JCommerceNet: Um *Framework* usando aspectos na geração de scripts para camada de persistência para automação de lojas virtuais

Andrêsa Vargas de Almeida

UNIVERSIDADE DO VALE DO RIO DOS SINOS
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
CURSO DE INFORMÁTICA

JCommerceNet: Um *Framework* usando aspectos para geração de scripts para camada de persistência para automação de lojas virtuais

Andrêsa Vargas de Almeida

Prof. Dr. Sérgio Crespo Coelho da Silva Pinto
Orientador

Monografia submetida como requisito parcial para a obtenção do título de Bacharel em Informática.

São Leopoldo, novembro de 2005.

Dedico esta monografia a duas pessoas em especial: meu pai, Vilmar Vargas de Almeida e minha mãe, Sônia Maria de Almeida. Vocês, que são minha fonte de vida, me apoiaram o tempo todo para que este trabalho se realizasse, por isso dedico esta conquista a vocês dois. Amo vocês!

AGRADECIMENTOS

Expresso meus agradecimentos a todas as pessoas que me auxiliaram ao longo da minha vida acadêmica e que contribuíram para a realização deste trabalho:

- Aos meus pais, Vilmar e Sônia, pela força e incentivo constantes para a conquista de meus objetivos e, especialmente, pela compreensão nos momentos de ausência;

- Ao meu irmão, por não me interromper nas horas de estudo e deixar livre o computador;

- Ao meu orientador, Professor Dr. Sérgio Crespo Coelho da Silva Pinto, pela oportunidade de trabalhar com uma pessoa extremamente competente, pela orientação ao trabalho, pelo incentivo e, principalmente, pela amizade já conquistada desde a época do trabalho junto à pesquisa;

- À Unisinos, por ser a instituição responsável por garantir esta conquista, e pela responsabilidade e apoio aos seus alunos;

- Aos meus colegas de graduação, que ao longo desses sete anos de faculdade sempre estiveram presentes, com alegria e companheirismo, tornando nossa vida acadêmica mais agradável;

- Aos meus amigos, que muito ouviram falar no meu trabalho de conclusão ao longo deste ano, e que devem estar cansados, mas afirmo que foi por uma boa causa;

- Ao amigo Alberto Egon Shaeffer Filho, pelas dicas que me deu no decorrer do trabalho;

- Ao amigo Mickaël Joseph Thomas Bordeleau Jalbert pela ajuda na revisão do resumo;

- A Deus, pela saúde, pela luz, pelos momentos de paz e pela força que me deu durante todos esses anos que passei na faculdade e, principalmente, neste último ano, onde a dedicação foi integral para que este trabalho se realizasse.

RESUMO

Construir *Web sites* comerciais é uma tarefa complexa que requer um esforço para entendimento de negócios, de tecnologia e questões sociais, tão bem como uma abordagem sistemática. Em virtude do crescimento do comércio eletrônico e do interesse de muitas empresas em possuir um *Web site* comercial para comercialização de seus produtos e serviços, torna-se necessária a utilização de tecnologias de Engenharia de *Software*, a fim de padronizar e aumentar a confiabilidade do desenvolvimento de sistemas para a Internet.

Esta monografia apresenta o desenvolvimento de um *framework*, usando aspectos na geração de scripts para camada de persistência para automação de lojas virtuais de pequeno porte na Internet. JCommerceNet utiliza tecnologia de *frameworks*, padrões de projeto e aspectos para a geração de scripts, dirigidos ao desenvolvimento de aplicações que têm por objetivo a construção de lojas virtuais na Internet.

Um estudo de caso foi implementado, de forma a consolidar o *framework*, os métodos de geração de scripts para camada de persistência e validar a sua utilização no mundo real. O estudo de caso utilizou todos os métodos do *framework* e instanciou alguns *hot-spots*, abordando todos os aspectos levantados na análise de requisitos.

ABSTRACT

Build an e-commerce Web site is a complex task that requires an effort to understand the business, the technology, the social matters and a systematic boarding. The growth of the e-commerce business and the interest of many companies in possessing a commercial Web site to commercialize its products and services create the need to use the technologies of Engineering of Software in order to standardize and to increase the trustworthiness of the development of systems for the Internet.

This monograph presents the development of a framework using aspects in the generation of scripts for persistence layer to automate a virtual store of small transport on Internet. JCommerceNet uses technology of frameworks, design patterns and aspects for the generation of scripts, directed to the development of applications that have as objective the construction of virtual store on Internet.

A case study has been implemented to consolidate the framework, the methods of generation of scripts for persistence layer and to validate its use in the real world. The case study used all the methods of the framework and extends some hot-spots, approaching all the aspects raised in the requirements analysis.

SUMÁRIO

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

LISTA DE ABREVIATURAS

AJDT	*AspectJ Development Tools*	
B2B	*Business to Business*	Negócios entre empresas
B2C	*Business to Consumer*	Negócios entre empresas e consumidores
C2B	*Consumer to Business*	Negócios entre consumidores e empresas
C2C	*Consumer to Consumer*	Negócios entre consumidores
CRUD	*Create,Read,Update,Delete*	Criação, Leitura, Atualização e Remoção
EDI	*Eletronic Data Interchange*	Troca eletrônica de dados
EJB	*Enterprise Java Bean*	
GOF	*The Gang of Four*	
IBGE		Instituto Brasileiro de Geografia e Estatística
J2EE	*Java 2 Platform Enterprise Edition*	
JDK	*Java Development Kit*	
JSP	*Java Server Pages*	
MVC	*Model View Controller*	
PAC	*Presentation Abstration Controller*	
PARC	*Palo Alto Research Center*	
POA	*Aspect Oriented Programming*	Programação orientada a aspectos
SEBRAE		Sistema Brasileiro de Apoio às Micro e Pequenas empresas
OECD	*Organization of Economic Co-operation and Development*	
OO		Orientado a objetos
AO		Orientado a aspectos
SQL	*SQL Code Description*	
TI		Tecnologia de Informação
UML	*Unified Modeling Language*	
WWW	*Word Wide Web*	

1 INTRODUÇÃO

A popularidade da Internet fez com que a prática do comércio eletrônico também se popularizasse. Segundo LAUDON e TRAVER [42], o comércio eletrônico é ubíquo, ou seja, está disponível em todo lugar, em todo tempo, tornando possível a compra através do computador, de casa, do trabalho, ou até mesmo do automóvel. Muito se discute sobre seu potencial e seu impacto nos hábitos dos consumidores, mostrando sempre previsões muito otimistas. Números e estatísticas indicam que o comércio baseado na Internet é mais eficiente e possivelmente mais efetivo do que o realizado pelos meios tradicionais [26].

Alguns fatores contribuem para o crescimento mundial do comércio eletrônico, como: baixo custo de aquisição, redução e inventários mais exatos, baixo período de tempo entre transmissão e recebimento de informações, aperfeiçoamento dos serviços de clientes, diminuição de vendedores e custos de marketing, e novas oportunidades de vendas [15].

A Internet e o comércio eletrônico têm beneficiado em grande escala as organizações e seus clientes, por exemplo: na realização de transações comerciais, na aproximação de compradores e vendedores, na diminuição de custos, na personalização de serviços e na automação de tarefas, mas enfrentam também os desafios que podem limitar o crescimento. Executivos Europeus citam exemplos como: segurança, privacidade e a falta de uma estrutura regulamentar comercial como as principais barreiras do comércio eletrônico [15]. Outro importante fator, abordado por Fortunato [26] é o processo de desenvolvimento de *Web sites*, incluindo decisões associadas à plataforma, às inúmeras opções de projetos e à velocidade com que quaisquer umas dessas alternativas ficam obsoletas, aliadas aos custos intrínsecos, acabam levando muitos a esperar por uma solução mais consolidada, ou seja, um padrão de mercado. Assim, dificuldades em definir uma especificação e implementar funcionalidades ainda são problemas enfrentados por quem deseja ter uma presença comercial na Internet [26].

Este trabalho vem trazer uma contribuição neste sentido, oferecendo ferramentas para a geração de scripts para automação de lojas virtuais de pequeno porte na Internet.

Com o crescimento da comunidade na Internet, surgem novas maneiras de comercializar produtos, e a loja virtual oferece a opção de comércio eletrônico dirigida a pequenos, médios e grandes negócios. Uma loja virtual pertence ao segmento de comércio eletrônico B2C (Negócios entre empresas e consumidores), onde são ofertados produtos diretamente para o consumidor final.

O modelo do *framework*, que será projetado através deste domínio de aplicação, será destinado à empresas de pequeno porte que desejam expandir os limites físicos de suas lojas e vender para o mercado nacional durante 24 horas por dia, 365 dias/ano com custo operacional baixo. Um conjunto de soluções destinadas à pequena empresa seria: catálogo de produtos, vendas sugeridas (promoções), busca por produtos, registro de clientes, cadastro de características dos clientes, carrinho de compras virtual, gerenciamento de catálogo de produtos, gerenciamento de produtos, gerenciamento de vendas, gerenciamento de inventário, gerenciamento da loja *on-line*, gerenciamento de clientes, estatísticas sobre os produtos e estatísticas de acesso.

Segundo informações extraídas da pesquisa realizada pelo Instituto Brasileiro de Geografia e Estatística (IBGE) [31], empresas de pequeno porte vêm aumentando gradativamente sua participação no segmento de comércio e serviços, cuja representatividade passou de 95,5% em 1985, para 97,6% em 2001. De forma geral, esse crescimento ocorreu com mais intensidade nas empresas comerciais, refletindo a maior viabilidade do pequeno investidor se estabelecer nessa atividade, em função da exigência de menor volume de investimento e de requisitos de qualificação profissional mais baixos, considerando que este segmento empresarial atende à demanda de necessidades básicas do consumidor, tais como: lojas de roupas, bazares, sapatarias, livros, etc.

Tendo em vista a necessidade de implementar funcionalidades para quem deseja ter uma presença comercial na Internet, definem-se os objetivos do trabalho.

1.1 Objetivos do Trabalho

1.1.1 Objetivo Geral

Desenvolver um *framework* usando aspectos para definição de *hot-spots* na geração de scripts para automação de lojas virtuais de pequeno porte na Internet.

1.1.2 Objetivos Específicos

Estudar as tecnologias envolvidas no trabalho: *frameworks*, padrões de projeto, aspectos e comércio eletrônico.

Analisar sistemas de geração de lojas virtuais existentes e com isso fazer a engenharia reversa que servirá como base para a modelagem do *framework*.

Validar o *framework* através do desenvolvimento de um estudo de caso, onde se realizarão algumas instâncias para os *hot-spots*.

1.2 Estrutura do Trabalho

Além dessa introdução, o trabalho compreende os seguintes capítulos:

O capítulo 2 – **Comércio Eletrônico**: apresenta os principais conceitos de comércio eletrônico, enfatizando o conceito do tipo de comércio eletrônico B2C.

O capítulo 3 – **Padrões de Projeto e *Frameworks***: descreve as tecnologias de Padrões de Projeto e *frameworks*, utilizadas no projeto de desenvolvimento do *framework*.

O capítulo 4 – **Programação Orientada a Aspectos**: descreve os conceitos básicos deste recente paradigma, e também a linguagem de programação orientada a aspectos baseada em Java, "AspectJ", utilizada no projeto de desenvolvimento do *framework*.

O capítulo 5 – **JCommerceNet**: descreve o *framework*. É apresentado um estudo de sistemas de criação, manutenção e utilização de *Web sites* comerciais, de forma a permitir elicitar os principais requisitos para a definição do *framework*. Também é apresentado o projeto e o modelo de classes, destacando os *hot-spots* e o *kernel*.

No capítulo 6 – **Implementação do *Framework***: descreve como o *framework* foi implementado, quais tecnologias foram utilizadas e um exemplo de instanciação.

No capítulo 7 – **Estudo de Caso**: descreve um estudo de caso, de forma a validar o *framework*, os métodos de geração de scripts a camada de persistência e validar a sua utilização no mundo real.

No capítulo 8 – **Conclusões**: é apresentada a conclusão do trabalho e, por fim, seguem as Referências Bibliográficas.

2 COMÉRCIO ELETRÔNICO

O ambiente empresarial, tanto em nível mundial como nacional, tem passado por profundas mudanças nos últimos anos, as quais têm sido consideradas diretamente relacionadas com a Tecnologia de Informação (TI) [3]. O surgimento de novas tecnologias, ou novas aplicações, para atender às necessidades do novo ambiente, até o aparecimento de novas oportunidades empresariais criadas pelas novas tecnologias ou novas formas de sua aplicação são fatores que contribuem para essas mudanças.

Atualmente, algumas das características do novo ambiente empresarial, tais como: globalização, integração interna e externa das organizações, entre outras, têm confirmado as tendências da criação e utilização de mercado e comércio eletrônicos, que já são considerados uma realidade [3].

2.1 Conceito

Segundo ALBERTIN [3], o comércio eletrônico é "a realização de toda a cadeia de valor dos processos de negócios num ambiente eletrônico, por meio da aplicação intensa das tecnologias de comunicação e de informação, atendendo aos objetivos de negócio". A cadeia de valor permite à empresa receber matérias-primas, acrescentar-lhes valor por variados processos e vender produtos transformados aos clientes com uma margem, fornecendo uma avaliação adequada do verdadeiro custo e lucro unitário de cada produto. Salienta ainda que o comércio eletrônico oferece uma infra-estrutura predominantemente pública, de fácil e livre acesso público e baixo custo, onde os processos podem ser realizados de forma completa ou parcial incluindo:

- Transações negócio-a-negócio – facilitam as transações econômicas de interação social, gerenciamento de finanças pessoais, e informações e compra de produtos;
- Transações negócio-a-consumidor – facilitam as aplicações de negócio de gerenciamento de fornecedor, estoque, distribuição, canal e pagamento;
- Transações intra-organizacional – facilitam as aplicações de negócio de comunicações de grupo de trabalho, publicação eletrônica e produtividade da força de vendas.

Segundo LIMEIRA [45], o comércio eletrônico é uma aplicação da Internet que se expandiu exponencialmente nos últimos cinco anos e que deve desenvolver-se a taxas elevadas nos próximos anos. Porém, muito ainda terá de ser feito para se tirar maior proveito de todas as suas potencialidades e das oportunidades por ele oferecidas.

Segundo a *Organization for Economic Co-operation and Development* (OECD)[1], a definição de comércio eletrônico engloba a realização de negócios por meio da Internet, incluindo a venda não

[1] OECD. *E-Commerce: Impacts and policy challenges*. Economics department working papers, n. 252, 23 June 2000. Disponível em: <http://www.olis.oecd.org/olis/2000doc.nsf/linkto/eco-wkp(2000)25>

só de produtos e serviços físicos, entregues *off-line*, isto é, por meios tradicionais, mas também de produtos como *softwares*, que podem ser digitalizados e entregues *on-line*, por meio da Internet.

2.2 Tipos de Comércio Eletrônico

O comércio eletrônico encontra-se dividido em quatro principais segmentos de mercado: B2B (negócios entre empresas), B2C (negócios entre empresas e consumidores), C2B (negócios entre consumidores e empresas) e C2C (negócios entre consumidores) [42] [45]. A figura 1 apresenta alguns exemplos dos quatro segmentos de aplicações de mercado descritas acima:

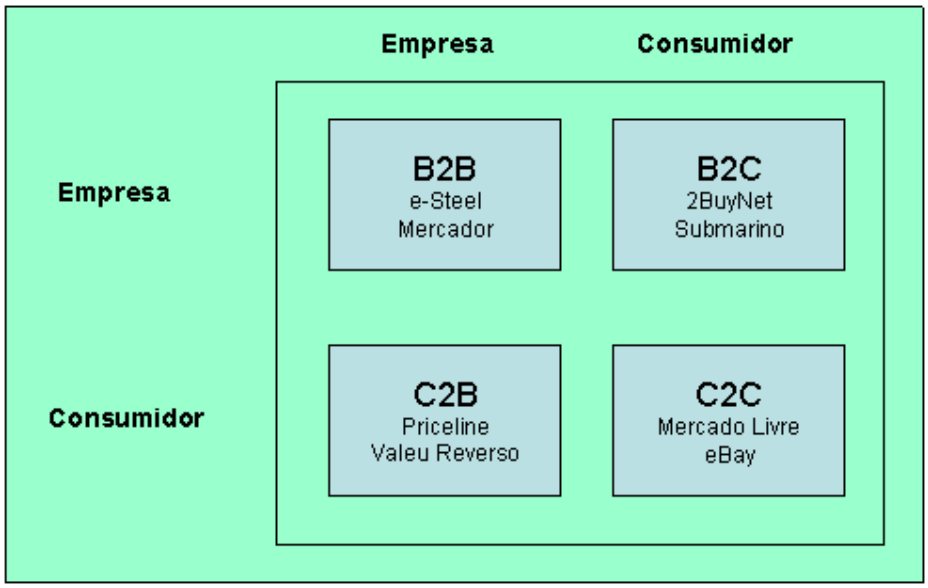


Figura 1 - Exemplos de aplicações de comércio eletrônico por segmento de mercado

- B2B – *Business to Business*: esse segmento faz a realização de negócios entre empresas através da Internet. Com a utilização de novas tecnologias, é possível fazer compras, vendas, pesquisa de preços e manter contatos com fornecedores e clientes. Permite o desenvolvimento de soluções tais como: portais verticais e *Marketplace*[2], sistemas de compra e cotações *on-line*, interfaces seguras para consultas de dados via Internet, leilões virtuais, administração e armazenamento seguro de informações, integração com sistemas existentes, etc. Proporciona redução de tempo, distância, custo e dá agilidade e eficiência nas relações com clientes, fornecedores e parceiros [42]. Como exemplos de aplicações podem-se citar: o Mercador <http://www.mercador.com.br>, que aproxima Supermercadistas e Vendedores de produtos para supermercados e *e-Steel* <http://www.e-Steel.com> que ajuda compradores e vendedores a comprar e vender produtos da indústria do aço.

[2] Marketplace: É um espaço de mercado eletrônico digital onde fornecedores e compradores comerciais podem conduzir transações (KAPLAN, Steven; SAWHNEY, Mohanbir, 2000) [37].

- B2C – *Business to Consumer*: esse segmento é o que concentra o maior número de aplicações disponíveis no comércio eletrônico atualmente na Internet, na qual as empresas ofertam produtos diretamente para o consumidor final (pessoas que navegam na Internet), que desejam comprar algum produto [42]. Fazem parte deste segmento: lojas virtuais <http://www.2buynet.com.br>, provedores de conteúdo <http://www.cnn.com>, provedores de serviços <http://www.xdrive.com>, provedores de comunidades <http://www.about.com>, portais <http://www.yahoo.com>, corretores de transação <http://www.e-trade.com>, vendas de produtos para consumidor final <http://www.amazon.com>.

- C2B – *Consumer to Business*: esse segmento concentra aplicações onde o usuário final define o preço e todas suas intenções de compra sobre um determinado produto, e vendedores competem e jogam com suas margens de lucro para conquistar a venda [45]. Exemplos de aplicações que utilizam este tipo de segmento: *Priceline* <http://www.priceline.com>, que atua no mercado de passagens aéreas, pela qual o usuário final define o preço do bilhete, e Valeu Reverso <http://www.valeu.com.br>, *site* brasileiro que oferece um serviço de leilão, que permite ao comprador fazer seu preço de acordo com algumas categorias disponíveis no leilão.

