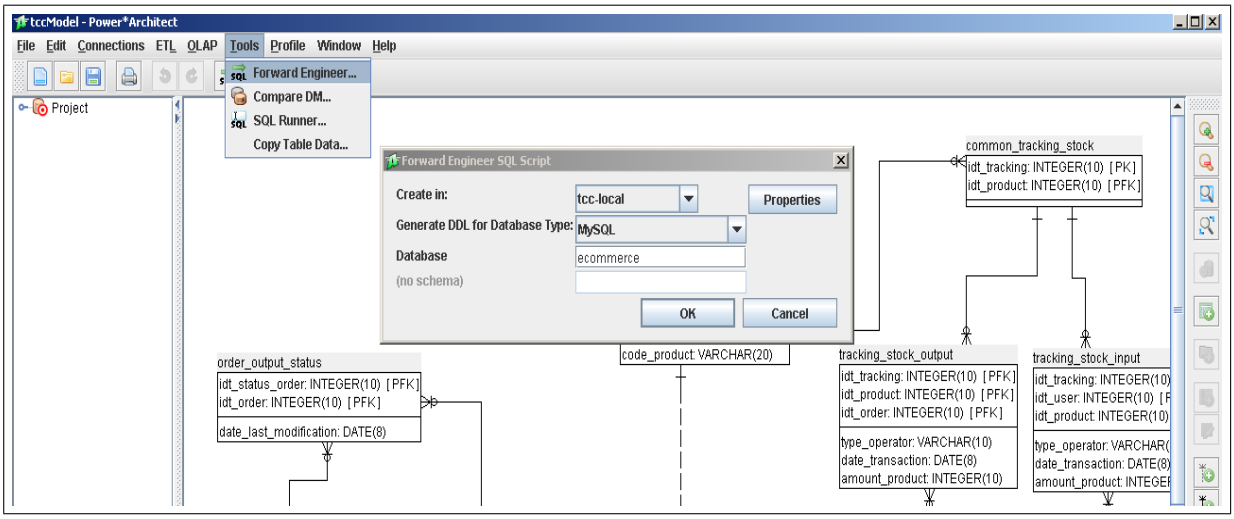

Figura 2.20: *Interface engenharia reversa Power Architect*

Argo UML

ArgoUML é uma aplicação open source que usa UML para modelar o desenho de software de computador. *ArgoUML*[18] pode ser executado na maior parte das platafor-

[18]A versão ArgoUML utilizada é a 0.28.1

mas uma vez que é implementada em Java. Está distribuída sob a licença BSD. Providencia suporte para quase todos os tipos de diagrama da UML padrão e inclui suporte cognitivo. A Figura 2.21 exibe o diagrama de Casos de Uso desenvolvido na aplicação E-Commerce Filito.