- C2C – *Consumer to Consumer*: esse segmento permite aos consumidores a venda de produtos a outros consumidores, já que está representado principalmente por leilões. Neste segmento, o consumidor prepara o produto para o mercado, colocando o produto para leiloar ou vender, e confia no fabricante de mercado para fornecer o catálogo, ferramentas de busca, e potencialidades de transações de modo que os produtos possam ser facilmente indicados, descobertos e pagos [42]. Exemplos de aplicações que utilizam este tipo de segmento: *eBay* <http://www.eBay.com>, mercado livre <http://www.mercadolivre.com.br>.

2.3 Valor de Negócio de Comércio Eletrônico

Os benefícios que o comércio eletrônico proporciona ao valor de negócio de uma empresa são apresentados em [3] [11], estando divididos por tipo de benefícios:

- Benefícios diretamente mensuráveis, quantitativos:

1. Promoção de produtos: por meio do contato direto, rico em informação e interativo com os clientes, pode melhorar a promoção dos produtos;
2. Novo canal de vendas: o comércio eletrônico proporciona um alcance direto aos clientes, representando um novo canal de vendas para os produtos existentes, tais como: produtos físicos (*hardware*, vinhos, roupas etc.), e produtos que podem ser adicionalmente entregues utilizando-se um meio de comércio eletrônico (informação, *software* etc.). O comércio eletrônico tem significativo valor nos mercados em que a informação é de valor adicionado significativo para os produtos que estão sendo comprados. Também permite o surgimento da noção de venda de produto de informação, criando novas categorias de produtos. Algumas maneiras de fazer dinheiro utilizando a *Word Wide Web* (WWW) poderiam ser: venda direta (vender produtos), venda de conteúdo (vender informação), anúncios de propaganda

(fornecer gratuitamente alguma informação para certo tipo de cliente e vender espaço de propaganda para os anunciantes), e transações e ligações (cobrar uma taxa para a troca de transações ou cobrar uma taxa para realizar a ligação com provedor de serviços);

3. Economia direta: utilizando uma infra-estrutura pública compartilhada, tal como a Internet, para transmissão e reutilização de informação digital, os sistemas de comércio eletrônico podem reduzir o custo de entrega de informação a clientes;
4. Inovação de produtos: a maneira como os produtos podem ser anunciados, pedidos e entregues se dá pelo grande potencial de coletar informações, que são utilizadas para customizar os produtos;
5. Tempo para comercializar: permite reduzir o ciclo de vida de produzir e entregar informações e serviços, com base em sua natureza instantânea;
6. Serviço a clientes: por meio de inteligência construída nos sistemas, e da disponibilidade ampliada de sistemas de apoio inteligentes, os sistemas de comércio eletrônico podem melhorar o serviço a clientes;

- Benefícios indiretos, qualitativos:

7. Novas oportunidades de negócio: alterando as estruturas da indústria, os sistemas de comércio eletrônico permitem o surgimento de novos modelos de negócios, baseados na ampla disponibilidade de informações e sua distribuição direta aos clientes finais;
8. Relacionamento com clientes: habilidade em obter informações sobre as necessidades dos clientes e padrões de comportamento, permitindo construir um relacionamento mais personalizado entre fornecedores e seus clientes;
9. Imagem de marca ou corporativa: sistemas de comércio eletrônico podem vir a ser um componente da imagem de marca ou corporativa, especialmente em empresas que comercializam produtos de forte competição;
10. Aprendizagem de tecnologia e laboratório organizacional: é necessário que as empresas se familiarizem rapidamente com o avanço do comércio eletrônico, já que terão a oportunidade de experimentar novos produtos, serviços e processos.

2.4 Estrutura de Análise de Comércio Eletrônico

Segundo ALBERTIN [3], o comércio eletrônico pode ser analisado conforme duas dimensões: aspectos a serem considerados na sua utilização e contribuições que a sua utilização oferece às organizações. Em uma de suas publicações [2] apresentou um conjunto de aspectos e benefícios a serem considerados nos estudos e nas aplicações de comércio eletrônico. Com base nesse trabalho, um novo conjunto de aspectos e benefícios foi definido. A seguir serão apresentados os aspectos levantados na estrutura descrita acima:

2.4.1 Aspectos do Comércio Eletrônico

Os aspectos a serem considerados na utilização de comércio eletrônico, segundo ALBERTIN [3], são os seguintes:

- Adoção: a baixa adoção de sistemas de comércio eletrônico deve-se aos seguintes fatores: resistência, aprendizagem, falta de infra-estrutura e pouca participação das organizações;

- Relacionamento: tendência a melhorar o relacionamento com clientes e fornecedores, através dos seguintes motivos: eliminação de intermediários, diminuição da interação direta, integração eletrônica disponível e facilmente acessível, etc;

- Adequação: oferece novas maneiras de se realizar os processos de negócios, ou seja, novas formas de divulgação, venda e distribuição;

- Estratégia: oferece uma grande quantidade de novas oportunidades para as organizações, exigindo a adequação na definição de suas estratégias de negócios;

- Comprometimento organizacional: necessidade de investimentos para a criação e manutenção dos sistemas, administração de mudanças organizacionais, gerenciamento de riscos, etc;

- Privacidade e segurança: preocupação dos clientes com privacidade e segurança;

- Sistemas eletrônicos de pagamento: trocas financeiras entre compradores e vendedores devem ocorrer em ambiente simples, universalmente aceito, seguro e barato. Os tipos de sistemas eletrônicos de pagamento são: dinheiro eletrônico (*e-cash*), cheque eletrônico (*e-check*), cartões inteligentes (*smart cards*), cartões de crédito e débito;

- Aspectos legais: aspectos legais como por exemplo, a regulamentação de um setor, podem restringir a aplicação de comércio eletrônico, seja pela obrigatoriedade de documentos e presença física, seja pela não-consideração das novas formas de negócios e processos;

- Aspectos de implementação: necessita que a tecnologia esteja alinhada com a organização e adequada à tecnologia externa.

2.4.2 Contribuições do Comércio Eletrônico

As contribuições que o comércio eletrônico proporciona por meio da sua utilização, segundo ALBERTIN [3], são as seguintes:

- Relacionamento: proporciona melhora na comunicação de clientes e fornecedores, mais eficiência nas relações de vendas com clientes e fornecedores, mais atratividade no mercado, melhora na coleta de informações sobre o ambiente externo à organização, estabelecimento de parcerias baseadas em meios eletrônicos entre clientes e fornecedores e compartilhamento de plataformas e mercados eletrônicos com concorrentes;

- Inovação e customização de produtos: flexibilidade e poder de resposta às mudanças das necessidades e particularidades dos clientes;

- Novos canais de venda/distribuição: representam um novo canal de vendas e distribuição para os produtos, existentes ou novos, devido ao alcance direto e à bidirecionalidade na comunicação de informações;

- Promoção de produtos: proporciona melhora na promoção de produtos e serviços, permitindo ao cliente navegar nos produtos e serviços, através de um contato direto, informativo e interativo com os clientes;

- Novas oportunidades de negócio: permite o surgimento de novos modelos de negócios, baseados na ampla disponibilidade de informações e sua distribuição direta aos clientes e fornecedores;

- Estratégia competitiva: pode proporcionar vantagens de custos, permitir a diferenciação de produtos e serviços, possibilitar melhor relacionamento com clientes, permitir entrada mais fácil em alguns mercados, possibilitar o estabelecimento de barreiras de entrada, auxiliar a introdução de produtos substitutos, tornar mais fácil a eliminação de intermediários, facilitar o surgimento de novos intermediários que adicionem valor por meio de informação, e permitir novas estratégias competitivas com o uso de sua tecnologia;

- Economia direta: reduz custos de comercialização, distribuição e serviços a clientes, utilizando infra-estrutura digital pública e compartilhada, como a Internet, comparado a uma infra-estrutura própria;

- Infra-estrutura pública: a criação e utilização de uma infra-estrutura pública é a forma de garantir o fácil e livre acesso a um custo não proibitivo ao ambiente digital do comércio eletrônico.

2.5 Fatores que Estimulam o Crescimento do Comércio Eletrônico

BOLIN [15] apresenta alguns fatores que contribuem para o crescimento mundial do comércio eletrônico. Estes fatores são detalhados a seguir:

- Baixo custo para aquisição de produtos: com a automatização e a redução do processo de compra através das transações eletrônicas, as empresas ganham tempo e dinheiro adquirindo produtos diretamente dos *Web sites* do fornecedor;

- Redução e inventários mais exatos: a Internet permite que as empresas automatizem seus processos obtendo assim um melhor controle sobre seus inventários, fornecendo dados reais e exatos aos parceiros, fornecedores, e compradores, permitindo a comunicação eletrônica entre todas as pessoas envolvidas no processo;

- Baixo período de tempo entre transmissão e recebimento de informações: a comunicação entre fornecedores e clientes é acelerada pela facilidade da troca de informações;

- Aperfeiçoamento dos serviços de clientes: disponibilidade de informações da situação do pedido de compra, informação sobre os produtos, suporte técnico e *downloads* de *softwares on-line*;

- Diminuição de vendedores e custos de marketing: eliminação das forças de vendas tradicionais e oferecimento de informações sobre as preferências e perfis de clientes através das novas tecnologias oferecidas;

- Novas oportunidades de vendas: possibilidade de vender produtos e serviços para novos mercados e novas localizações.

2.6 Limitações do Comércio Eletrônico

A Internet e o comércio eletrônico têm beneficiado em grande escala as organizações e seus clientes, por exemplo: na realização de transações comerciais, na aproximação de compradores e vendedores, na diminuição de custos, na personalização de serviços e na automação de tarefas, mas enfrentam também desafios que podem limitar o crescimento. Executivos Europeus citam exemplos como: segurança, privacidade, e a falta de uma estrutura regulamentar comercial como as principais barreiras do comércio eletrônico [15].

2.7 Loja Virtual

Com o uso freqüente da Internet pelos comerciantes, a tendência é que ocorra um grande aumento na comercialização *on-line*, e também a busca por empresas de desenvolvimento de lojas virtuais. A loja virtual oferece a opção de comércio eletrônico dirigida a pequenos, médios e grandes negócios [61].

Segundo YESIL [61], uma loja virtual é "um lugar onde os clientes podem fazer compras utilizando seus computadores (domésticos e/ou de trabalho) e onde os comerciantes podem oferecer mercadorias e serviços por uma fração[3] do investimento necessário em uma vitrine física[4]".

Uma das principais tarefas na implementação de uma loja virtual é a divulgação, para que as pessoas venham visitá-la, e dessa forma conhecer e comprar os produtos oferecidos. O modelo de loja virtual de venda de produtos pela Internet apresenta-se como uma alternativa complementar à estrutura comercial vigente. Os estabelecimentos comerciais podem, agora, em virtude das novas tecnologias, optar pela venda via Internet, atendendo um grande número de clientes da forma mais cômoda e interativa possível [61].

2.7.1 Tipos de Lojas Virtuais

[3] Fração refere-se a uma parte sobre o valor do investimento em relação a uma vitrine física, que comparado com a virtual é bem maior.

[4] Esta palavra significa o espaço onde os produtos ficam disponíveis para amostra.

Segundo YESIL [61], existe uma segmentação das lojas virtuais por categoria industrial: bens de consumo duráveis, bens de consumo voláteis (informações) e serviços. A figura 2 apresenta a segmentação descrita acima.

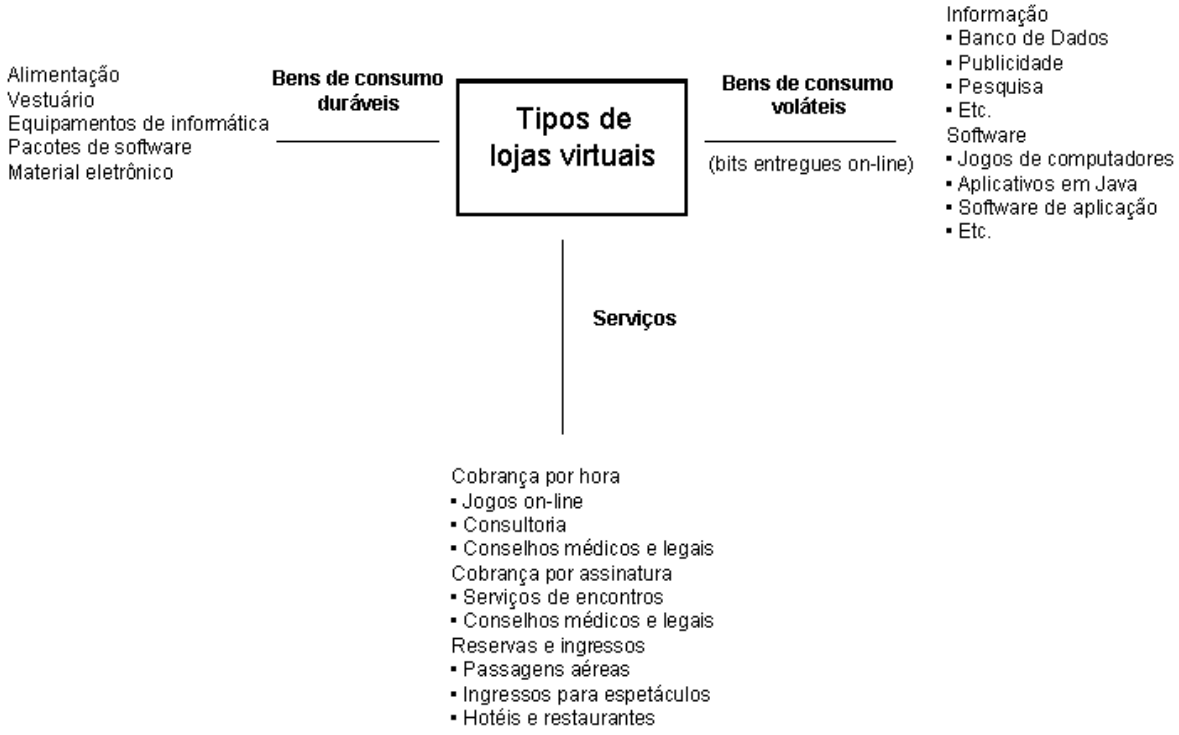


Figura 2 - Diferentes tipos de lojas virtuais. Adaptado de [61], p.13

- Comercialização de bens de consumo duráveis: a venda *on-line* de bens de consumo duráveis é semelhante à operação de vendas por catálogo. A diferença é que o recebimento de encomendas e pagamentos é efetuado pela Internet. Deve-se oferecer aos clientes a mesma sensação de segurança e também a garantia de devolução do dinheiro, que caracterizam uma empresa de vendas por catálogo de boa reputação [61].

A tabela a seguir contém uma lista das lojas virtuais que comercializam bens de consumo duráveis.

Nome	Endereço *on-line*	Tipo de bem de consumo
Amazon	<http://www.amazon.com>	Diversos
Submarino	<http://www.submarino.com.br>	Diversos
Book Stacks Unlimited	<http://www.books.com>	Livros
Flores *on-line*	<http://www.floresonline.com.br>	Flores
Taco	<http://www.taco.com.br>	Roupas

Tabela 1 - Bens de consumo duráveis. Adaptado de [61], p.17

- Comercialização de bens de consumo voláteis: os bens com base na informação são naturalmente propícios para a venda no ambiente da Internet, pois toda a transação pode ser completada dentro da mídia eletrônica. Utilizando-se exclusivamente o computador, os bens podem ser encomendados, o pagamento pode ser recebido e os bens adquiridos podem ser entregues. Tipicamente, os bens com base na informação consistem de algum tipo de material publicado, tais como: artigos de jornal, ou programas de computadores [61].

A tabela a seguir contém uma lista de lojas virtuais que comercializam bens com base na informação.

Nome	Endereço *on-line*	Tipo de serviço
Microsoft	<http://www.microsoft.com/products>	*Softwares* aplicativos da Microsoft
Pathfinder	<http://www.pathfinder.com>	Publicações *on-line*

Tabela 2 - Bens de consumo voláteis. Adaptado de [61], p.21

- Comercialização de serviços *on-line*: a comercialização de serviços *on-line* parece ser uma área onde imaginação e inovação atrai um grande número de clientes. Essencialmente, qualquer serviço comercializado no campo físico também pode estar na Internet, por vezes exibindo uma nova e diferente aparência [61].

A tabela a seguir relaciona uma série de lojas virtuais que prestam serviços.

Nome	Endereço *on-line*	Tipo de serviço
Fedex	<http://www.fedex.com>	Transportadora de volumes para o mundo todo
Tam	<http://www.tam.com.br/b2c/jsp/default.jhtml>	Reservas de passagens aéreas
McEntrega	<http://www.mcdonalds.com.br/>	Entrega de McDonald a domicílio
Localiza	<http://www.localiza.com>	Reservas para aluguel de carro
Match.Com	<http://www.match.com>	Serviços de encontros

Tabela 3 - Serviços *on-line*. Adaptado de [61], p.22

2.7.2 A Utilização da Loja Virtual

Naturalmente, há muito mais na Internet do que *e-mails* e *Electronic Data Interchange* (EDI)[5]. Muitas empresas implementam *sites* na Internet, com intenção de divulgar produtos para o mundo e fornecer informações aos clientes na forma de catálogos *on-line*, mas também existem outras que objetivam a venda direta de produtos e serviços *on-line* [61].

[5] Este termo é definido como sendo à transferência eletrônica de dados comercialmente estruturados utilizando um formato eletrônico padrão.

Para YESIL [61], a decisão sobre qual tipo de *site on-line* é correta para o negócio da empresa deverá levar em consideração alguns fatores, que serão apresentados a seguir, pois nem toda empresa é capaz de gerar novos rendimentos através da venda direta *on-line*. Mas um *site on-line* também pode ser eficaz na redução de custos – por exemplo, como um veículo[6] de marketing e atendimento ao consumidor, mais barato e mais eficiente. Algumas empresas criam lojas virtuais tanto para geração de novas vendas quanto para a redução de custos de marketing, de comercialização de produtos e de atendimento ao consumidor; outras empresas preferem se concentrar fundamentalmente na redução de custos. Outro aspecto que deve ser observado é que algumas empresas criam lojas virtuais simplesmente pelo prestígio proporcionado por uma vitrine *on-line*.

YESIL [61], ressalta que antes de investir em uma loja virtual, é preciso identificar claramente os objetivos que se espera alcançar estabelecendo uma loja *on-line*. Sejam quais forem os objetivos – criar uma nova fonte de renda, cortar despesas de marketing ou promover uma imagem sofisticada – o segredo é certificar-se de que eles são claros, e as expectativas, realistas.

Segundo YESIL [61], os fatores que devem ser analisados, a fim de determinar a conveniência de uma loja virtual são:

- Custos de marketing (pré-venda), tais como: propaganda, promoções, mala direta, distribuição de folhetos, pesquisa de mercado e processo de qualificação, correspondem a um grande percentual de custos gerais da empresa? Se assim for, pode-se utilizar a loja virtual como um redutor de custos para o negócio da empresa;

- Custos pós-venda, tais como: atendimento ao consumidor, suporte técnico, atualização de produtos, etc. – correspondem a um grande percentual de custos gerais da empresa? Em caso afirmativo, pode-se utilizar a loja virtual como um redutor de custos para o negócio da empresa;

- Produtos ou serviços que só podem ser entregues exclusivamente pela Internet, tal como uma pesquisa atualizada ou a venda de acesso à Internet, podem gerar uma loja virtual que se torne uma nova e significativa fonte de renda;

- Produtos ou serviços que só podem ser entregues pela Internet, tal como *software*, informação, artigos, fotografias, jogos para computadores ou jogos *on-line*, que são cobrados por hora, podem gerar uma loja virtual que retorne uma significativa parcela do investimento;

- Se os clientes costumam adquirir produtos, tais como: material para escritório e artigos para o lar repetidamente, a loja virtual pode gerar novos recursos utilizando o serviço de assinaturas;

- Se a empresa comercializa produtos que são adquiridos pelos clientes sazonalmente, em intervalos previsíveis, tais como: presentes de aniversário, artigos domésticos ou gêneros alimentícios;

[6] Veículo no sentido de meio de difusão, transmissão e propagação de serviços de marketing, tais como propaganda, promoções.

- Se a empresa comercializa produtos que poupam o tempo das pessoas se forem adquiridos *on-line*, tais como: presentes que precisam ser embalados e despachados para uma transportadora;

- Se a empresa comercializa produtos passíveis de serem associados a uma atividade que pode ser mais bem executada *on-line* – por exemplo, produtos que exijam uma pesquisa que pode apresentar melhores resultados se for realizada *on-line*, e não por encomenda – embora a entrega do produto adquirido não seja pela Internet;

- Se o negócio da empresa se caracteriza pela venda de produtos de qualidade a preços comprovadamente baixos, a loja virtual pode gerar muitas vendas;

- Se a empresa explora um nicho de mercado[7], com seguidores fiéis, que já esteja na Internet, cria-se a possibilidade de vender produtos de nicho de maneira eficaz.

2.7.3 Elementos que Compõem o Sucesso de uma Loja Virtual

Conforme estudo realizado [61], os cinco principais elementos para o sucesso de uma loja virtual são:

1. Fixar as metas corretas para a loja virtual;
2. Compreender o processo de vendas ao consumidor e criar um ambiente apropriado ao oferecimento dos produtos e serviços *on-line*;
3. Aprender a conhecer os clientes e, desse modo, gerar negócios repetidos e adaptar a loja virtual de acordo com o retorno da parte deles;
4. Divulgar e promover a loja virtual nos locais certos, o que leva ao crescimento das vendas;
5. Receber pagamentos na própria loja virtual.

2.7.4 Funções que uma Loja Virtual deve Desempenhar

O sucesso no mundo virtual não depende do tamanho da empresa, segundo YESIL [61], e, sim, ter uma loja virtual completa e funcional que ofereça produtos e serviços de qualidade.

YESIL [61], enumerou quatro tipos de funções que uma loja virtual completa deve ser capaz de desempenhar. Sugere, ainda, que toda loja virtual deve ser capaz de cumprir, ao menos, as duas primeiras, caso contrário não será de fato uma loja. Conforme o tipo de loja que se esteja implementando, as duas últimas funções poderão ser ou não aplicáveis ao caso. As funções são apresentadas a seguir:

1. Exibir produtos e serviços: toda loja virtual deve ser projetada de modo a realçar os produtos e serviços disponíveis;

[7] "Nichos de Mercado são pequenos segmentos que oferecem oportunidades de negócios incrementais. [60]".

2. Processar adequadamente os pedidos e as indagações dos clientes: uma vez que a Internet consiste num ambiente orientado à informação, é obrigatório que a loja virtual esteja preparada para responder às perguntas e para fornecer informações adicionais aos clientes em perspectiva;
3. Processar adequadamente as transações relativas aos pagamentos: implementar alguma forma de receber pagamentos – cartões de crédito, cheques e dinheiro (na forma digital ou não) – diretamente *on-line*, ou através do uso de uma outra mídia, tal como: telefone ou fax. O maior dentre os desafios a serem vencidos ao se receber pagamentos *on-line* é a questão da segurança – seja em relação ao investimento com a segurança, seja em relação à divulgação do benefício entre os clientes. Faz-se necessário o uso de navegadores (*browsers*) que possam oferecer a segurança necessária à transferência de dados dos cartões de crédito e outras informações delicadas pela Internet e uma conta em um banco comercial;
4. Entregar os produtos e serviços através da própria Internet: para bens baseados em informação, a distribuição *on-line* é geralmente a opção menos cara. Faz sentido transferir através da Internet, vídeos e capítulos de um livro que estejam sendo vendidos. Para uma loja virtual que vende bens duráveis a entrega *on-line* não faz sentido. Contudo, pode-se enviar um *e-mail* solicitando a confirmação do pedido.

No próximo capítulo serão apresentadas as tecnologias de Padrões de Projeto e *Frameworks*, bem como sua contribuição para o desenvolvimento de sistemas de comércio eletrônico.

3 PADRÕES DE PROJETO E *FRAMEWORKS*

O reuso de *software* é essencial para aumentar o retorno de seus custos de desenvolvimento. Os requisitos relativos a menores custos de produção e manutenção de *software*, à maior rapidez na entrega de sistemas e ao aumento da qualidade só podem ser atendidos pelo reuso generalizado e sistemático de *software* [58]. Um outro fator que surgiu com o advento da *Web* diz respeito a uma nova plataforma, que introduz uma outra forma de pensar em como disponibilizar informações rápidas e seguras. Assim, torna-se necessário repensar em como desenvolver sistemas de *software* que atendam diferentes necessidades. Diversas metodologias são projetadas de forma a atender essas necessidades; porém, alguns aspectos que devem ser considerados chamam a atenção: que tecnologia será utilizada para desenvolver o *software* e que possíveis rotinas, bibliotecas, componentes, funções e serviços já desenvolvidos podem ser reutilizados.

De acordo com o estudo realizado, alguns autores ressaltam a importância da utilização de *frameworks* orientados a objetos e padrões de projeto na construção de *softwares*, por permitirem uma maior reutilização de código [17] [22] [28] [35].

Nas seções seguintes serão apresentados os conceitos de padrões de projeto e *frameworks* que vêm se tornando cada vez mais comuns e importantes para o desenvolvimento de sistemas de *software*.

3.1 Padrões de Projeto

Pode-se pensar em padrões de projeto, como uma reutilização da essência de uma solução para determinados problemas similares. Segundo Christopher Alexander [4]: "cada padrão descreve um problema no nosso ambiente e o núcleo da sua solução, de tal forma que você possa usar esta solução mais de um milhão de vezes, sem nunca fazê-lo da mesma maneira". O padrão de projeto identifica as classes e instâncias participantes, seus papéis, colaborações e a distribuição de responsabilidades. Cada padrão de projeto focaliza um problema ou tópico particular de projeto orientado a objetos [28]. Os padrões de projeto ajudam a escolher alternativas de projeto que tornam um sistema reutilizável e a evitar alternativas que comprometam a reutilização [28].

Em geral, um padrão de projeto tem quatro elementos essenciais [28]:

- Nome do padrão;
- Problema – descreve quando aplicar o padrão, podendo ser a descrição de problemas de projeto específicos, ou de estruturas de classes e objetos, ou ainda uma lista de condições que devem ser satisfeitas para que faça sentido aplicá-lo;
- Solução – descrição abstrata de um problema de projeto e de como um arranjo geral de elementos (classes e objetos) resolve o mesmo;
- Conseqüências – resultados e análises das vantagens e desvantagens da aplicação do padrão.

3.1.1 Descrição de Padrões de Projeto

Para descrever padrões de projeto, Erich Gamma, Richard Helm, Ralph Johnson e John Vlissides, conhecidos como *The Gang of Four* (GOF), definiram um gabarito que fornece uma estrutura uniforme às informações, tornando os padrões de projeto mais fáceis de aprender, comparar e usar [28]. A estrutura é apresentada a seguir:

- **Nome e classificação do padrão** – um bom nome, pois se tornará parte do vocabulário do projeto, e a categoria na qual o padrão é classificado (ver seção 3.1.2);

- **Intenção e objetivo** – curta declaração sobre o que faz o padrão, quais seus princípios e sua intenção e que tópico ou problema particular de projeto ele trata;

- **Também conhecido como** – outros nomes conhecidos para o padrão, se existirem;

- **Motivação** – um cenário que ilustra um problema de projeto e como as estruturas de classes e objetos no padrão solucionam o problema, para ajudar a compreender as descrições mais abstratas;

- **Aplicabilidade** – quais as situações nas quais o padrão de projeto pode ser aplicado, que exemplos de mau projeto ele pode tratar e como essas situações podem ser reconhecidas;

- **Estrutura** – representação gráfica das classes do padrão baseada em *Unified Modeling Language* (UML) [14] [29] e diagramas de interação para ilustrar seqüências de solicitações e colaborações entre objetos;

- **Participantes** – classes e/ou objetos que participam do padrão e suas responsabilidades;

- **Colaborações** – como os participantes colaboram para executar suas responsabilidades;

- **Conseqüências** – como o padrão suporta a realização de seus objetivos, quais seus custos e benefícios e o resultado de sua utilização, e que aspecto da estrutura do sistema ele permite variar independentemente;

- **Implementação** – considerações específicas de linguagem, se existirem, e quais armadilhas, sugestões ou técnicas é preciso conhecer quando da implementação do padrão;

- **Exemplo de código** – trechos de código que ilustram como o padrão pode ser implementado;

- **Usos conhecidos** – exemplos do padrão encontrados em sistemas reais;

- **Padrões relacionados** – padrões que estão intimamente relacionados com este, as diferenças importantes, e com quais outros padrões este deveria ser utilizado.

Como exemplo da utilização deste gabarito, a seguir são apresentados alguns itens da descrição do padrão de projeto *Factory Method* [28] (Tabela 4).

Factory Method	Criação de classes

Intenção

Definir uma interface para criar um objeto, mas deixar as subclasses decidirem que classe instanciar. O *Factory Method* permite que uma classe delegue a responsabilidade de instanciação às subclasses [28].

Também conhecido como

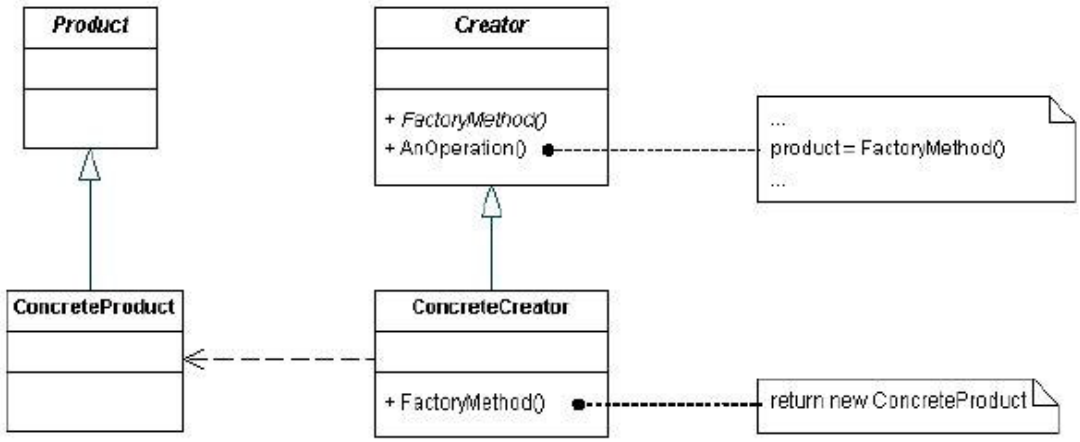
Virtual Constructor

Aplicabilidade

Usar o padrão *Factory Method* quando:

- Uma classe não pode antecipar a classe de objetos que deve criar;
- Uma classe quer que suas subclasses especifiquem os objetos que criam;
- Classes delegam responsabilidade para uma dentre várias subclasses auxiliares, e deseja-se localizar o conhecimento de qual subclasse auxiliar que é a delegada.

Estrutura



Participantes

- *Product*
  - Define a interface dos objetos que o *factory method* cria.
- *ConcreteProduct*
  - Implementa a interface do *Product*.
- *Creator*
  - Declara o *factory method*, o qual retorna um objeto do tipo *Product*. Também pode definir uma implementação por omissão do *factory method* que retorna por omissão um objeto *ConcreteProduct*.

- Pode chamar o *factory method* para criar um objeto *Product*. • *ConcreteCreator* - Redefine o *factory method* para retornar a uma instância de um *ConcreteProduct*. **Padrões relacionados** *Abstract Factory* é freqüentemente implementado utilizando o padrão *Factory Method*. *Factory Method* são usualmente chamados dentro de *Template Methods*. *Prototypes* não exigem subclassificação de *Creator*. Contudo, freqüentemente necessitam uma operação *Initialize* na classe *Product*. A *Creator* usa *Initialize* para inicializar o objeto. O *Factory Method* não exige uma operação deste tipo.

Tabela 4 - Exemplo parcial de descrição do padrão de projeto *factory method*. Adaptado de [28], p.112

3.1.2 Classificação de Padrões de Projeto

A classificação de padrões de projeto ajuda a aprender os padrões mais rapidamente, referenciando a família de padrões relacionados [28]. Eles podem ser classificados segundo as seguintes categorias:

- Padrões **de criação** – abstraem o processo de instanciação. Um padrão de criação de classe deixa alguma parte da criação de objetos para subclasses, enquanto que um padrão de criação de objetos delegará a instanciação para outro objeto;

- Padrões **estruturais** – lidam com a composição de classes ou de objetos. Os padrões estruturais voltados para classe utilizam herança para compor classes, enquanto que um padrão de estrutura de objetos descreve maneiras de montar objetos;

- Padrões **comportamentais** – caracterizam as maneiras pelas quais classes ou objetos interagem e distribuem responsabilidades. Padrões comportamentais de classe utilizam herança para descrever algoritmos e fluxo de controle, enquanto que um padrão comportamental de objetos descreve como um grupo de objetos coopera para executar uma tarefa que um único objeto não pode executar sozinho.

3.1.3 Catálogo de Padrões de Projeto

Padrões de projeto variam na sua granularidade e no seu nível de abstração. Segundo GAMMA et. al [28], os padrões de projeto são classificados de maneira que se pode referir à família de padrões relacionados. A classificação baseia-se em dois critérios: o primeiro, chamado **finalidade**, reflete o que o padrão faz; o segundo, chamado **escopo**, especifica se o padrão se aplica primariamente a classes ou a objetos. A tabela 5 apresenta essa classificação:

		Propósito		
		Criação	**Estrutural**	**Comportamental**
Escopo	Classe	*Factory Method*	*Adapter (class)*	*Interpreter* *Template Method*
	Objeto	*Abstract Factory* *Buider* *Prototype* *Singleton*	*Adapter (object)* *Bridge* *Composite* *Decorator* *Façade* *Flyweight* *Proxy*	*Chain of Responsibility* *Command* *Iterator* *Mediator* *Memento* *Observer* *State* *Strategy* *Visitor*

Tabela 5 - Organização dos padrões de projeto segundo GOF

3.1.4 Utilização de Padrões de Projeto

Programas orientados a objetos são feitos de objetos. Um objeto empacota tanto os dados quanto os procedimentos que operam sobre esses dados. Para que um objeto execute uma operação ele deve receber uma solicitação, e para que seus dados internos sejam alterados devem ser executadas operações [28].

A partir disso, num projeto orientado a objetos, a decomposição de um sistema em objetos é a tarefa mais difícil: na fase de análise nem todos os objetos do sistema são definidos. Alguns fatores contribuem para dificultar esse processo: encapsulamento, granularidade, dependência, flexibilidade, desempenho, evolução e reutilização, entre outros [28].

Padrões de projeto ajudam a identificar abstrações menos óbvias bem como os objetos que podem capturá-las. Por exemplo: como decidir o que deve ser um objeto? O padrão *Façade* descreve como representar subsistemas completos como objetos, e o padrão *Flyweight* descreve como suportar enormes quantidades de objetos nos níveis de granularidade mais finos. Outros padrões descrevem maneiras específicas de decompor um objeto em objetos menores, como o *Builder*, o *Abstract Factory* e o *Command* [28].

Padrões também ajudam a definir interfaces pela identificação de seus elementos-chave e pelos tipos de dados que são enviados através de uma interface. Um padrão também pode dizer o que não colocar numa interface. O padrão *Memento* é um bom exemplo. Ele descreve como encapsular e salvar o estado interno de um objeto, de modo que o objeto possa ser restaurado àquele estado mais tarde. Os padrões também especificam relacionamentos entre interfaces. Isso é importante, pois as interfaces são fundamentais em sistemas orientados a objetos [28].

Padrões de criação como *Abstract Factory*, *Builder*, *Factory Method*, *Prototype* e *Singleton* permitem abstrair o processo de criação de objetos, propiciando várias maneiras de associar uma interface com sua implementação de forma transparente no momento da instanciação. Isto reduz as dependências de implementação entre subsistemas favorecendo a reutilização [28].

Para que se possa ter maior reutilização, é importante que o sistema possa evoluir de acordo com as mudanças nos requisitos existentes. Um padrão permite que o sistema possa mudar segundo maneiras específicas, com a variação de algum aspecto da estrutura do sistema, independentemente de outros aspectos [28].

Mas como decidir pelo melhor padrão para tratar certo problema? Diversas abordagens são sugeridas segundo GAMMA et. al [28], e apresentadas a seguir:

- Considerar como padrões solucionam problemas de projeto;
- Examinar o item **Intenção** da descrição de cada padrão, podendo focalizar a busca utilizando sua classificação, a fim de encontrar um ou mais padrões que pareçam relevantes para o problema em questão;
- Estudar como os padrões se inter-relacionam;
- Estudar padrões de finalidades semelhantes;
- Examinar causas que possam resultar na reformulação do projeto;
- Considerar o que deveria ser variável no projeto, o que poderia ser mudado sem precisar reformular o projeto.

Outra questão é: Como utilizar o Padrão de Projeto escolhido? GAMMA et. al [28], sugere uma abordagem passo a passo:

- Ler o padrão por inteiro uma vez, para obter sua visão geral;
- Voltar e estudar as seções **Estrutura**, **Participantes** e **Colaborações**, para compreender as classes e objetos e como se relacionam entre si;
- Olhar a seção **Exemplo de Código**, para ver um exemplo concreto de código do padrão;
- Escolher nomes para os participantes do padrão que tenham sentido no contexto da aplicação;
- Definir as classes;
- Definir nomes específicos da aplicação para as operações no padrão;
- Implementar as operações para suportar as responsabilidades e colaborações presentes no padrão.

Padrões de Projeto não devem ser usados indiscriminadamente. Por oferecerem flexibilidade e variabilidade, se não aplicados para esse fim, podem ocorrer perdas em relação ao custo/benefício de sua utilização. A seção **Conseqüências** na descrição de padrões são muito úteis na sua avaliação [28].

3.1.5 Padrão de Projeto Camada de Persistência

O padrão de projeto Camada de Persistência (*Persistence Layer*) foi proposto por YODER et. al [62] para minimizar a incompatibilidade existente entre o paradigma orientado a objetos e a persistência de dados em um banco relacional. Esse padrão de projeto isola no desenvolvimento de *software* os detalhes de implementação de persistência e protege a aplicação desenvolvida de mudanças que possam ocorrer. Esse padrão consiste de uma camada de persistência que cuida da interface entre objetos de aplicação e tabelas de banco de dados relacional.