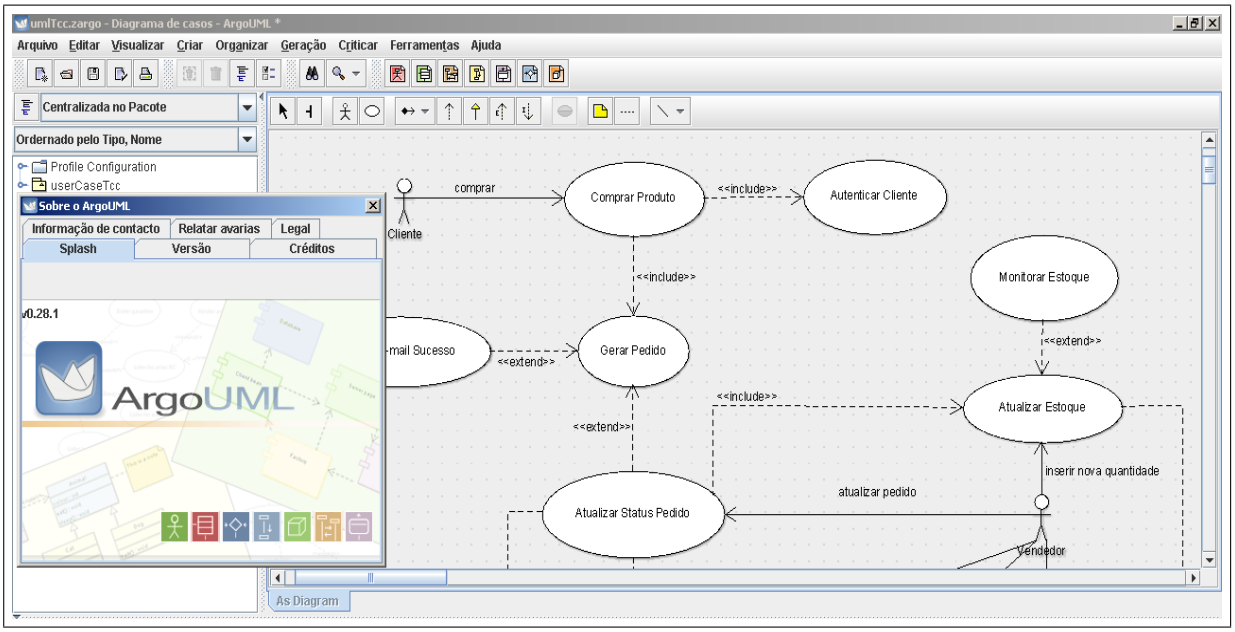

Figura 2.21: *Construcao de diagrama Caso de Uso com Argo UML*

Capítulo 3

Especificação E-Commerce Filito

Esta seção tem por objetivo construir a especificação de um fluxo de compra por completo para o site de E-Commerce Filito, possibilitando aos clientes comprar os produtos disponíveis a qualquer hora. Utilizando uma infra-estrutura de comunicações adequada, um usuário - o Cliente - através de um browser fará solicitações que serão repassadas e processadas por um servidor J2EE e em seguida, serão retornadas adequadamente para o usuário.

3.1 Diagrama Casos de Uso

Utilizando a notação UML (*Unified Modeling Language*) para representar os componentes de serviços especificados nessa aplicação, é apresentado na Figura 3.1 o diagrama de Casos de Uso relacionados ao fluxo de compra. Nele estão figurados os Casos de Uso e suas interações com o usuário Cliente e com o usuário Vendedor, sendo uma ferramenta essencial na captura de requisitos e no planejamento e controle do projeto [6].

Os seguintes passos serão necessários para que um fluxo de compra seja iniciado e concluído com sucesso:

Passo 1) Através da URL *http://filito.com.br* o Cliente acessa a página de compra de produtos;

Passo 2) Na página exibida, o Cliente deve selecionar o produto que deseja comprar, a quantidade desejada e a forma de pagamento;

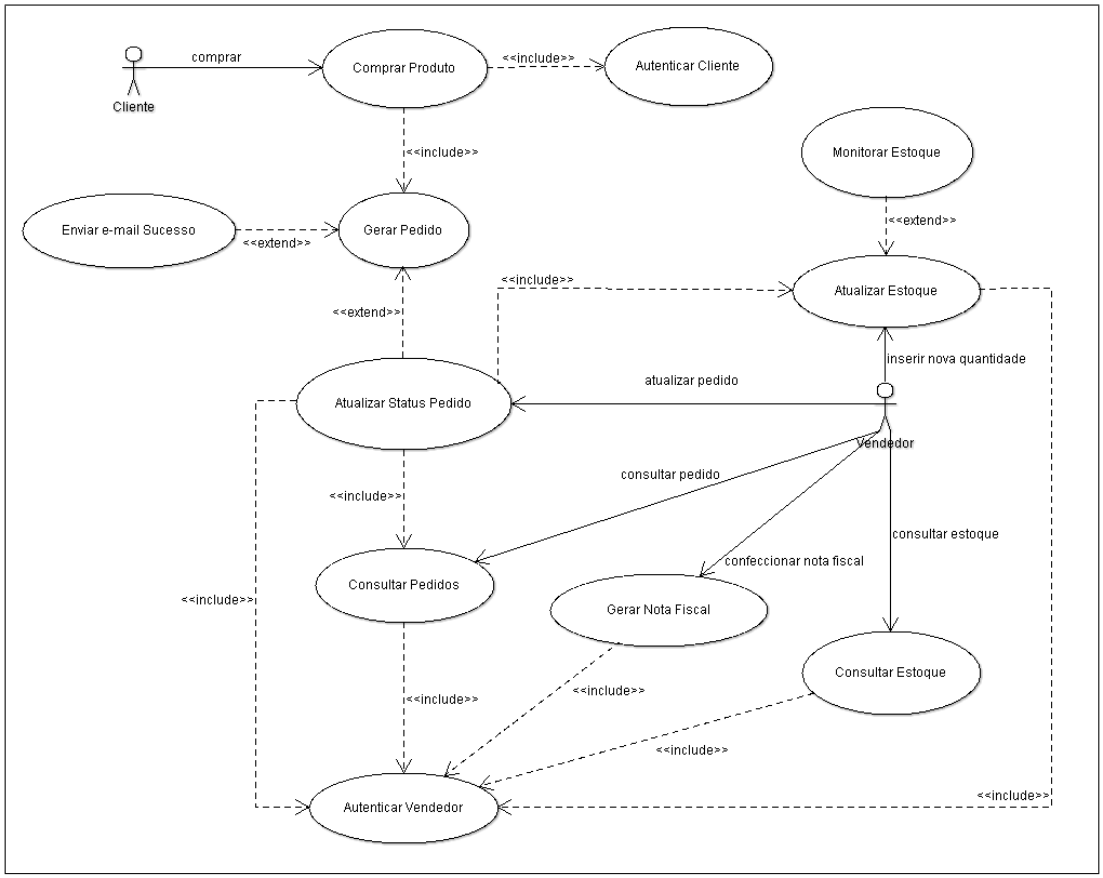

Figura 3.1: *Diagrama de Casos de Uso*

Passo 3) O Cliente clica em “Finalizar Compra”;