Leitura e escrita de objetos para uma base de dados requer basicamente operação de *create*, *read*, *update*, e *delete*. Assim cada objeto pode ter uma interface para acessar o banco de dados, tornando fácil o uso e a manutenção se prover um conjunto de operações para a camada de persistência que todos os objetos possam utilizar.

YODER et. al [62] definem um conjunto de sub-padrões que podem ser utilizados para a implementação dessa camada de persistência que será apresentado a seguir:

- Camada de persistência (*Persistence Layer*): provê uma camada para mapear os objetos em um banco de dados relacional ou outro banco de dados;

- CRUD (Criação (*Create*), Leitura (*Read*), Atualização (*Update*) e Remoção (*Delete*)): operações mínimas necessárias para a persistência de objetos, que são: criação, leitura, atualização e remoção;

- Descrição de código SQL (*SQL Code Description*): define a geração de cláusulas SQL para cada objeto persistente da aplicação. Essas cláusulas são utilizadas na implementação das operações do padrão CRUD;

- Método de mapeamento de atributos (*Attribute Mapping Methods*): efetua o mapeamento de valores entre os atributos de um objeto e as respectivas colunas da tabela (ou tabelas) do banco de dados;

- Conversão de tipos (*Type Conversion*): trabalha em conjunto com métodos de mapeamento de atributos para converter os valores do banco de dados para o tipo apropriado de objetos e vice-versa;

- Gerenciador de mudanças (*Change Manager*): mantém o controle de quando os objetos sofreram modificações, para que o sistema possa determinar quais objetos devem ser persistidos ou não. Isso determina a necessidade de escrever o valor para uma tabela do banco de dados ou não;

- Gerenciador de identificadores únicos (*OID Manager*): gera automaticamente um identificador único para o objeto durante uma inclusão;

- Gerenciador de transação (*Transaction Manager*): mecanismo que permite o controle de transações com o banco de dados enquanto existir um objeto sendo persistido;

- Gerenciador de conexão (*Connection Manager*): efetua a conexão do sistema com a base de dados e garante a sua manutenção;

- Gerenciador de tabelas (*Table Manager*): gerencia o mapeamento de um objeto para sua respectiva tabela (ou tabelas) e coluna (ou colunas) no banco de dados.

Todos esses padrões descritos colaboram juntos para prover um mecanismo para mapear objetos persistentes para um banco de dados. A seguir é apresentada na figura 3, a junção de todos os padrões.

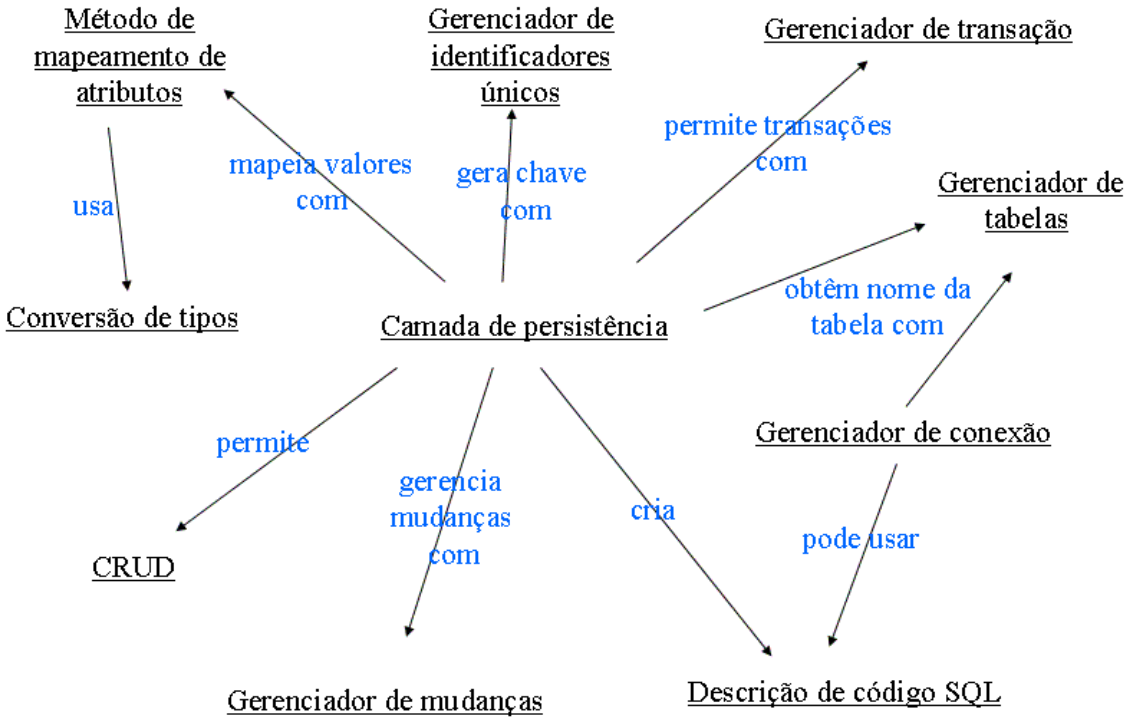


Figura 3 - Diagrama de interação de padrões. Adaptado de [62]

A figura 3 apresenta como os padrões interagem uns com os outros. A camada de persistência provê uma interface padrão para operações CRUD necessárias para persistir o objeto de domínio. A camada de persistência constrói as chamadas para o banco de dados utilizando descrição de código SQL fornecido para o objeto de domínio. Durante a geração do código SQL, a camada de persistência interage com o gerenciador de tabelas para obter o nome correto da tabela e das colunas no banco de dados. Quando os valores estão sendo retornados do banco de dados ou escritos no banco de dados, os valores dos atributos devem ser mapeados para o nome da coluna no banco de

dados e vice-versa. Isso é feito com método de mapeamento de atributos. Métodos de mapeamento de atributos fazem alguma conversão de tipos, enquanto o código SQL está sendo gerado. Métodos de mapeamento de atributos e conversão de tipos também podem ajudar, enquanto a camada de persistência instancia um novo objeto. A camada de persistência salva o objeto no banco de dados através do gerenciador de conexão somente quando algum valor do objeto tem sido alterado. Esta alteração é gerenciada pelo gerenciador de mudança. O gerenciador de conexão pode interagir com o gerenciador de tabela para decidir qual base de dados utilizar. A camada de persistência provê acesso para o gerenciador de transação sempre que o processamento de uma transação se faz necessário [62].

3.2 *Frameworks*

O reuso de *software* tem sido um dos principais objetivos da Engenharia de *Software* por décadas [22]. Reutilizar *software* não é simples, e muitos esforços resultam em pequenas reutilizações, através de componentes caixa-preta, onde os recursos existentes são reutilizados e estendidos por meio da definição de um componente adequado a uma interface específica, junto à integração de novos componentes. Com o surgimento do paradigma orientado a objetos continua sendo possível reutilizar componentes, porém torna-se necessária à definição de aplicações de *frameworks* orientados a objetos. *Frameworks* atraem atenção de muitos pesquisadores e engenheiros de *softwares*, e têm sido definidos para uma grande variedade de domínios [22].

Frameworks podem ser definidos como:

- Segundo FIRESMITH [23], framework é "uma coleção de classes colaborativas que capturam ambos padrões em pequena escala e mecanismos maiores que implementam requisitos e projetos em comum num domínio de aplicação específico".

- Segundo JOHNSON e FOOTE [35], *framework* é "um conjunto de classes que constitui um projeto abstrato para soluções de uma família de problemas".

- Para JOHNSON [36], *framework* é "um conjunto de objetos que colaboram com o objetivo de cumprir um conjunto de responsabilidades para uma aplicação ou um domínio de um subsistema".

- Para MATTSSON [48], *framework* é "uma arquitetura desenvolvida com o objetivo de se obter a máxima reutilização, representada como um conjunto de classes abstratas e concretas, com um grande potencial de especialização".

3.2.1 Benefícios da Utilização de *Frameworks*

Os benefícios primários identificados em [17] [22] [36] são:

- Modularidade – f*rameworks* aumentam a modularidade, pois encapsulam detalhes de implementação dentro de interfaces bem definidas e estáveis. A modularidade de *frameworks* auxilia na qualidade de *software*, localizando os lugares de impacto no projeto e nas trocas de implementações, que reduz o esforço necessário para entender e realizar manutenção em *softwares* existentes e futuros;

- Reutilização – interfaces estáveis presentes nos *frameworks* aumentam a reutilização pela definição de componentes abstratos, que podem ser redefinidos para criarem novas aplicações. A reutilização de *frameworks* eleva o conhecimento do domínio e o esforço prévio de desenvolvedores experientes, a fim de evitar a recriação e revalidação de soluções comuns para repetidos requisitos de aplicações;

- Extensibilidade – um *framework* aumenta a extensibilidade na medida em que oferece métodos *hook* explícitos [54]. Estes permitem que aplicações possam ser estendidas pelas interfaces estáveis existentes. Métodos *hook* sistematicamente desacoplam interfaces estáveis e comportamentos de um domínio de aplicação das variações requeridas pela instanciação de uma aplicação num contexto particular. Extensibilidade de *frameworks* é essencial para assegurar a oportuna customização de novos serviços e características de aplicações;

3.2.2 Classificação de *Frameworks*

Quanto ao contexto de utilização de *frameworks* há uma classificação proposta em [17] [22] [48], segundo a qual *frameworks* são classificados em: *frameworks* de aplicação, que se refere ao domínio de um problema, como por exemplo: desenvolvimento de interface de usuário; *frameworks* de estrutura, relacionados à arquitetura; e *frameworks* de uso específicos, que solucionam problemas em áreas determinadas.

Quanto ao domínio do problema

- *Frameworks* de aplicação. Cobrem funcionalidades que podem ser aplicadas em diferentes domínios. Exemplos de *frameworks* de aplicação são *frameworks* para interfaces gráficas de usuários (GUI) [59].

- *Frameworks* de domínio capturam conhecimento e peculiariedades de um problema específico. *Frameworks* de negócios, tais como: telecomunicações, aviação, finanças, educação, controle de produção e multimídia são exemplos de *frameworks* de domínios [55].

- *Frameworks* de suporte são *frameworks* que oferecem serviços de baixo nível em sistemas como, por exemplo, *drives* para dispositivos e acesso a arquivos.

Quanto à estrutura

Se a estrutura interna do *framework* está descrita, torna-se fácil entender o comportamento do *framework*.

A estrutura interna de um *framework* está relacionada à concepção da arquitetura de *software*. O princípio da estrutura interna de um *framework* orientado a objetos é descrito pela arquitetura do *framework*. BUSCHMANN [16] relaciona algumas das principais arquiteturas de *frameworks*:

- *Framework* com arquitetura em camadas: ajuda a estruturar aplicações que podem ser decompostas em grupos de subtarefas com diferentes níveis de abstrações.

- *Frameworks* com arquitetura *pipes* e *filters*: utilizados para estruturar aplicações que podem ser divididas em muitas subtarefas completamente independentes, que devem ser executadas em uma seqüência determinada ou em paralelo.

- *Framework* com arquitetura MVC (*Model View Controller*): define uma estrutura para aplicações interativas que separam a sua interface com o usuário do seu núcleo de funcionalidade.

- *Framework* com arquitetura PAC (*Presentation Abstration Controller*): o *software* deve possuir muita interação com o usuário, que possibilite controles múltiplos da apresentação dos modelos de abstração que podem ser decompostos em subfunções independentes.

- *Framework* com arquitetura reflexiva: é utilizado em aplicações que necessitam considerar adaptações futuras para mudanças do ambiente, tecnologia e requisitos. Essas adaptações não interferem na estrutura e na implementação.

- *Framework* com arquitetura *Microkernel*: é utilizado em sistemas de *softwares* que promovem diferentes visões de acordo com suas funcionalidades e que têm que adaptar novos requisitos ao sistema, por exemplo: sistemas operacionais.

- *Framework* com arquitetura *Blackboard*: ajuda a estruturar aplicações complexas que contêm diversos subsistemas especializados para diferentes domínios. Esses cooperam entre si para construir uma solução para o problema.

- *Framework* com arquitetura *Broker*: utilizado em sistemas de *softwares* distribuídos, nos quais a interação entre os componentes é feita via chamada remota.

Quanto ao uso

Frameworks podem ser utilizados de duas maneiras, através da derivação de novas classes, ou da combinação de classes já existentes. A seguir são classificados os dois tipos de utilização [22] [47]:

- *White-box* – utilizado através da derivação de suas classes;

- *Black-box* – utilizado por meio da combinação de suas classes;

No tipo *White-box*, a instanciação só é possível através da codificação e criação de novas classes. Estas classes e códigos podem ser introduzidos no *framework* por meio de herança ou composição [47]. Este tipo exige que desenvolvedores tenham conhecimento da estrutura interna do *framework* [22].

Já o tipo *Black-box*, é baseado em componentes de *software*. Os recursos existentes são reutilizados e estendidos por meio da definição de um componente adequado a uma interface específica e integração de novos componentes. São considerados mais fáceis de trabalhar, pois utilizam componentes existentes, sem a necessidade da construção de novos [22].

3.2.3 Arquitetura de *Frameworks*

Um *framework* é uma aplicação genérica que permite que diferentes aplicações sejam criadas de uma família de aplicações. Todas as aplicações de uma família formam um subdomínio de uma aplicação de domínio. Visto que *frameworks* são criados para gerar aplicações para um domínio específico, devem existir pontos flexíveis que são customizados de acordo com a aplicação específica para solução de um problema [17] [22] [47].

Um *framework* pode ser customizado para uma aplicação específica, através da criação de subclasses específicas para a aplicação, sendo essas subclasses específicas das classes abstratas do *framework*.

Os pontos de flexibilidade de um *framework* são chamados de *hot-spots*, assim cada aplicação gerada a partir do *framework* implementa da sua maneira os *hot-spots*, instanciando-se o *framework* [47].

Hot-spots muitas vezes são classes abstratas ou métodos abstratos que não possuem implementação, assim devem ser especializados (customizados) para necessidades específicas da aplicação a ser gerada. Quando um *hot-spots* for uma classe abstrata, a especialização pode ser realizada tanto por herança como por delegação, dependendo de como os *hot-spots* foram planejados [17].

Mas nem todos os aspectos de uma aplicação podem ser criados de modo flexível. Assim, algumas características de um *framework* não serão mutáveis e não poderão ser alteradas. Estes pontos de imutabilidade constituem o *kernel* de um *framework*, também chamado de *frozen-spots*, ou seja, as partes fixas que são compartilhadas por todas as aplicações geradas a partir do *framework* [47].

3.2.4 Processo de Desenvolvimento

Um *framework* pode ser visto como generativo desde que seja intencionado para uso no desenvolvimento de um número de aplicações no domínio de aplicação capturado pelo *framework* [48]. Um *framework* Orientado a Objetos (OO) captura os aspectos comuns de uma família de aplicações e também a experiência do projetista durante o processo de construção da aplicação. Isso possibilita a reutilização tanto em projeto como em programação [56], o que não se consegue num processo de desenvolvimento OO tradicional.

No processo de desenvolvimento tradicional [48] [49] [50], a fase de análise fornece subsídios para a elaboração de um projeto que resultará na implementação de uma única aplicação. Essa aplicação descreve um programa executável completo que satisfaz os requisitos de especificação [48].

O processo de desenvolvimento baseado em *frameworks* [47] [48] [49] [50], consiste de três fases: análise do domínio, projeto do *framework* e instanciação do *framework*.

A fase de **análise do domínio** é uma análise extensiva de um domínio completo, ou seja, um levantamento e definição dos requisitos que serão utilizados no desenvolvimento de um *framework,* que poderá ser instanciado diversas vezes para aplicações diferentes pertencentes ao domínio analisado. Sendo assim, não é necessário um novo trabalho de análise e projeto para construir uma nova aplicação, deve-se customizar os *hot-spots* que constituirão as características para a nova aplicação. Nesta fase, os *hot-spots* e *frozen-spots* começam ser encontrados [47].

A fase de **projeto** do *framework* concentra os esforços de criar *frameworks* de abstração. Nesta fase, os *hot-spots* e *frozen-spots* devem ser modelados, e a extensibilidade e flexibilidade propostas na análise do domínio são delineadas [47].

Finalmente, na fase de **instanciação**, os *hot-spots* do *framework* são implementados e um sistema de *software* é gerado. Cada uma das aplicações geradas terá em comum pelo menos um *framework frozen-spots* [47].

A figura 4 apresenta uma comparação entre as fases do processo de desenvolvimento do *framework* e das fases de desenvolvimento de projeto orientado a objetos tradicionais [47] [48].

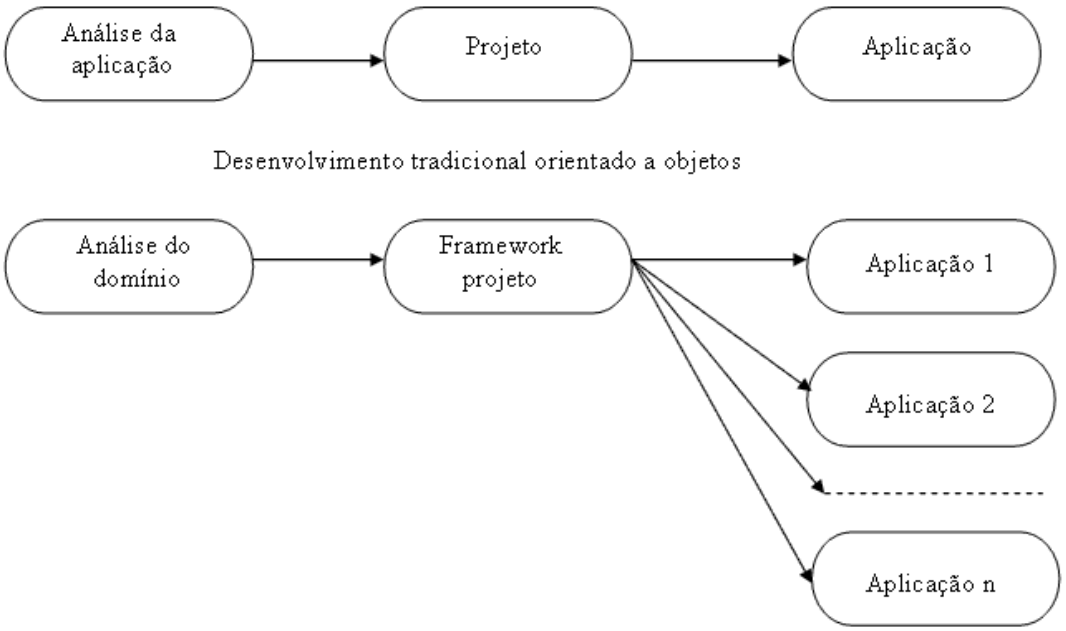


Figura 4 - Processo de desenvolvimento OO versus baseado em *frameworks*. Adaptado de [48]

O processo de desenvolvimento de um *framework* depende do grau de experiência da organização no domínio do problema. Uma organização com mais experiência pode adotar um processo mais avançado no desenvolvimento, desde que possua menos problemas com o problema do domínio. A seguir serão apresentados alguns processos propostos para o desenvolvimento de um *framework* [48].

3.2.4.1 Processo Baseado em Experiência de Aplicações já Desenvolvidas

Este processo de desenvolvimento [48] [49] inicia-se com o desenvolvimento de n aplicações (mínimo de duas) baseadas no domínio do problema. Quando prontas, inicia-se a primeira interação no processo, identificam-se as características em comum das aplicações e extraem-se essas para o

framework. Para verificar se as características extraídas estão corretas, refazem-se as aplicações com base no *framework* desenvolvido. Isso deve ser fácil se as características em comum identificadas estão corretas. No caso de demandar muito esforço, reescreve-se o *framework*, quando necessário, e utiliza-se essa experiência no desenvolvimento da nova versão. Baseando-se na nova versão do *framework*, novas aplicações podem ser desenvolvidas, repetindo-se as iterações quantas vezes forem necessárias. Esse processo de desenvolvimento pode ser visto na figura 5.

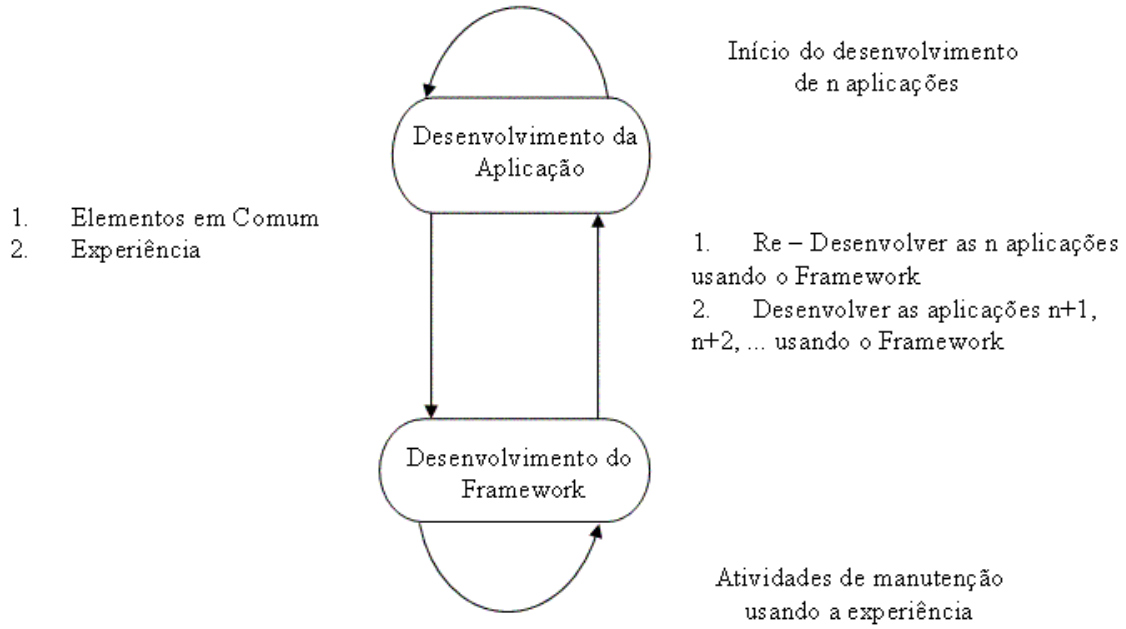


Figura 5 - Desenvolvimento de *frameworks* baseado em experiência. Adaptado de [48]

3.2.4.2 Processo Baseado na Análise do Domínio

A primeira etapa do processo de desenvolvimento [48] [49] é analisar o domínio do problema para identificar e entender possíveis abstrações no domínio. Analisar o domínio requer analisar aplicações existentes (na qual é uma tarefa difícil de ser executada), e somente é possível se a organização já possuir aplicações desenvolvidas. Essa análise poderá tornar-se um grande custo para a organização, e que segundo JOHNSON et. al [35] consiste de três atividades principais:

- Análise do contexto do domínio – identificar o escopo do domínio. Como resultado desta análise, obtém-se o modelo da análise do domínio, contendo os requisitos do domínio, os conceitos do domínio, e os relacionamentos entre esses conceitos;

- Modelagem das abstrações do domínio – identificar os componentes certos do domínio e seus dados, comportamento e interação;

- Arquitetura do domínio – gerar o *backbone* do *framework*, ou seja, o esqueleto do *framework*.

Após identificar as abstrações, desenvolve-se o *framework* juntamente com uma aplicação de teste, modificando o *framework* quando necessário, revisando as aplicações anteriores para verificar se continuam funcionando. A figura 6 apresenta esse processo de desenvolvimento.

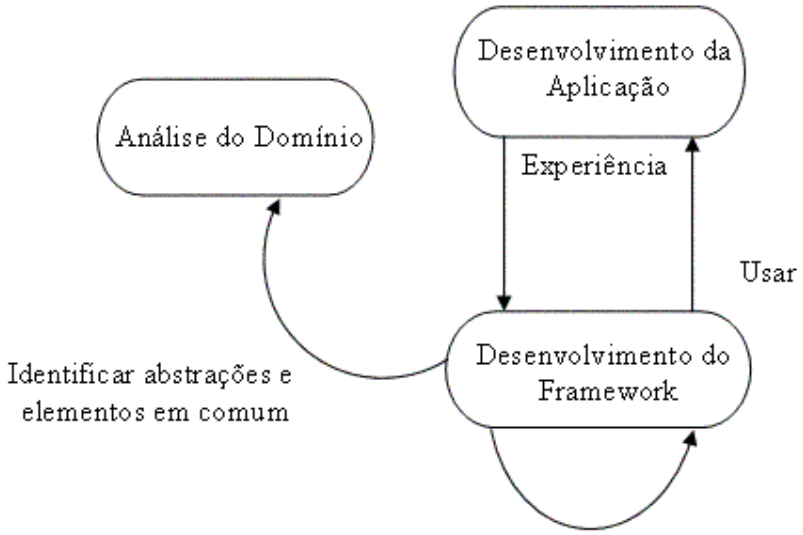


Figura 6 - Desenvolvimento de *frameworks* baseado na análise de domínio. Adaptado de [48]

3.2.4.3 Processo Utilizando Padrões de Projeto

Neste processo [48] [49], o primeiro passo consiste em desenvolver uma aplicação e, em seguida, aplicar sistematicamente um conjunto de padrões de projeto para criar o *framework*. A partir desse momento iniciam-se as interações entre as aplicações e o *framework*. Também nesse processo, as atividades de manutenção do *framework* são programadas. Esse processo é apresentado pela figura 7.

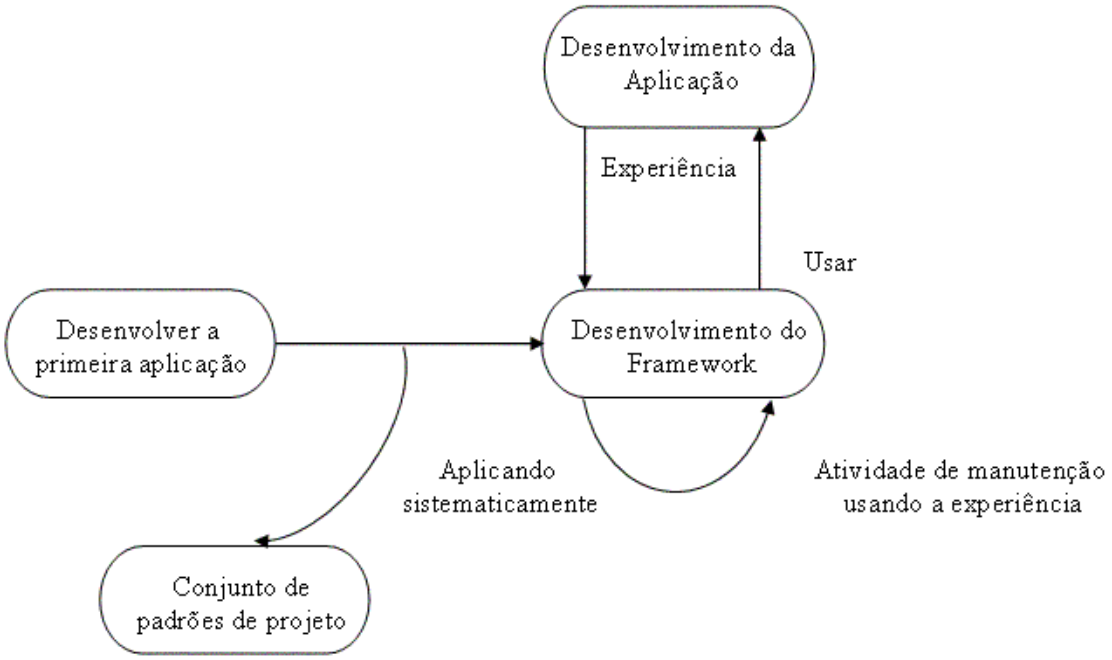


Figura 7 - Desenvolvimento de *frameworks* utilizando padrões de projeto. Adaptado de [48]

3.2.4.4 Processo Geral de Desenvolvimento de *Frameworks*

Elementos em comum são identificados aos processos de desenvolvimento de *frameworks* [48] [49]:

- Análise do domínio do problema. Esta é executada sistematicamente ou através do desenvolvimento de uma ou algumas aplicações do domínio onde abstrações-chaves são encontradas;
- A primeira versão do *framework* é desenvolvida utilizando-se as abstrações encontradas;
- Uma ou mais aplicações são desenvolvidas baseadas no *framework*. O teste é importante para verificar se o *framework* é realmente reutilizável;
- Problemas encontrados durante o desenvolvimento da aplicação baseados no *framework* são capturados e resolvidos na próxima versão;
- Após repetir o ciclo várias vezes, o *framework* vai atingindo um nível de maturidade aceitável, possibilitando, desta forma, a sua reutilização por vários usuários.

A figura 8 apresenta esse processo de desenvolvimento:

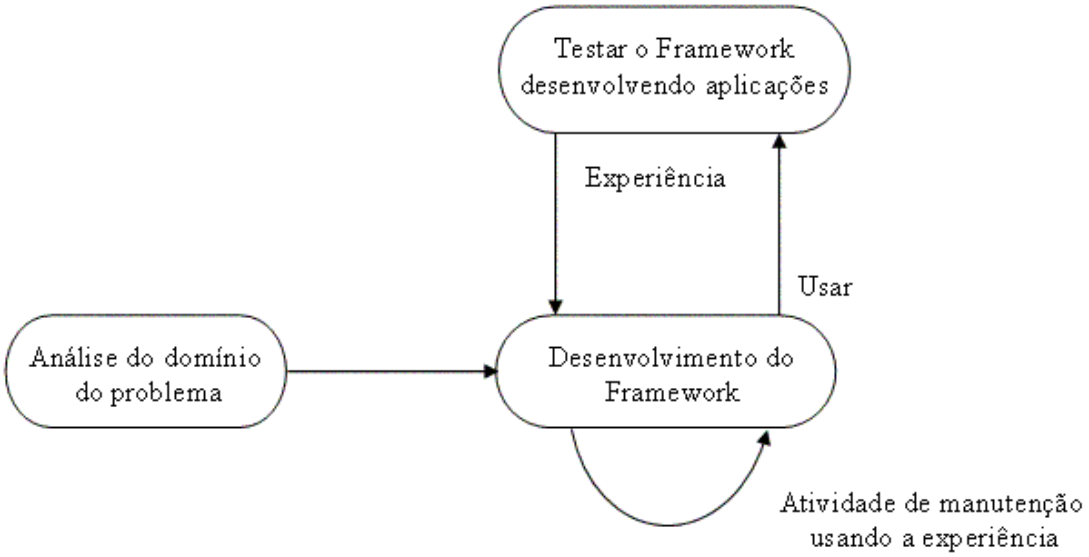


Figura 8 - Processo geral de desenvolvimento de *frameworks.* Adaptado de [48]

3.2.5 Documentação de *Frameworks*

Um *framework* existe para suportar o desenvolvimento de uma família de aplicações. O reuso envolve um desenvolvedor da aplicação, ou um time de desenvolvedores da aplicação, customizando o *framework* para construir uma aplicação concreta. Normalmente, um *framework* é desenvolvido por projetistas experientes que têm um profundo conhecimento do domínio da aplicação e uma longa experiência em projeto de *software* [22].

No desenvolvimento de *software*, comentários no código e documentação constituem uma parte necessária para o entendimento e manutenção por parte de um outro desenvolvedor que não tenha participado do desenvolvimento. Pensando-se em *frameworks*, torna-se importante o uso da documentação, visto que se um desenvolvedor não entender o *framework*, não poderá utilizá-lo [22].

A documentação de um *framework* tem o objetivo de introduzir rapidamente como o *framework* deve ser utilizado de um modo geral. Essa deve ser descrita em diferentes níveis de abstração desde que atenda as necessidades de desenvolvedores com vários níveis de experiência. Existem diferentes necessidades para um usuário experiente do *framework* e para um usuário de primeiro instante [48].

Em geral, a documentação deve disponibilizar [48]:

- O propósito do *framework*, ou seja, uma descrição do domínio do problema para o qual foi desenvolvido;

- Como usar os fundamentos do *framework*. Descrever como utilizar o *framework*, sem entrar nos detalhes de como funciona. Deve ser apresentado onde e como o *framework* pode ser adaptado ou estendido;

- Exemplos de utilização dos fundamentos do *framework*, onde cada exemplo é apresentado passo a passo, sendo interessante mostrar seu uso em diferentes contextos;

- Projeto do *framework*. Descreve as classes e os relacionamentos entre as classes do *framework*.

O reuso de *framework* pode ser feito por diferentes tipos de pessoas. Assim, diferentes tipos de informações devem ser disponibilizados, com diferentes níveis de abstração e detalhe [22]:

- Reuso por um desenvolvedor de aplicações – um desenvolvedor de aplicações quer saber como customizar o *framework* para produzir a aplicação desejada. Ele precisa conhecer os *hot-spots* relevantes e como customizá-los, quais classes especializar, quais métodos sobrescrever, como manter um protocolo de colaboração entre as classes. A documentação deve ser prescritiva. O desenvolvedor pode não ser um grande conhecedor do domínio do problema ou ser um experiente desenvolvedor de *software*.

- Reuso por um desenvolvedor responsável pela manutenção do *framework* – o desenvolvedor responsável pela manutenção e evolução do *framework* deve entender seu projeto. Existem diversos aspectos do *framework* que devem ser conhecidos: o domínio da aplicação ao qual o *framework* está relacionado, a arquitetura e todos os relacionamentos entre classes, as razões que influenciam a escolha dos *hot-spots*, quais padrões de projeto utilizados e quais provêm flexibilidade em cada um dos *hot-spots*, a responsabilidade de cada classe, as interfaces, a colaboração entre as classes, entre outros. A documentação deve ser descritiva. O desenvolvedor é *expert* em *software* e no domínio do problema.

- Reuso por um desenvolvedor de outro *framework* – desenvolvedores de *frameworks* buscam idéias em *frameworks* existentes, mesmo que esses pertençam a outros domínios. O interesse principal é nos padrões de projeto que provêm flexibilidade aos *hot-spots*. O desenvolvedor requer informação primeiramente de um alto nível de abstração, assim, os tipos de informações necessitam ser similares aos utilizados pelos responsáveis pela manutenção. Os desenvolvedores são *experts* em projeto de *software*, mas não necessariamente *experts* no domínio do *framework* que estão reutilizando.

- Reuso por um verificador – alguns desenvolvedores de aplicações e desenvolvedores de *frameworks* podem estar preocupados com o rigor dos sistemas desenvolvidos por eles. Eles podem ter a necessidade de verificar certas propriedades do sistema, de modo a satisfazer requisitos que envolvem algum tipo de rigidez. Isto requer métodos formais de especificação e verificação. Especificação para reuso é geralmente mais descritiva do que prescritiva: o reutilizador deve perceber as implicações da especificação em termos da customização desejada. Alguns aspectos como: protocolos que podem ser customizados, colaborações que devem ser suportadas por novas subclasses, entre outros, devem ser claramente especificados.

- Sumário – a maioria dos reutilizadores de *frameworks* são desenvolvedores de aplicações e verificadores executando em último instante.

Muitos usuários não estão interessados nos detalhes de um *framework*, mas estão em busca de uma documentação que descreva como utilizá-los. Alguns estilos de documentação são apresentados em [22] [48]:

- *Cookbook*: um *cookbook* é uma coleção de receitas. Um guia aos índices das receitas é fornecido geralmente, tal como uma tabela de índices ou a primeira receita agindo como uma visão geral para o *cookbook*. Existem algumas diferentes abordagens que utilizam a idéia de *cookbook* existente [48]: MVC é usado para construir interfaces para usuários em *Smalltalk-80*, que descreve como decompor uma aplicação que tem uma interface de usuário. A metáfora MVC é descrita e refinada pelo padrão de projeto *Observer* [28], nessa abordagem é descrita uma solução por inteiro para um determinado problema; *MacApp* [1] é um *framework* com aplicação em interfaces gráficas de usuário desenvolvido pela *Apple Inc*. Essa abordagem trata de um conjunto de receitas onde cada receita resolve um problema em particular; *the Active* utiliza metapadrões como um caminho para descrever um *framework* orientado a objetos. Em cada metapadrão é definido qual método *template* ou classe é, qual é o método *hook* ou classe. Um metapadrão é unido às partes no *framework* onde a semântica deve ser variada.

- Padrões: um padrão de projeto fornece uma abstração sobre o nível de classes e objetos. Um padrão é um metaconhecedor sobre como incorporar flexibilidade para um *framework*. A descrição de um padrão de projeto explica o problema e seu contexto, a solução e uma discussão das conseqüências de adoção da solução. Uma abordagem utilizada para descrever padrões é a *Motif* [41], a qual descreve como usar um *framework*. Os *Motifs* utilizam uma estrutura comum. Primeiro uma descrição do problema, seguido de uma discussão sobre os diferentes caminhos para resolver o problema. O *Motif* termina com um sumário de soluções e ponteiros para outros *Motifs*. Outra abordagem é a de padrão de projeto orientado a objetos, que descreve partes de um projeto orientado a objetos. Padrões de projeto são úteis no contexto de descrever partes de um projeto de *framework* através de diagramas para os padrões de projeto e vocabulário introduzidos. Em alguns casos onde o *framework* oferece aos usuários poucas entradas para adaptar o *framework*, padrões de projeto podem ser uma aproximação apropriada.

- Linguagem de descrição do *framework*: relacionamentos hierárquicos, referências e seqüências de criação de objetos são itens necessários para serem descritos quando documentar um *framework*. Devem existir informações que respondam as seguintes perguntas: Quais novas classes devem ser fornecidas para o *framework*? Que classes deverão ser utilizadas do *framework*? Que classes deverão ser subclasses? Em resumo, a documentação de um *framework* deve consistir de: uma descrição do *framework* em uma linguagem de descrição do *framework*; diagramas de classes; programas de exemplos e receitas.

Esses são alguns exemplos de estilos de documentação de *frameworks* encontrados em [22] [48], que podem ser utilizados com o objetivo de facilitar o entendimento de um *framework* para que possa ser reutilizado. Para facilitar o aprendizado do *framework*, podem-se combinar estilos ou utilizar um em particular.