Passo 4) A aplicação gera um “Novo Pedido”com status “Pendente”;

Passo 5) A aplicação envia um e-mail para o Cliente informando detalhes da compra;

Passo 6) Uma página com dizeres “Compra efetuada com sucesso”é exibida para o mesmo. Ainda nessa mesma página é exibida uma mensagem dizendo que um e-mail foi encaminhado;

Passo 7) O Vendedor atualiza o pedido descrito no passo 4 para “Aprovado”;

Passo 8) A aplicação gera uma Nota Fiscal para essa compra efetuada;

Passo 9) O Vendedor encerra o fluxo de compra ao atualizar o status do pedido descrito nos passos 4 e 7 para “Finalizado”;

Conforme observa-se no diagrama apresentado na Figura 3.1 cada Caso de Uso realiza uma função específica e bem definida conforme está detalhado abaixo:

- **Caso de Uso: Comprar Produto**

  Descrição: O Cliente através de um navegador de internet acessa a página de compra de produtos.

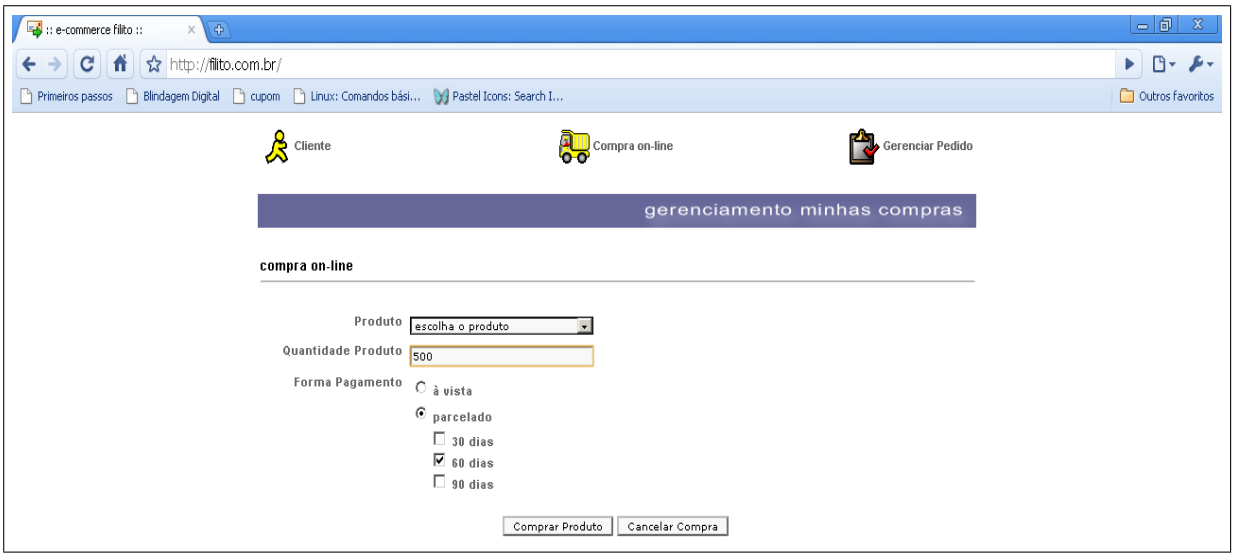

Figura 3.2: *Pagina referente a comprar produto*

Ator Principal: Cliente

Pré-condições: O Cliente deve estar autenticado na aplicação (ver Caso de Uso "*Autenticar Cliente*")

Fluxo Principal:

1. O Cliente acessa a página de compra de produtos através do endereço http://filito.com.br.
2. O Cliente informa os dados para a aplicação.
3. A aplicação valida os dados informados pelo Cliente.
4. A aplicação retorna uma página de sucesso para o Cliente.
5. Caso de Uso se encerra.

Fluxo Alternativo:

1.1) O Cliente é redirecionado para a página de autenticação. Cliente informa os dados de acesso "*username*"e "*senha*"e fluxo retorna para o passo 1.