3.2.6 Vantagens e Desvantagens do Uso de *Frameworks*

Frameworks estão se tornando cada vez mais comuns e importantes. Eles são a maneira pela qual os sistemas orientados a objetos conseguem a maior reutilização. Aplicações orientadas a objetos maiores terminarão por consistir-se de camadas de *frameworks* que cooperam uns com os outros. Assim como a sua utilização proporciona benefícios como o reuso, também apresenta algumas desvantagens como o alto custo de desenvolvimento, se comparado a uma única aplicação [27] [28].

Vantagens do uso de *frameworks* [27]:

- Reutilização de código e projeto;
- Facilidade de manutenção – devido às aplicações desenvolvidas no *framework* possuírem partes em comum;
- Qualidade – devido à reusabilidade, projeto testado e funcionamento comprovado.

Desvantagens do uso de *frameworks* [27]:

- Alto custo – é necessário mais tempo e dinheiro do que comparados com uma única aplicação;
- Mudanças no domínio de problema – o domínio deve ser relativamente estável, se não, esforços e dinheiro serão perdidos;
- Evolução – qualquer mudança no *framework* irá afetar as aplicações desenvolvidas a partir dele.

3.2.7 Padrões de Projeto x *Frameworks*

Padrões de projeto e *frameworks* possuem algumas similaridades, porém, segundo GAMMA et al [28] eles diferem em três aspectos principais:

- Padrões de Projeto são mais abstratos que *frameworks*. *Frameworks* podem ser materializados em código. Isto não é o caso de padrões de projeto, onde somente exemplos de padrões de projeto podem ser materializados em código. *Frameworks* podem ser escritos em uma linguagem de programação, sendo não apenas estudados, mas executados e reutilizados diretamente. Ao contrário, padrões de projeto têm que ser implementados cada vez que eles são utilizados. Os padrões de projeto também explicam as intenções, custos e benefícios e conseqüências de um projeto, que não é o caso de *frameworks*.
- Padrões de projeto são elementos de arquitetura menores que *frameworks*. Um *framework* pode conter um número de padrões de projeto, mas a recíproca nunca é verdadeira. Assim, padrões de projeto não têm maior impacto na arquitetura da aplicação.

- Padrões de projeto são menos especializados que *frameworks*. *Frameworks* sempre têm um particular domínio de aplicação. Por exemplo, um *framework* para um editor gráfico poderia ser usado na simulação de uma fábrica, mas não seria confundido com um *framework* para simulação. Em contraste, padrões de projeto, podem ser usados em quase qualquer tipo de aplicação.

Frameworks estão cada vez mais comuns e importantes no desenvolvimento de aplicações. Tornaram-se a forma pelas quais sistemas orientados a objetos conseguem a maior reutilização. A maior parte do projeto e do código da aplicação virá, ou será influenciada pelos *frameworks* que utiliza.

3.3 Aplicação de Padrões de Projeto e *Frameworks* no Comércio Eletrônico

Aplicar Padrões de Projeto e *frameworks* em áreas onde os requisitos de *software* costumam mudar rapidamente mostra-se adequada às aplicações via *Web*.

Planejar o desenvolvimento de um sistema para comércio *on-line* exige um esforço em levantar e identificar os requisitos funcionais. Escolher a tecnologia apropriada para o desenvolvimento do sistema exige que os requisitos estejam compreendidos. O comércio *on-line* deve compreender todas e mais funções executadas no comércio físico. Deve oferecer aos clientes meios e caminhos para que possam efetuar uma compra de forma rápida e tranqüila.

O uso de *frameworks* permite que todo o trabalho de obtenção de requisitos, bem como o projeto e a implementação de sistemas sejam reaproveitados. Com a utilização de padrões de projeto, o uso de *frameworks* proporciona redução de tempo e recursos necessários para o desenvolvimento, além do reuso de *software*.

No próximo capítulo, será apresentado o novo paradigma de programação orientada a aspectos (POA), e alguns aspectos da linguagem de programação orientada a aspectos *AspectJ*.

4 PROGRAMAÇÃO ORIENTADA A ASPECTOS

A programação orientada a aspectos (POA)[8] [38] emerge como uma técnica de programação para solucionar algumas saídas levantadas em algumas decisões de projeto. Estas decisões de projeto não podem ser bem resolvidas com técnicas de programação procedurais, nem de programação orientada a objetos, já que essas técnicas entrecortam a implementação em muitas partes do código, ou a funcionalidade básica do programa. Isso acaba forçando na decisão de projeto por implementar códigos espalhados pelo sistema, resultando num código entrelaçado que é excessivamente difícil para desenvolver e dar manutenção [38].

POA é uma nova evolução na linha de tecnologia para separação de interesses [21] – tecnologia que permite projeto e código serem estruturados de modo que os desenvolvedores possam pensar sobre o sistema. POA procura obter uma maior modularidade de projetos de *software* em situações práticas, significando que interesses estão melhor localizados do que espalhados e têm interfaces melhor definidas com o resto do sistema, assim, torna a estrutura do *software* mais clara e de mais fácil manutenção.

4.1 Separação de Interesses

Em Ciência da Computação, as linguagens de programação têm evoluído dos códigos de máquina e linguagens de programação *assembly* para uma variedade de paradigmas tais como: tradução de fórmulas, programação procedural, programação funcional, programação lógica, e programação orientada a objetos [20] [43].

Separação de interesses (do inglês *separation of concerns*) é um dos princípios da Engenharia de *Software* que é aplicado durante todo o processo de desenvolvimento de *software* [29] [53]. Decompor um sistema em pequenas partes é uma forma essencial de reduzir a complexidade e garantir sua evolução, assim, aspectos individuais de um problema podem ser separados e tratados individualmente.

Durante o desenvolvimento, muitas decisões devem ser tomadas. Essas decisões devem ser bem isoladas uma das outras. Entretanto, muitas decisões são fortemente relatadas e interdependentes, e é praticamente impossível conduzir todas as saídas que se deve levar em conta ao mesmo tempo. Assim, para controlar a complexidade de um projeto, faz-se necessário separar os diferentes interesses.

De um modo geral, o avanço da tecnologia de programação proporcionou uma melhora na habilidade do desenvolvedor de *software* de conseguir claramente separar interesses, ou "conseguir identificar, encapsular, e manipular somente partes do *software* que são relevantes para um conceito, objetivo, ou propósito" [53]. Em projetos de *software* – para fins de simplicidade, legibilidade, e conseqüentemente facilidade na manutenção e reutilização – é importante que

[8] O termo programação orientada a aspectos foi proposto por Kiczales et al [38] em 1997 no Xerox Palo Alto Research Center.

interesses[9] (do inglês *concerns*) relevantes para um domínio de aplicação estejam localizados em módulos separados. Interesses são os critérios preliminares para decompor um *software* em partes menores, mais gerenciáveis e compreensíveis, cada um desses direciona para um ou mais interesses.

De forma geral, todas as técnicas de programação oferecem suporte a esse tipo de separação de interesses[10] (DIJKSTRA, 1976) [18], porém cada uma à sua maneira (utilizando sub-rotinas, procedimentos, APIs) e em graus diferentes. Entretanto, estas técnicas podem não ser suficientes para separação de algum interesse especial encontrado em sistemas mais complexos.

Além de POA, outras abordagens têm sido consideradas na separação de interesses, tais como: programação adaptativa [44], programação orientada a assuntos [30], filtros composicionais [16], programação intencional [57], entre outras.

4.2 Paradigma Orientado a Aspectos

A programação orientada a objetos tornou-se o paradigma de programação dominante, onde um problema é decomposto em objetos que abstraem comportamento e dados em uma única entidade. A tecnologia orientada a objetos trabalha com metodologias de desenvolvimento de *software* e ferramentas, incluindo metodologia orientada a objetos, ferramentas de análise e projeto, e linguagens de programação. O desenvolvimento de aplicações complexas tais como: interfaces gráficas de usuários, sistemas operacionais e aplicações distribuídas, referindo-se à redução da complexidade de escrita e manutenção de código, tornaram-se possíveis com orientação a objetos [20] [43].

Um dos maiores benefícios das linguagens de programação orientadas a objetos tais como: C++ e *Smalltalk* é o grau elevado de reuso que se pode conseguir com sistemas bem-projetados. Um grau elevado de reuso significa que menos códigos devem ser escritos quando uma nova aplicação surgir, conseqüentemente, menos códigos para dar manutenção, aumentando a produtividade de desenvolvimento e o suporte a mudanças de requisitos. Reuso de *software* pode ser visto de muitas formas, por exemplo: reuso de linhas individuais de código, classes específicas, conjuntos de classes, componentes, entre outros [13].

Embora a tecnologia de objetos ofereça maior habilidade para separação de interesses, ainda possui dificuldade em localizar interesses que envolvam restrições globais, que não são utilizadas por um módulo somente, ou mesmo para módulos que possuam dependência entre si [20] [43]. OSSHER et. al [52], apresenta algumas limitações do paradigma orientado a objetos. Limitações que surgem ao utilizar a tecnologia de objetos para desenvolver sistemas grandes, conjuntos de interoperabilidade ou aplicações integradas, ou sistemas que evoluam de formas inesperadas. Essas limitações incluem algumas fraquezas que são apresentadas a seguir [52]:

- Extensão e evolução de sistemas não-invasivos: criando extensões para, e configurações de, *software*, sem modificar o código fonte original;

---

[9] O conceito de interesse (concern) foi introduzido por DIJKSTRA (1976) [18] há quase trinta anos. De uma maneira geral, um interesse pode indicar tanto um requisito funcional (código de negócio), quanto não funcional (propriedades que afetam o comportamento) de um sistema.

[10] O princípio idealizado por DIJKSTRA (1976) [18], refere-se à separação de interesses como a divisão de conhecimento em partes menores, para poder entender objetivamente cada uma delas. Portanto, a implementação de um sistema particionado em módulos que contêm cada um dos interesses comuns é uma conseqüência da utilização desse princípio.

- Reuso e integração em larga escala: tecnologia de objetos possui mecanismos insuficientes para conseguir reuso em larga escala ou integração de componentes separados, sem um replanejamento significante;

- Decomposição de sistemas: decomposição de sistemas de objetos/classes é usual para modelar partes de um sistema de central de dados. Outras decomposições, por exemplo: por características, por função, por requisitos, são melhores para modelar outros aspectos. Sem eles, manutenção, compreensão e reusabilidade são prejudicadas;

- Desenvolvimento em múltiplos-times/descentralizados: desenvolvimento orientado a objetos conduz ao excesso da disputa compartilhada, classes centralizadas, com isso, o desenvolvedor é forçado a concordar com um único modelo de domínio, não podendo utilizar modelos apropriados para programar suas rotinas;

- Entrelaçamento e espalhamento de código: o entrelaçamento de código de negócio com código de interface com o usuário, e o espalhamento de código de acesso a dados em vários módulos do sistema.

Algumas técnicas padrões da tecnologia de objetos tais como: subclasses, *frameworks* e padrões de projetos, amenizam algumas dessas limitações, mas não as resolvem adequadamente. Uma limitação maior é que o tipo de flexibilidade que eles provêem requer preplanejamento, porém, custos de desenvolvimento e execução, podem proibir preplanejamento para todas as eventualidades. Assim, outros mecanismos de programação tornam-se necessários, a fim de flexibilizar o desenvolvimento de *software* orientado a objetos.

POA está baseada na idéia de que sistemas são melhor programados separadamente pela especificação de vários interesses (propriedades ou áreas de interesses) de um sistema e alguma descrição do relacionamento entre eles. O objetivo de POA é tornar projeto e código mais modulares, significando que interesses são melhor localizados do que dispersos e têm relações de interfaces bem definidas com o resto do sistema [21]. Interesses podem variar de noções de alto nível, como segurança e qualidade de serviços a noções de baixo nível, como sincronização, manipulação de *buffers* de memória, *caching* e *logging*. Eles podem ser tanto funcionais, como: características ou regras de negócios, quanto não-funcionais, como: sincronização e gerenciamento de transações e persistência [29].

Segundo LIEBERHERR et. al [21], "dois interesses se entrecortam se o método relacionado para aqueles interesses se cruzarem". POA se preocupa com os interesses transversais (*crosscuting concerns*) e descrições, projeto e implementação para esses interesses. Os artefatos utilizados para descrever, projetar e implementar um dado interesse são chamados de métodos. Um método está relacionado para um interesse se o método contribui para a descrição, projeto, ou implementação do interesse.

A figura 9 apresenta um exemplo ilustrado por KICZALES et. al [21], para melhor entender interesses transversais e aspectos:

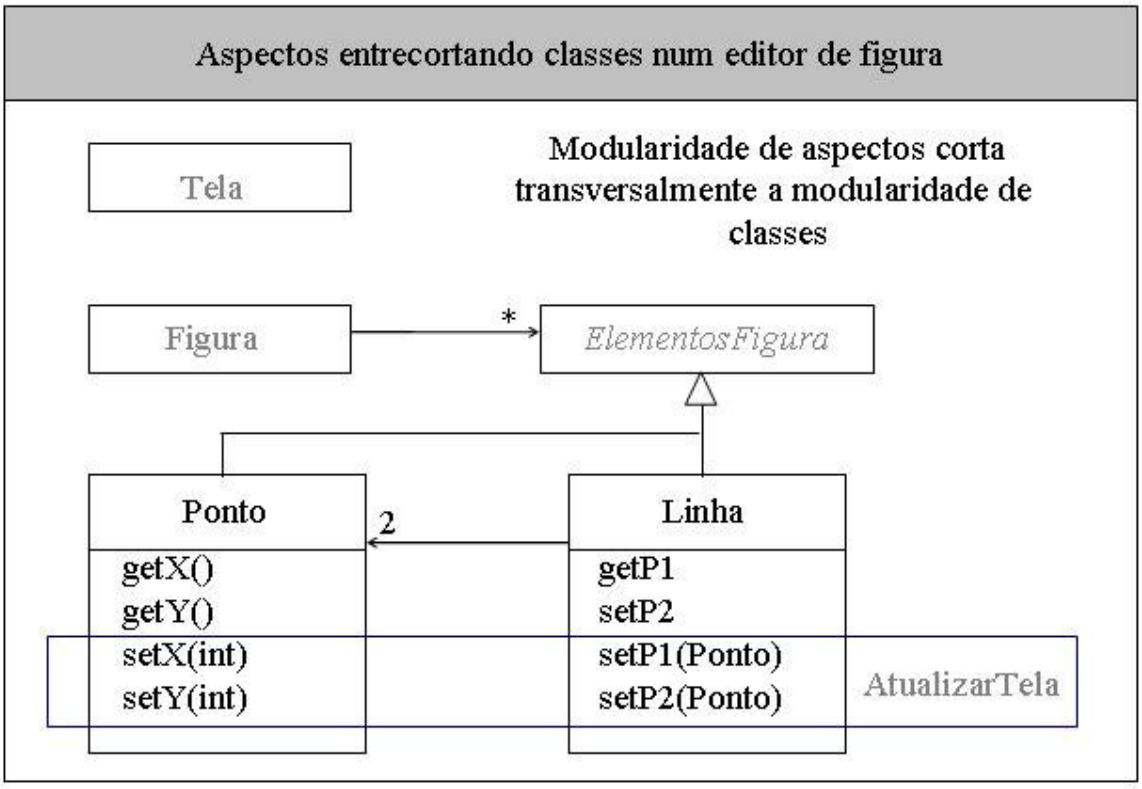


Figura 9 - Diagrama de um editor de texto. Adaptado de [21]

O diagrama em UML [14] [29] apresenta uma simples representação de um editor de texto, na qual existem duas classes concretas para a classe ElementosFigura, que são: Ponto e Linha. Essas classes manifestam uma boa modularidade, na qual o código pesquisado em cada classe está internamente ligado (coesão) e cada classe tem uma interface limpa e bem definida. Mas considerando o interesse que o gerenciador da tela deverá ser notificado toda vez que ElementosFigura se mover, isto requer que cada método que mover ElementosFigura faça a notificação.

Existem dois interesses para o editor: manter o registro da posição de cada elemento da figura (interesse de dados) e atualizar a tela sempre que um elemento mover-se (interesse de característica). A caixa sobre as classes Ponto e Linha na figura 9 está desenhada em torno de cada método que deve implementar esse interesse de característica, exatamente como as caixas ponto e linha são desenhadas em torno de cada método que implementa esses interesses. Observa-se que a caixa para atualizarTela não está dentro nem ao redor das outras caixas na figura. Essa representação é o que se chama de interesses transversais. Fazendo uso somente da programação orientada a objetos, a implementação de interesses transversais tenderia a estar espalhada pelo sistema, exatamente como está representado na figura. Utilizando os mecanismos de POA, pode-se modularizar a implementação do comportamento de atualizarTela para um único aspecto. A programação orientada a objetos seria utilizada para o interesse de dados, onde decompõe o elemento gráfico assim que o interesse de dados for localizado ordenadamente [21] [43].

POA parece bastante promissora [51] para promover a melhora na separação de interesses, conduzindo à produção de sistemas de *software* mais fáceis de dar manutenção e reutilizar. POA aumenta a modularidade do sistema, separando o código que implementa funções específicas, que

afetam diferentes partes do sistema, chamadas interesses transversais, das partes que são mais específicas que compõem as regras de negócio. Alguns exemplos de interesses transversais mais comuns, referentes a requisitos não-funcionais, podem ser: persistência, distribuição, controle de concorrência, tratamento de exceções e depuração.

4.3 Conceitos

O objetivo de sistemas de POA não é somente prover um caminho para expressar interesses transversais em sistemas computacionais, mas, também, assegurar que esses mecanismos sejam conceitualmente diretos e tenham execuções eficientes [21].

KICZALES et. al [38] separa as propriedades que devem ser implementadas em um sistema em dois tipos:

- componentes: se podem ser facilmente modularizados em um procedimento generalizado, por exemplo: objeto, método, procedimento, API. Esses componentes tendem a ser unidades de decomposição funcional de um sistema, tais como: filtros de imagem e clientes de banco.

- aspectos: se não podem ser facilmente modularizados em um procedimento generalizado. Os aspectos são propriedades que precisam ser satisfeitas em vários componentes de um sistema, como sincronização e persistência.

POA é uma nova tecnologia para separação de interesses transversais para unidades únicas chamadas aspectos. Aspectos são definidos como unidades modulares que agrupam características em comum de componentes do sistema, de modo que possam ser utilizados em qualquer ponto sem precisar modificar toda a estrutura de classes. Assim, o objetivo de POA é possibilitar ao desenvolvedor separar componentes e aspectos de forma bastante clara, e ainda compô-los entre si da maneira apropriada, de forma a constituir o sistema desejado [38].

Com POA, cada aspecto pode ser expresso num formulário separado, e pode então ser automaticamente combinado junto, na execução final do formulário por um combinador (*aspect weaver*). Como resultado, um único aspecto pode contribuir para a implementação de um número de procedimentos, módulos, ou objetos, aumentando a reusabilidade de código [43].

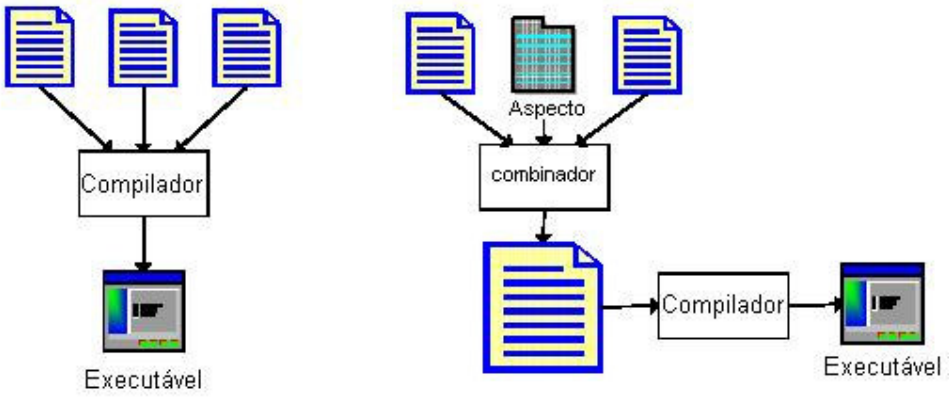


Figura 10 - Abordagem tradicional x POA. Adaptado de [43]

A figura 10 apresenta uma comparação entre a abordagem tradicional e POA. POA permite que aspectos e códigos sejam combinados juntos por um combinador (*aspect weaver*) antes que o programa seja compilado e executável. No exemplo do editor de figuras, apresentado na figura 9, o combinador pode automaticamente inserir chamadas atualizarTela para todo conjunto de operações, permitindo, assim, a separação de interesses transversais a nível de código fonte [43].

Na abordagem tradicional apresentada na figura 10, a implementação de uma aplicação, segundo KICZALES et. al [38], consiste de: i) uma linguagem, ii) um compilador (ou interpretador) para esta linguagem, e iii) um programa escrito na linguagem que implementa a aplicação.

A implementação de uma aplicação baseada em POA, consiste de [38]:

i.a) uma linguagem de componentes,

i.b) uma (ou mais) linguagem(ns) de aspectos,

ii) um combinador (*aspect weaver*), capaz de combinar a (s) linguagem (ns), ou seja, os programas de componentes e de aspectos, de forma a gerar o programa final,

iii.a) um programa de componente, que implementa os componentes usando a linguagem de componentes e,

iii.b) um (ou mais) programas de aspectos que implementa os aspectos usando a (s) linguagem (ns) de aspectos.

Projetar sistemas baseados em POA envolve entender o que deve entrar na linguagem de componentes, o que deve entrar na linguagem de aspectos e o que deve ser compartilhado entre as duas linguagens [38]. Uma linguagem de componentes normalmente é uma linguagem de programação que se conforma a um dos paradigmas de programação dominantes, como Java, Lisp, C++, Pascal, C, etc. Algumas linguagens de aspectos podem ser genéricas quanto à sua aplicação,

como AspectJ [39] [40], AspectC [6], AspectPHP [10] e AspectC# [7]. Outras são voltadas a preocupações específicas, onde podem ser utilizadas múltiplas linguagens de aspectos, uma para cada preocupação. Um exemplo pode ser visto em D [46], onde são usadas Ridl (para distribuição) e COOL (para concorrência).

O combinador de aspectos (*aspect weaver*) deve processar a linguagem de componentes e de aspectos, fazendo uma recomposição para produzir o código final da aplicação. Esse processamento se dá através da identificação dos pontos de junção, que são elementos semânticos da linguagem de componentes com as quais os programas de aspectos se coordenam. Alguns exemplos comuns de pontos de junção são: invocação de métodos, geração de exceções, criação de objetos, entre outros [38]. Esses combinadores podem atuar em tempo de compilação ou de execução. Implementações de combinadores em tempo de execução têm a possibilidade de permitir a adição/exclusão de aspectos com a aplicação em funcionamento.

LIEBERHERR et. al [21] apresenta cinco elementos da linguagem POA para modularizar interesses transversais [21]:

- um modelo de pontos de junção, que descrevem os *hooks* onde realces podem ser adicionados;
- meios de identificar pontos de junção;
- meios de identificar o comportamento dos pontos de junção;
- unidades encapsuladas que combinam especificações de pontos de junção e comportamentos de realces;
- e um método de assessoramento de unidades para um programa.

4.4 Benefícios da Programação Orientada a Aspectos

POA oferece alguns benefícios em relação ao desenvolvimento de sistemas, tais benefícios podem ser vistos a seguir [33]:

- Execução modularizada de interesses transversais: POA dirige-se a cada interesse separadamente com mínimo acoplamento, resultando em execuções modularizadas mesmo na presença de interesses transversais. Tal execução produz um sistema com menos código duplicado. Uma vez que a execução de cada interesse esteja separada, ajuda também a reduzir a desordem no código. Assim, a execução modularizada também resulta em um sistema que é fácil de entender e dar manutenção;
- Sistemas facilmente evoluídos: uma vez que os módulos que fazem uso de aspectos não precisam ter conhecimento dos interesses transversais, torna-se fácil adicionar novas funcionalidades, criando novos aspectos. Assim, quando adicionar novos módulos no sistema, o aspecto existente os entrecorta, ajudando a criar uma evolução coerente;

- Ligações atrasadas em decisões de projeto: com POA, um arquiteto pode atrasar decisões de projeto para requisitos futuros, desde que possam ser implementadas como aspectos separados;

- Mais reuso de código: Porque AOP implementa cada aspecto como um módulo separado? Para que cada módulo seja mais facilmente acoplado. Por exemplo: pode-se usar um módulo interagindo com um banco de dados num aspecto *logger* separado com um diferente requisito de *logging*.

4.5 Programação Orientada a Aspectos em *AspectJ*

AspectJ é uma extensão orientada a aspectos de propósito geral para a linguagem de programação Java [32] [39] [40], originalmente desenvolvida de um trabalho de GREGOR KICZALES junto ao laboratório Palo Alto Research Center (PARC), da Xerox [8], e hoje é mantida pelo projeto de código aberto Eclipse [19].

AspectJ adiciona ao Java [32] somente um novo conceito, chamado de ponto de junção, e este é um nome justo para o conceito de Java existente. Adiciona ao Java alguns construtores novos, tais como: conjunto de pontos de junção, adendos, declaração inter-tipos e aspectos. O conjunto de pontos de junção e adendos afeta dinamicamente o fluxo de um programa, declaração inter-tipos afeta estaticamente uma hierarquia de classes de um programa, e aspectos encapsulam esses novos construtores [9].

AspectJ realiza a modularização de interesses transversais, usando pontos de junção e adendos. Pontos de junção são pontos bem definidos no fluxo de um programa e adendos definem código que é executado quando pontos de junção são alcançados [40]. Estas são as partes dinâmicas de *AspectJ*. Com poucos construtores, *AspectJ* provê suporte para execução modular de uma escala de interesses transversais [9]. Utilizando aspectos escritos em *AspectJ*, pode-se afetar a execução normal de um programa escrito em Java [32], na qual é um componente da linguagem de uma implementação baseada em POA usando *AspectJ*. Um conjunto de pontos de junção escolhe quais pontos de junção deixar de fora e os valores desses pontos.

AspectJ possui também diferentes declarações inter-tipos, que permitem que o programador modifique a estrutura estática de um programa, a saber, os membros das classes e os relacionamentos entre classes.

Em *AspectJ*, aspectos são unidades de modularização para interesses transversais. Comportam-se um tanto como classes em Java, mas podem também incluir conjuntos de ponto de junção, adendos e declaração inter-tipos.

4.5.1 Construtores Básicos da Linguagem

4.5.1.1 Pontos de Junção

Conforme visto na seção anterior, pontos de junção é um ponto bem definido no fluxo de execução de um programa. *AspectJ* define muitos tipos de pontos de junção [9], que serão apresentados a seguir:

- *Method call*: quando um método é chamado, não incluiu chamadas "super" de classes não-estáticas;
- *Method execution*: quando o corpo do código para um método atual executar;
- *Constructor call*: quando um objeto é construído e o construtor inicial desse objeto é chamado (isto é, não para chamadas de construtores "super" ou "*this*");
- *Construction execution*: quando o corpo do código para um construtor atual executar, depois de chamadas de construtores "super" ou "*this*";
- *Static initializer execution*: quando o inicializador estático para uma classe executar;
- *Object pre-initialization*: o código de inicialização do objeto para uma classe particular executa antes. Isto abrange o tempo entre o início da primeira chamada ao construtor e o início do construtor-pai;
- *Object initialization*: quando o código de inicialização do objeto para uma classe particular executa. Isto abrange o tempo entre o retorno do construtor-pai e o retorno da primeira chamada ao construtor;
- *Field reference*: quando um campo não-constante é referenciado;
- *Field set*: quando um campo for atribuído para;
- *Handler execution*: quando um alimentador de exceção executar;
- *Advice execution*: quando o corpo do código para um adendo executar.

Cada ponto de junção potencialmente tem três partes de estados associados com ele: a execução do objeto corrente, o alvo do objeto, e uma disposição dos argumentos dos objetos. Estes são expostos por três estados de exposição de conjunto de pontos de junção, que são: "*this*", "*args*", "*target*", respectivamente.

4.5.1.2 Conjuntos de Pontos de Junção

Em *AspectJ*, designadores de conjuntos de pontos de junção, ou simplesmente conjuntos de pontos de junção, identificam coleções de pontos de junção no fluxo de um programa. Conjuntos de pontos de junção são utilizados primariamente por adendos. Podem ser compostos através de operadores "*booleanos*" ('&&', '||' e '!') para construir acima de outros conjuntos de pontos de junção. Alguns exemplos de conjuntos de pontos de junção são apresentados a seguir [9]:

- *call* (método): captura a chamada do método no ponto de junção;
- *execution* (método): captura a execução do método no ponto de junção;

- *handler* (tipo): captura a execução de alimentadores de exceção;
- *adviceexecution* (): captura a execução de um adendo;
- *this* (tipo), *target* (tipo): definem o tipo de objeto utilizado para discriminação;
- *args* (tipo): escolhe em cada ponto de junção onde os argumentos são instâncias do mesmo tipo;
- *within* (tipo): o código de execução é definido na declaração de um dos tipos em tipo. Isso inclui inicialização de classes, inicialização de objetos, execução de métodos e construtores de pontos de junção para o tipo, tão bem como algum ponto de junção associado com as indicações e expressões do tipo;
- *cflow* (conjunto de ponto de junção): captura pontos de junção baseados no fluxo de controle.

Os conjuntos de pontos de junção apresentados acima são chamados de primitivos, que identificam pontos de junção em diferentes caminhos. A tabela 6, apresenta alguns exemplos de como utilizar conjuntos de pontos de junção.

Conjunto de pontos de junção	Descrição
call (void Ponto.*setX(int))*	Identifica alguma chamada para Ponto.*setX(int).*
call (void Ponto.*setX(int))* || *call (void* Ponto.*setY(int))*	Identifica alguma chamada entre o método *setX* ou *setY* definidos em Ponto.
call (public * Figura.*(..))*	Identifica algum método público definido em Figura.
Pointcut move () : *call (void* Ponto.*setX(int))* || *call (void* Ponto.*setY(int))* || *call (void* Linha.*setP1(Ponto))* || *call (void* Linha.*setP2(Ponto));*	Define uma chamada ao conjunto de ponto de junção *move*.
cflow (move())	Designa todos os pontos de junção que ocorrem entre chamadas de métodos recebidas para os métodos em *move* e os retornos dessas chamadas.

Tabela 6 - Exemplos de conjuntos de pontos de junção. Adaptado de [9]

4.5.1.3 Adendos

Conjuntos de pontos de junção são utilizados na definição de adendos. Adendo é um método, tal como um mecanismo usado para declarar que determinado código deve executar em cada um dos pontos de junção no conjunto de pontos de junção [39]. *AspectJ* suporta três tipos de adendos [9]:

before, *after* e *around*. O tipo de adendo determina como interagir com os pontos de junção definidos. A seguir serão apresentados os tipos de adendos providos por *AspectJ*:

- *Before*: executa antes de cada ponto de junção;
- *After returning*: executa depois que cada ponto de junção retorna normalmente. Pode-se opcionalmente especificar o valor retornado, fazendo com que esse valor seja acessado;
- *After throwing*: executa depois que cada ponto de junção dispara uma *Throwable*. Se o tipo de *Throwable* for especificado, o adendo executa somente depois de cada ponto de junção que dispara a *Throwable* específica e têm-se acesso ao valor da exceção *Throwable*;
- *After*: executa de qualquer maneira normalmente depois que cada ponto de junção retorna ou não;
- *Around*: executa em qualquer lugar do ponto de junção. O ponto de junção pode ser executado pela chamada de um processamento, na qual toma o mesmo número e tipos de argumentos como em torno do adendo.

O exemplo a seguir, apresentado na figura 11, ilustra uma possível implementação do adendo *before*, na qual é responsável por incluir a execução de instruções antes dos pontos identificados pelo conjunto de ponto de junção *setter*.

```
before(Ponto P): setter(p) {
       System.out.println(“Ponto ”+p+”será modificado.”);
}
```

Figura 11 - Exemplo de implementação do adendo *before*

No exemplo da figura 11, o adendo imprime uma mensagem na tela antes de todas as chamadas dos métodos para *setX* ou *setY* no objeto Ponto. Como o objeto Ponto é um parâmetro do conjunto de ponto de junção, o conteúdo é exposto e o valor corrente do objeto identificado por “p” pode ser impresso. Uma outra possível implementação deste adendo, apresentada na figura 12, usando um conjunto de ponto de junção anônimo seria:

```
before(Ponto P): target(p) && (call(void setX(int)) || call(void setY(int))) {
       System.out.println(“Ponto ”+p+”será modificado.”);
}
```

Figura 12 - Exemplo de implementação do adendo *before* utilizando conjunto de pontos de junção

Além do *before*, conforme ilustrado nas figuras 11 e 12, existem outros tipos de adendos. Para ilustrar alguns desses adendos, consideramos o exemplo do Editor de Figura apresentado

anteriormente na seção 4.2, com o problema de atualizarTela, na qual é um único interesse transversal. Toda alteração na posição de um ponto, ou mover de uma linha para outra, a classe Figura deve disparar uma repintagem na tela. Se for inserida uma classe Ponto ou Linha, o código necessário para invocar a repintagem na tela, como um objeto Ponto ou Linha não deve estar preocupado com a mostra de uma única tela. Com *AspectJ*, pode-se definir, em um módulo isolado (um aspecto), o conjunto de pontos de junção e adendos necessários para executar a atualização, como ilustrado no exemplo da figura 13:

```
aspect AtualizarTela {
  pointcut move(): call(void ElementosFigura.moveBy(int, int)) ||
         call (void Ponto.setX(int)) ||
         call (void Ponto.setY(int)) ||
         call (void Linha.setP1(int)) ||
         call (void Linha.setP2(int));
  after() returning: move() {
     tela.needsRepaint();
  }
}
```

Figura 13 - Exemplo de implementação do aspecto AtualizarTela

O adendo *after* definido na figura 13 invoca o método *needsRepaint* em Tela quando algum dos pontos de junção escolhido pelo conjunto de pontos de junção *move* aconteça e nenhuma *Throwable* é disparada.

Para exemplificar o adendo *after throwing*, é apresentado um exemplo na figura 14:

```
after(Ponto p) throwing (PointOutOfBoundsException e) :setter(p){
      errorLogger.write("Ocorreu um erro enquanto ponto era movido
"+p+":"+e.getMessage());
}
```

Figura 14 - Exemplo de implementação do adendo *after throwing*

Com este adendo, pode-se escrever no arquivo de *log* de erros uma mensagem especificando o ponto onde um movimento foi executado e a mensagem de exceção disparada. Isto é feito incluindo algumas linhas de código em outras partes do sistema.

Para substituir ou mudar a execução normal de um ponto de junção, utiliza-se o adendo *around*. O exemplo apresentado na figura 15 ilustra como alterar a execução do método *setX*.

```
void around(int x): execution(void setX(int)) && args(x) {
       if  (x>=0 && x<=100){
           proceed(x);
       } else {
           proceed(10);
       }
}
```

Figura 15 - Exemplo de implementação do adendo *around*.

No adendo apresentado na figura 15, o parâmetro *int* chamado de *x* é um argumento do método *setX*. No corpo do adendo, verifica-se se o argumento está dentro de um intervalo de 0-100. Se estiver, a execução procede normalmente, caso contrário, a execução procede, mas o valor do argumento *x* é alterado para 10.

4.5.1.4 Declaração de Inter-Tipos

Aspectos podem declarar membros (campos, métodos e construtores) que são donos de outros tipos. Eles são chamados de membros de inter-tipos. Aspectos podem também declarar que outros tipos implementam novas interfaces ou estendem novas classes.

Um exemplo de declaração de membro de inter-tipos é apresentado na figura 16:

```
public String Ponto.getX(){ return this.x}
```

Figura 16 - Exemplo de implementação de declaração de inter-tipos

No código apresentado na figura 16, declara publicamente um método na classe Ponto, que pode ser invocado em qualquer local onde o objeto Ponto está acessível. Dentro deste novo corpo do método, *this* corresponde ao objeto Ponto executado atualmente. A declaração de um novo construtor para a classe é similar, como ilustra o exemplo na figura 17:

```
public Ponto.new (int x, int y) { this.x = x; this.y = y;}
```

Figura 17 - Exemplo de implementação de um novo construtor para a classe Ponto

No código apresentado na figura 17, são declarados dois parâmetros no construtor da classe Ponto que podem ser chamados em qualquer local onde a classe Ponto estiver acessível. Pode-se declarar um campo nesta classe, quase que da mesma forma como é feito no corpo da classe, somente incluindo o nome da classe antes do nome do campo, apresentado na figura 18:

```
public static final int Ponto.x = 0;
```

Figura 18 - Exemplo de implementação de um campo na classe Ponto

4.5.1.5 Aspectos

Aspectos são unidades modulares de execução de interesses transversais. Aspectos são definidos por uma declaração de aspectos, que têm uma forma similar à declaração de classes. Na declaração de aspectos podem ser incluídos declaração de pontos de junção, declaração de adendos, e outros tipos de declarações permitidas nas declarações de classes [39].

Como classes, aspectos podem ser instanciados, mas *AspectJ* controla como a instanciação acontece, dessa forma, não se pode usar o operador *new* de Java [32] para construção de uma nova instância de aspecto. Por padrão, cada aspecto é um *singleton*, assim uma instância de aspectos é criada [9].

A declaração apresentada na figura 19 define um aspecto que implementa o comportamento de manter-se informado se um ElementosFigura tem se movido recentemente. Este aspecto pode ser utilizado por um mecanismo na tela de atualização, para encontrar saída onde algo tenha sido alterado desde a última vez que a tela foi atualizada [39].

```
aspect moveTracking{
    static boolean flag = false;
    static boolean testAndClear(){
          boolean result = flag;
          flag = false;
          return result;
    }
    pointcut move():
          call(void ElementosFigura.moveBy(int, int)) ||
          call (void Ponto.setX(int)) ||
          call (void Ponto.setY(int)) ||
          call (void Linha.setP1(int)) ||
          call (void Linha.setP2(int));
    after() : move() {
          flag = true;
    }
}
```

Figura 19 - Exemplo de implementação do aspecto *moveTracking*. Extraído de [9]

POA introduz um novo estilo de decomposição e está se apresentando como uma saída promissora para separação de interesses, que são usualmente muito difíceis de fazer na programação orientada a objetos (OO).