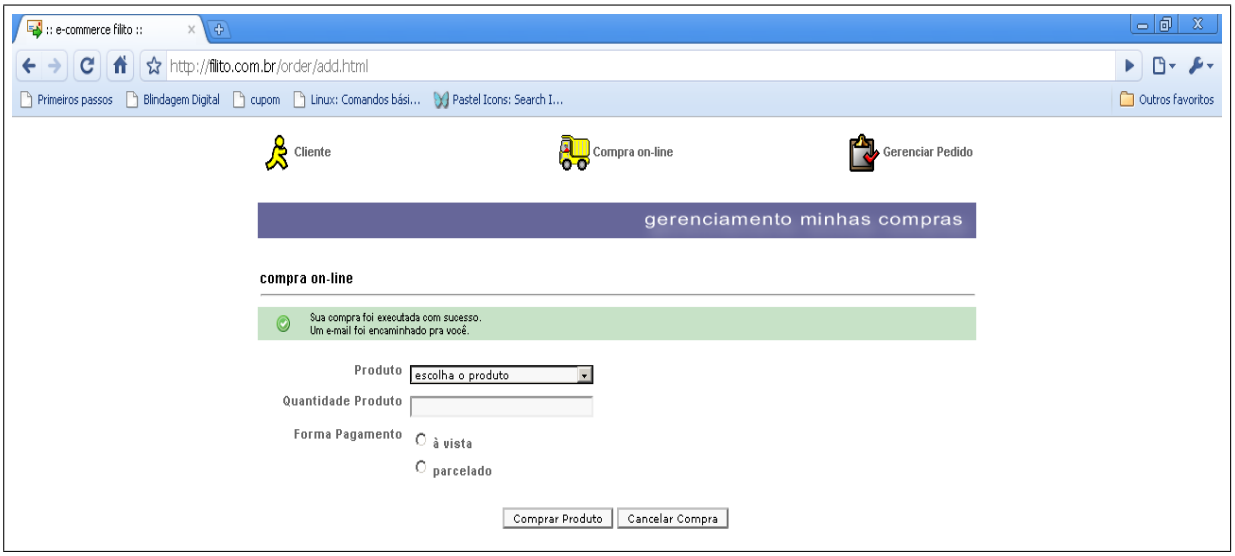

Figura 3.3: *Pagina compra efetuada com sucesso*

1.1.1) Cliente informa dados de acesso "*username*"e "*senha*"erroneamente. Fluxo é redirecionado para o passo 1.1 e uma mensagem sobre o ocorrido é exibida na página.

3.1) Cliente informa dados não válidos para a aplicação. Fluxo é redirecionado para o passo 2 e uma ou mais mensagens dizendo os dados inválidos são exibidas na página.

4.1) Ocorreu uma falha ao salvar as informações inseridas pelo usuário. Cliente é redirecionado para uma página de erro que informa o mesmo sobre o ocorrido e fluxo é redirecionado para o passo 5.

- **Caso de uso: Autenticar Cliente**

  Descrição: O Cliente através de um navegador de internet acessa a página de autenticação.

  Ator Principal: Cliente.

  Pré-condições: Os dados de acesso do Cliente já existem na base de dados.

  Fluxo Principal:

  1. O Cliente acessa a página de autenticação através do endereço: http://filito.com.br/login.html.

2. O Cliente informa o seu “login” e “senha” para aplicação.
3. O Cliente é redirecionado para a página de compra de produtos.
4. Caso de Uso se encerra.

Fluxo Alternativo:

2.1) O Cliente informou seu “login” e “senha” não válidos. Fluxo é redirecionado para o passo 1 e uma mensagem sobre o ocorrido é exibida na página.

3.2) Caso o Cliente já esteja em uma página da aplicação e a sua sessão venha a expirar o mesmo será redirecionado para o passo 1 e posteriormente a aplicação o redirecionará para a página de origem.

3.3) Ocorreu uma falha ao consultar as informações inseridas pelo Cliente. Cliente é redirecionado para uma página de erro que informa o mesmo sobre o ocorrido e fluxo é redirecionado para o passo 4.

- **Caso de Uso: Autenticar Vendedor**

  Descrição: O Vendedor através de um navegador de internet acessa a página de autenticação.

  Ator Principal: Vendedor.

  Pré-condições: Os dados de acesso do Vendedor já existem na base de dados.

  Fluxo Principal:

  1. O Vendedor acessa a página de autenticação através do endereço: http://adm.filito.com.br/login.html.
  2. O Vendedor informa o seu “login” e “senha” para aplicação.
  3. O Vendedor é redirecionado para a página de gerenciamento de pedidos.
  4. Caso de Uso se encerra.

  Fluxo Alternativo:

  2.1) O Vendedor informou seu “login” e “senha” não válidos. Fluxo é redirecionado para o passo 1 e uma mensagem sobre o ocorrido é exibida na página.

3.1) Caso o Vendedor já esteja em uma página da aplicação e a sua sessão venha a expirar o mesmo será redirecionado para o passo 1 e posteriormente a aplicação o redirecionará para a página de origem.

3.2) Ocorreu uma falha ao consultar as informações inseridas pelo Vendedor. Vendedor é redirecionado para uma página de erro que informa o mesmo sobre o ocorrido e fluxo é redirecionado para o passo 4.

- **Caso de Uso: Gerar Pedido**

  Descrição: A aplicação gera um novo pedido na plataforma e-commerce.

  Ator Principal: Scheduler Gerar Pedido.

  Pré-condições: Cliente deve efetuar um fluxo de compra com sucesso.

  Fluxo Principal:

  1. Scheduler Gerar Pedido recupera as informações da compra que o Cliente inseriu.
  2. Scheduler Gerar Pedido gera um novo pedido na base de dados com status "*Pendente*".
  3. Caso de Uso se encerra.

  Fluxo Alternativo:

  2.1) Ocorreu uma falha ao gerar novo pedido na base de dados. A falha ocorrida é registrada em um "arquivo de log" e um e-mail é enviado para o Administrador do Sistema informando o ocorrido. Feito isso o fluxo é redirecionado para o passo 3.

- **Caso de uso: Enviar e-mail Sucesso**

  Descrição: A aplicação envia um e-mail para Cliente.

  Ator Principal: Scheduler Enviar E-mail.

  Pré-condições: Scheduler Gerar Pedido foi executado com sucesso.

  Fluxo Principal:

1. Scheduler Enviar E-mail recupera as informações da compra que o Cliente inseriu e também o e-mail do Cliente logado.
2. Scheduler Enviar E-mail envia um e-mail para o Cliente autenticado com detalhes compra efetuada.
3. Caso de Uso se encerra.

Fluxo Alternativo:

1.1, 2.1) Ocorreu uma falha ao enviar e-mail para o Cliente. A falha ocorrida é registrada em um "arquivo de log" e um e-mail é enviado para o Administrador do Sistema informando o ocorrido. Feito isso o fluxo é redirecionado para o passo 3.

- **Caso de uso: Atualizar Status Pedido**

Descrição: O Vendedor atualiza o status de um pedido.

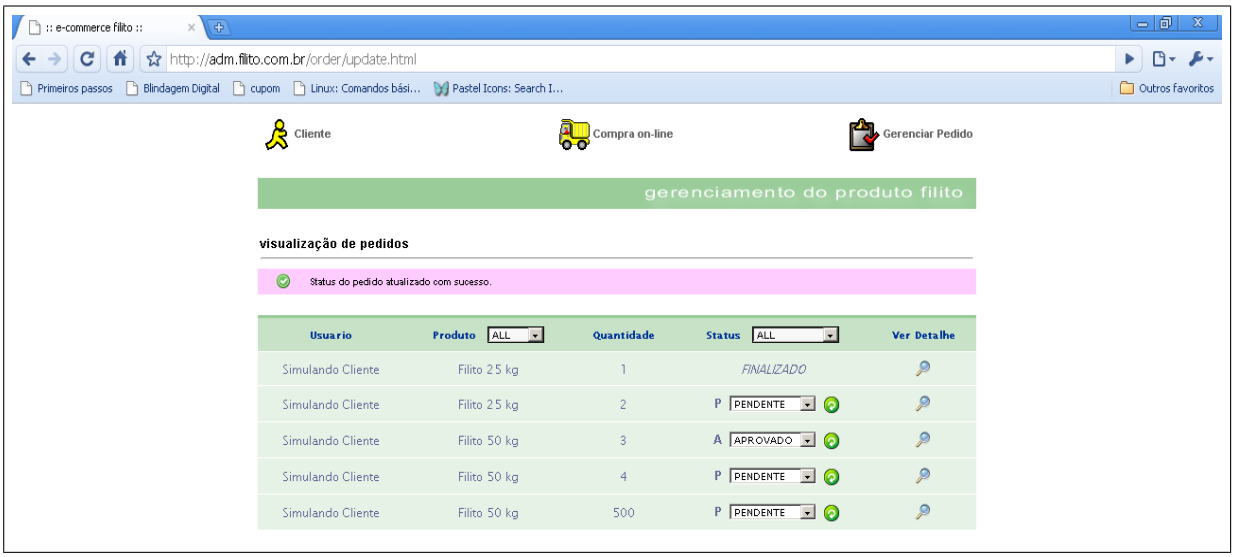

Figura 3.4: *Status do pedido atualizado*

Ator Principal: Vendedor.

Pré-condições:

Deve existir pedido com status "Pendente" ou "Aprovado".

O Vendedor deve estar autenticado no sistema e-commerce.

Fluxo Principal:

1. O Vendedor acessa a página de lista de pedidos que estão com status "*Pendente*" e "*Aprovado*" através do endereço: http://adm.filito.com.br/listOrder.html&update=true
2. O Vendedor escolhe um pedido na listagem que é exibida na página.
3. O Vendedor altera o status do pedido para "*Aprovado*" ou "*Finalizado*".
4. Caso de Uso se encerra.

Fluxo Alternativo:

1.1) Ocorreu uma falha ao acessar a página que lista os pedidos. A falha ocorrida é registrada em um "arquivo de log" e uma página de erro que informa o Vendedor sobre o ocorrido é exibida. Feito isso o fluxo é redirecionado para o passo 4.

2.1) Ocorreu uma falha ao carregar informações do pedido escolhido. A falha ocorrida é registrada em um "arquivo de log" e uma página de erro que informa o Vendedor sobre o ocorrido é exibida. Feito isso o fluxo é redirecionado para o passo 4.

3.1) Ocorreu uma falha ao salvar o novo status do pedido na base de dados. A falha ocorrida é registrada em um "arquivo de log" e uma página de erro que informa o Vendedor sobre o ocorrido é exibida. Feito isso o fluxo é redirecionado para o passo 4.

- **Caso de uso: Consultar Pedidos**

  Descrição: O Vendedor através de um navegador de internet acessa a página que contêm uma lista de pedidos.

  Ator Principal: Vendedor.

  Pré-condições: O Vendedor deve estar autenticado no sistema e-commerce.

  Fluxo Principal:

  1. O Vendedor acessa a página de lista de pedidos através do endereço: http://adm.filito.com.br/listOrder.html.
  2. Caso de Uso se encerra.

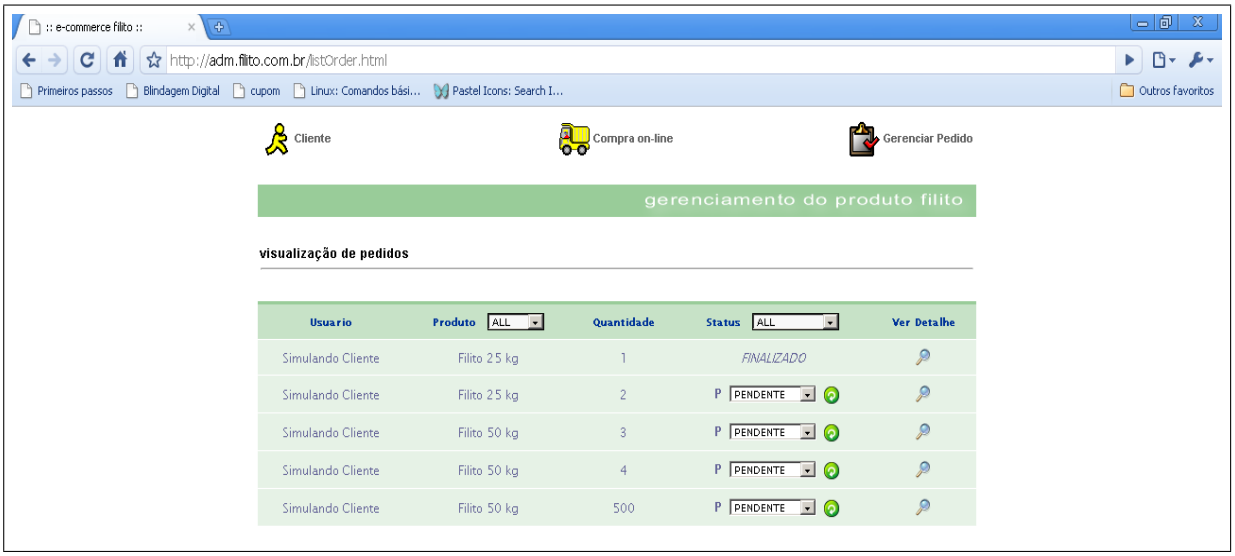

Figura 3.5: *Lista com pedidos gerados*

Fluxo Alternativo:

1.1) Ocorreu uma falha ao acessar a página que lista os pedidos. A falha ocorrida é registrada em um "arquivo de log" e uma página de erro que informa o Vendedor sobre o ocorrido é exibida. Feito isso o fluxo é redirecionado para o passo 2.

- **Caso de uso: Gerar Nota Fiscal**

  Descrição: O Vendedor através de um navegador de internet acessa a página que contêm uma lista de pedidos com status "*Finalizado*".

  Ator Principal: Vendedor.

  Pré-condições: O Vendedor deve estar autenticado no sistema e-commerce.

  Fluxo Principal:

  1. O Vendedor acessa a página de lista de pedidos já finalizados através do endereço: http://adm.filito.com.br/listOrder.html&status=finish.
  2. O Vendedor seleciona um pedido e clica em "*Gerar Nota Fiscal*".
  3. É exibido para o Vendedor uma pré-visualização da impressão.
  4. O Vendedor aprova a impressão clicando em "*Imprimir Nota*".
  5. A Nota Fiscal é impressa.

6. Caso de Uso se encerra.

Fluxo Alternativo:

1.1) Ocorreu uma falha ao acessar a página que lista os pedidos. A falha ocorrida é registrada em um “arquivo de log” e uma página de erro que informa o Vendedor sobre o ocorrido é exibida. Feito isso o fluxo é redirecionado para o passo 2.

- **Caso de uso: Consultar Estoque**

  Descrição: O Vendedor através de um navegador de internet acessa a página que exibe a quantidade de produtos existentes no estoque.

  Ator Principal: Vendedor.

  Pré-condições: O Vendedor deve estar autenticado no sistema e-commerce.

  Fluxo Principal:

  1. O Vendedor acessa a página que exibe a quantidade de produtos existe no estoque através do endereço: http://adm.filito.com.br/consultStock.html.
  2. Caso de Uso se encerra.

  Fluxo Alternativo:

  1.1) Ocorreu uma falha ao acessar a página que exibe produtos em estoque. A falha ocorrida é registrada em um “arquivo de log” e uma página de erro que informa o Vendedor sobre o ocorrido é exibida. Feito isso o fluxo é redirecionado para o passo 2.

- **Caso de uso: Atualizar Estoque**

  Descrição: O Vendedor através de um navegador de internet acessa a página que permite a ele inserir nova quantidade de produtos no estoque ou o Scheduler Atualizar Estoque vai atualizar o estoque sempre que um pedido for atualizado para status “*Finalizado*”.

  Ator Principal: Vendedor ou Scheduler Atualizar Estoque.

  Pré-condições:

  O Vendedor deve estar autenticado no sistema e-commerce.

Status do pedido deve ser atualizado para “*Finalizado*”.

Fluxo Principal:

1. O Vendedor acessa o endereço http://adm.filito.com.br/addStock.html que permite a ele adicionar no estoque a quantidade de produtos que desejar.
2. O Vendedor informa para a aplicação a quantidade de produto a ser adicionada e também qual é o tipo do produto.
3. A aplicação valida as informações inseridas.
4. A aplicação salva na base de dados as informações inseridas pelo Vendedor.
5. Caso de Uso se encerra.

Fluxo Alternativo:

1.1) Ocorreu uma falha ao acessar a página que permite o Vendedor adicionar produtos no estoque. A falha ocorrida é registrada em um “arquivo de log” e uma página de erro que informa o Vendedor sobre o ocorrido é exibida. Feito isso o fluxo é redirecionado para o passo 5.

1.2) O Scheduler Atualizar Estoque ao verificar que existe um novo pedido com status “*Finalizado*” debita do estoque a quantidade de produtos referentes ao especificado no pedido e o fluxo é redirecionado para o passo 5.

3.1) Os dados informados não são válidos. O fluxo é redirecionado para o passo 1 e uma mensagem sobre o dado inválido é exibida para o Vendedor.

4.1) Ocorreu uma falha ao salvar as informações na base de dados. A falha ocorrida é registrada em um “arquivo de log” e uma página de erro que informa o Vendedor sobre o ocorrido é exibida. Feito isso o fluxo é redirecionado para o passo 5.

3.2 Cenário da aplicação desenvolvida

Para o desenvolvimento da aplicação E-Commerce Filito, foi utilizada uma máquina cliente web para acessar por meio da Internet e uma outra máquina servidora, que possui como infra-estrutura o framework Struts utilizando os componentes J2EE, dois

“containers”: cliente e administrativo - responsáveis por gerenciar a execução destes componentes e um gerenciador de banco de dados, contendo as informações dos produtos oferecidos pelo site. A Figura 3.6 ilustra o cenário de uso acima descrito.

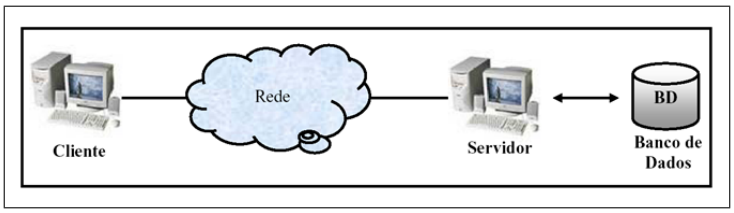

Figura 3.6: *Cenario utilizado no desenvolvimento*

Capítulo 4

Resultados e Trabalhos Futuros

O objetivo desse trabalho foi implementar um software que seguisse os padrões web existentes atualmente e ao mesmo tempo colocar em prática teorias e conceitos assimilados durante os anos acadêmicos. Este software tinha como um dos requisitos a disponibilidade via internet.

Dessa forma foi iniciado um estudo sobre a plataforma J2EE entre outros. Após um curto tempo de estudo a plataforma J2EE foi a escolhida por oferecer sólidas e completas ferramentas de desenvolvimento atrelados ao fato da maioria delas serem isentas de licenças.

4.1 Contribuições

Durante o desenvolvimento desse trabalho escrito duas ferramentas foram utilizadas. São elas:

GIMP: O *GIMP* (GNU Image Manipulation Program) é um programa de código aberto voltado principalmente para criação e edição de imagens raster, e em menor escala também para desenho vetorial.

LATEX: O *LATEX* é um conjunto de macros para o processador de textos (TeX), utilizado amplamente para a produção de textos matemáticos e científicos devido à sua alta qualidade tipográfica. Entretanto, também é utilizado para produção de artigos e livros sobre assuntos muito diversos.

4.2 Resultados Obtidos

Escolhido o tipo de plataforma era chegada a hora de começar o desafio de programar códigos orientados a objetos em Java mas antes foi preciso arquitetar e modelar toda a aplicação E-Commerce Filito.

Muitos foram os obstáculos durante o desenvolvimento dessa aplicação. Desenvolver em Java utilizando a plataforma J2EE sempre foi também conhecer as API's que nem sempre se mostraram fáceis de assimilar, como o Struts por exemplo. Contudo foi extraordinário o resultado positivo que pude obter durante essa longa jornada de trabalho.

Por fim, deve-se dizer que ao desenvolver e implementar a aplicação E-Commerce Filito, consolidou conhecimentos e trouxe nova visibidade como estudante e profissional.

4.3 Trabalhos Futuros

Como trabalho futuro relacionado ao tema estudado deve-se destacar:

O código da aplicação até aqui desenvolvida contempla apenas os Casos de Uso: Comprar Produto, Autenticar Cliente, Atualizar Status Pedido e Atualizar Estoque. Portanto os próximos passos serão implementar todos os Casos de Uso restantes.

Neste estudo não foi abordado nenhuma engenharia de teste de software, apenas testes executados pelo próprio desenvolvedor(eu). Portanto, estudar e aplicar testes aprofundados e automatizados sobre todo o código implementado é uma próxima meta.

Todo esse trabalho bem como as novas implementações estão sendo disponibilizados no seguinte endereço eletrônico: http://bizzuly.blogspot.com/.

Referências Bibliográficas

[1] *JDBC - Documentacao da tecnologia JDBC, disponivel em http://java.sun.com/products/jdbc/index.jsp no dia 18 de junho de 2007.*

[2] Khawar Zaman AHMED and Cary E UMRYSH. *Desenvolvendo Aplicacoes Comerciais em Java com J2EE e UML*. Ciencia Moderna, 2002.

[3] Peter COAD. Object-oriented patterns. communications of the acm, v.35, n 9, p. 152-159, setembro de 1992.

[4] Harvey M. DEITEL and Paul J. *Java: Como Programar*. 5º edição. Prentice Hall, 2002.

[5] Martin Fowler. *Inversion of Control Containers and the Dependency Injection pattern disponivel em http://www.ime.usp.br/ andrers/aulas/bd2005-1/aula3.html e acessado em 13 de novembro de 2009.*

[6] Jose Davi FURLAN. *Modelagem de Objetos Atraves da UML - The Unified Modeling Language.* 1998.

[7] Erich GAMMA, Richard HELM, Ralph JOHNSON, and John VLISSIDES. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[8] Jader and Jonathan. Disponivel em: http://indecisos.wordpress.com/2006/09/13/rup-rational-unified-process-ou-processo-unificado-da-rational/ no dia 08 de novembro de 2009.

[9] G. King and C. Bauer. *Hibernate in Action*. Manning Publications Co., 2005.

[10] Daniel Fernandes Martins. Disponivel em: http://javafree.uol.com.br/artigo/871446/ no dia 08 de janeiro de 2010.

[11] Sun Microsystems. Disponivel em: http://java.sun.com/j2ee/1.3/docs/tutorial/doc/overview3.html no dia 08 de janeiro de 2010.

[12] Sun Microsystems. Disponivel em: http://java.sun.com/j2ee/1.3/docs/tutorial/doc/overview4.html no dia 08 de janeiro de 2010.

[13] Sun Microsystems. *Visao Geral do Sistema de Gerenciamento de Banco de Dados MySQL disponivel em http://dev.mysql.com/doc/refman/4.1/pt/what-is.html e acessado em 13 de novembro de 2009.*

[14] KRUCHTEN P. *The Rational Unified Process: An Introduction*. Addison-Wesley, 1999. Capitulo 1 - ”Software Development Best Practices”.

[15] Andre Rodrigo Sanches. *Fundamentos de Armazenamento e Manipulacao de Dados disponivel em http://www.ime.usp.br/ andrers/aulas/bd2005-1/aula3.html e acessado em 13 de novembro de 2009.*

[16] HUSTED Ted, DUMOULIN Cedric, FRANCISCUS George, and WINTERFELDT David. *Struts in Action*. Manning Publications Co., 2004.

[17] Craig Walls. *Spring in Action*. Manning Publications Co., 2008.

[18] Wikipedia. *Injecao de dependencia disponivel em http://pt.wikipedia.org/wiki/Injecao_de_dependencia e acessado em 13 de novembro de 2009.*