Frameworks OO têm na sua estrutura um conjunto de unidades básicas (classes), da mesma forma, um *framework* OA, poderia ter na sua estrutura um conjunto de unidades básicas (classes), cuja presença na definição não é obrigatória, e unidades básicas de programação AO (aspectos). Não possuir classes, significa que um *framework* AO é composto somente por aspectos, sendo possível acontecer em situações especiais. Assim como em *frameworks* OO, *frameworks* AO podem ser definidos como um sistema semi-completo e reutilizável que poderá ser instanciado em situações que utilizam o domínio do *framework* utilizado.

A união de tecnologias de Padrões de Projeto e *frameworks*, apresentados no capítulo 3, e aspectos, resultam em economia de tempo, custos e recursos, mostrando-se ideal para áreas onde os requisitos mudem com certa rapidez, como é o caso do comércio eletrônico. O próximo capítulo trará uma aplicação prática destas tecnologias na área de comércio eletrônico: será apresentado o desenvolvimento do *framework* JCommerceNet.

5 JCOMMERCENET

Tecnologias utilizadas na área de desenvolvimento e geração de produtos, para lojas virtuais na Internet, estão em constante desenvolvimento. A utilização de *frameworks* para o desenvolvimento de lojas virtuais pode ser uma boa alternativa, pois os requisitos que passam por constantes mudanças, podem ser encarados como pontos de flexibilização (*hot-spots*), ao contrário dos requisitos que toda loja virtual deve atender, que são encarados como partes fixas (*frozen-spots*).

O processo de desenvolvimento de um *framework* depende do grau de experiência da organização no domínio do problema. Uma organização com maior experiência poderá adotar um processo mais avançado no desenvolvimento [17]. Com base na experiência adquirida no domínio do problema, optou-se neste trabalho por utilizar um processo baseado na análise do domínio.

Este processo caracteriza-se por:

- Analisar o domínio do problema para identificar e entender possíveis abstrações a partir do estudo de aplicações existentes, sendo possível somente se a organização possuir aplicações desenvolvidas;
- Identificar as abstrações e elementos em comum;
- Desenvolver o *framework* juntamente com uma aplicação de teste, modificando o *framework* quando necessário, revisando as aplicações anteriores para verificar se continuam funcionando.

5.1 Elicitação de Requisitos em Ambientes de Geração de Lojas Virtuais

Diversos sistemas de criação, manutenção e utilização de *Web sites* comerciais para a Internet foram observados com o objetivo de se fazer uma engenharia reversa e, a partir disso, definir as características pertinentes ao *kernel* do *framework* e aquelas que deveriam constituir os *hot-spots*.

Segundo SOMMERVILLE [58], a engenharia reversa é "o processo de analisar o *software* com o objetivo de recuperar seu projeto e sua especificação". Nela, o programa é analisado e as informações são extraídas dele, a fim de ajudar a documentar sua organização e funcionalidade. O programa permanece inalterado no processo de engenharia reversa. O código-fonte do *software* geralmente está disponível para a entrada do processo de engenharia reversa, mas se o código-fonte não estiver disponível, pode ser feita à análise com o código executável ou programa em execução.

Através da engenharia reversa, pretende-se fazer o levantamento das principais características de cada um dos produtos e serviços analisados. Características não só de funcionalidades que cada programa implementa ou que cada serviço disponibiliza, mas também as entidades que são criadas para modelar o problema junto à implementação.

No processo de modelagem do *framework* é importante a identificação das entidades que irão compor o *framework* para comércio eletrônico. O objetivo é conhecer as entidades e identificar

entidades coincidentes em diferentes produtos e serviços, de forma a construir um *framework* tão genérico quanto a união de todos os programas que fazem parte do estudo de *softwares* relacionados. As entidades utilizadas nos diferentes produtos e serviços solucionam um problema cada um com sua visão. Assim, o desenvolvimento de um *framework* mais genérico, permite uma visão mais abrangente do problema, com diferentes alternativas de solucioná-lo.

Os oito produtos analisados neste estudo podem ser divididos em dois grupos bem definidos. Os que representam a categoria-produto e os da categoria-serviço. Produto são todos aqueles *softwares* tradicionais, ou seja, existe um programa-produto, que deve ser adquirido e instalado numa máquina cliente para desenvolvimento. Na categoria de serviço, um *browser* conecta-se numa máquina remota onde está publicado um *software* que presta o serviço para resolver o problema.

A seguir, será feita uma breve descrição dos sistemas analisados. Também será apresentado um modelo feito para cada um dos sistemas, contendo as principais características verificadas durante a observação.

Um dos sistemas analisados foi o *AbleCommerce* <http://www.ablecommerce.com>. Este é um produto de uma empresa especializada em comércio eletrônico, totalmente customizável em tempo real. Permite facilmente a criação e modificação dos elementos da loja (cabeçalhos, rodapés, estilos, descrição dos produtos), utilizando editores padrão HTML, tais como: *Microsoft FrontPage* e *Macromedia DreamWeaver*. O conteúdo e o *layout* podem ser modificados desde que se possua um conhecimento em linguagem HTML e de Script. Os sub-módulos integrados incluem transporte, impostos, descontos, recuperação de pedidos, total do produto, opções do produto, vendas e relatórios. O núcleo do *AbleCommerce* é o *CommerceBuilder*®, um conjunto de módulos compilados que podem ser acessados via linguagem de Script. Esses módulos fornecem as funcionalidades necessárias ao comércio eletrônico, tais como: gerenciamento de produtos, processamento de pagamentos *on-line* e rastreamento de usuários. Possui um modelo de classes onde as principais entidades abordadas no modelo são: clientes, pedidos, departamentos, produtos, categorias, promoções. Seu modelo de classes pode ser visto na figura 20:

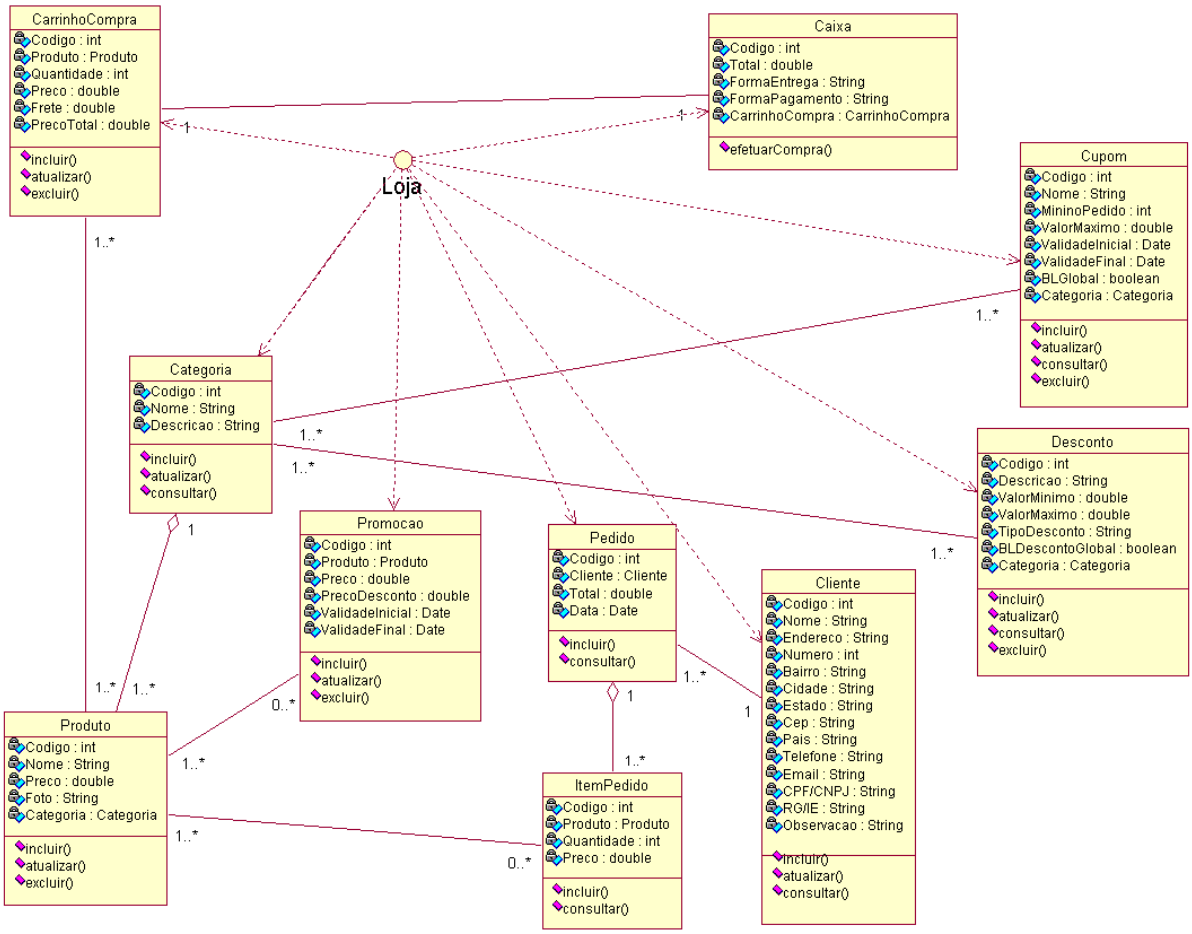


Figura 20 - Modelo simplificado do *AbleCommerce*

O *2BuyNet* <http://www.2buynet.com.br> é um serviço para geração automática de lojas virtuais desenvolvido no laboratório de Engenharia de *Software* da Pontifica Universidade Católica do Rio de Janeiro (PUCRJ), o qual surgiu de uma análise de diversas soluções existentes. O *2BuyNet* pressupõe uma estrutura hierárquica dividida em departamentos e sub-departamentos para organização de uma loja. As principais entidades previstas pelo *2BuyNet* são: produtos, departamentos, clientes, pedidos, carrinho de compras e o idioma. Seu mecanismo não implementa nada relativo a promoções, mas indexa todo o conteúdo da loja quanto ao idioma. Os produtos são totalmente customizáveis, exigindo-se apenas um mínimo de atributos para caracterizar um produto, esses atributos são: nome, descrição, foto e preço. Seu modelo pode ser visto na figura 21:

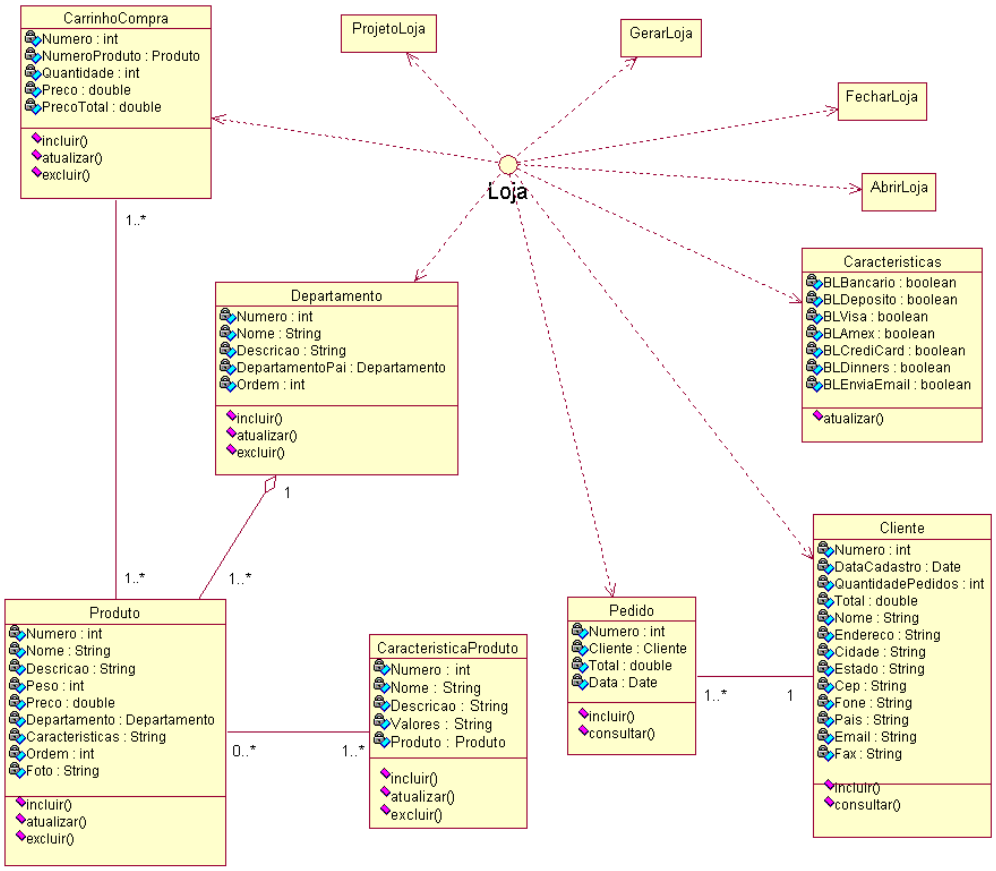


Figura 21 - Modelo simplificado do *2BuyNet*

Qualquer especificação adicional pode ser configurada pelo *software*, que implementa uma meta base de dados para representar tais configurações dos produtos. Um exemplo seria uma loja de vestuário, onde os produtos não teriam apenas nome, descrição, foto e preço, mas também tamanho e cor. Esses dois atributos: tamanho e cor, foram definidos pelo cliente e poderá ser atribuído a todos os produtos da loja ou apenas a alguns deles. É possível traduzir e publicar uma loja sem a necessidade de reconstruí-la. Por ser um serviço, o *2BuyNet* não disponibiliza os códigos-fontes das páginas HTML para o cliente. No processo de geração de uma loja, esta é gerada e publicada dentro do próprio servidor de *Web* do *2BuyNet*. O processo é dividido em duas etapas, conforme descrito anteriormente, nesse tipo de procedimento. A primeira etapa consiste na geração e publicação temporária e só posteriormente, a publicação na Internet.

O viaSebrae <http://www.viasebrae.com.br> é um *site* do Sistema Brasileiro de Apoio às Micro e Pequenas Empresas (SEBRAE), que permite às pequenas empresas comprarem e venderem seus produtos de forma vantajosa e com baixos custos operacionais. Tratando-se de vendas de produtos, o *site* oferece a modalidade de comércio eletrônico entre Empresa-consumidor (B2C), que é a venda direta aos consumidores finais, que acessam livremente o *site*. O viaSebrae desenvolve, hospeda, dá suporte técnico e orientação comercial para as empresas virtuais que se associam à solução. Para adquirir o serviço, a empresa deve solicitar uma proposta de orçamento e selecionar que deseja vender produtos na Internet. As entidades identificadas no modelo disponibilizado pelo *site* para

desenvolvimento de uma loja virtual são: categorias, lista de produtos por categoria, carrinho de compras e ofertas. O *site* oferece ainda a possibilidade de pesquisar produtos na loja virtual, alteração e exclusão de produtos no carrinho de compras e, quando terminadas as compras, optar por uma das formas de pagamento disponíveis e preencher os campos destinados à informação quanto ao local de entrega. Também há a possibilidade de gravar o pedido de compra para eventuais compras futuras, que poderá ser utilizado na próxima vez que o cliente visitar o *site*. Seu modelo pode ser visto na figura 22:

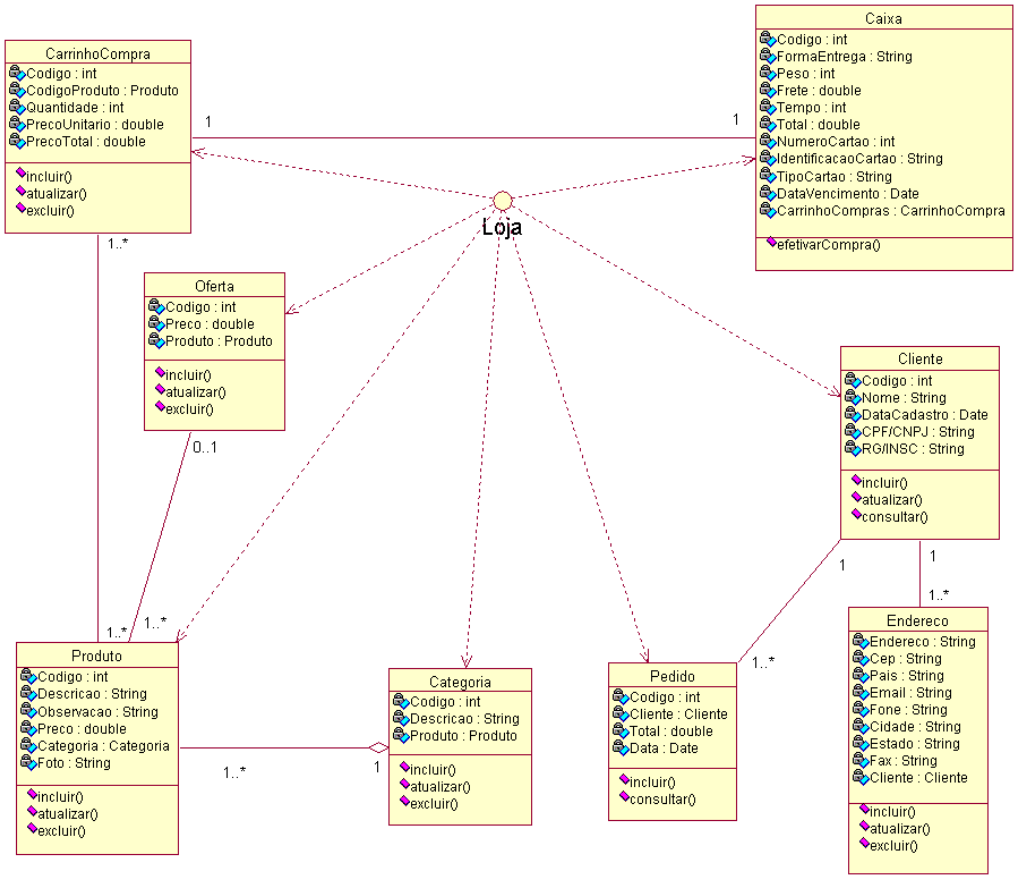


Figura 22 - Modelo simplificado do viaSebrae

O *Web Sites Factory* <http://www.websites.com.br> é um *site* de uma empresa que desenvolve soluções para ambiente *Web*. Desenvolve soluções de lojas virtuais para venda ao consumidor, oferecendo diferentes opções conforme a necessidade. São três soluções oferecidas: solução 3.0 – Quiosque *Web Sites*: Alugue sua loja; solução 5.0 – Loja *Web Sites*: Compre sua loja; solução 9.0 – *SuperStore Web Sites*: Compre sua Super Loja. As entidades identificadas nessas soluções são: produtos, promoções, clientes, pedidos, carrinho de compras e usuário. O sistema de compras *on-line* oferece um sistema de busca utilizando o banco de dados, um formulário para cadastramento de usuário e opções de pagamento em boleto bancário e cartão de crédito. Seu modelo pode ser visto na figura 23:

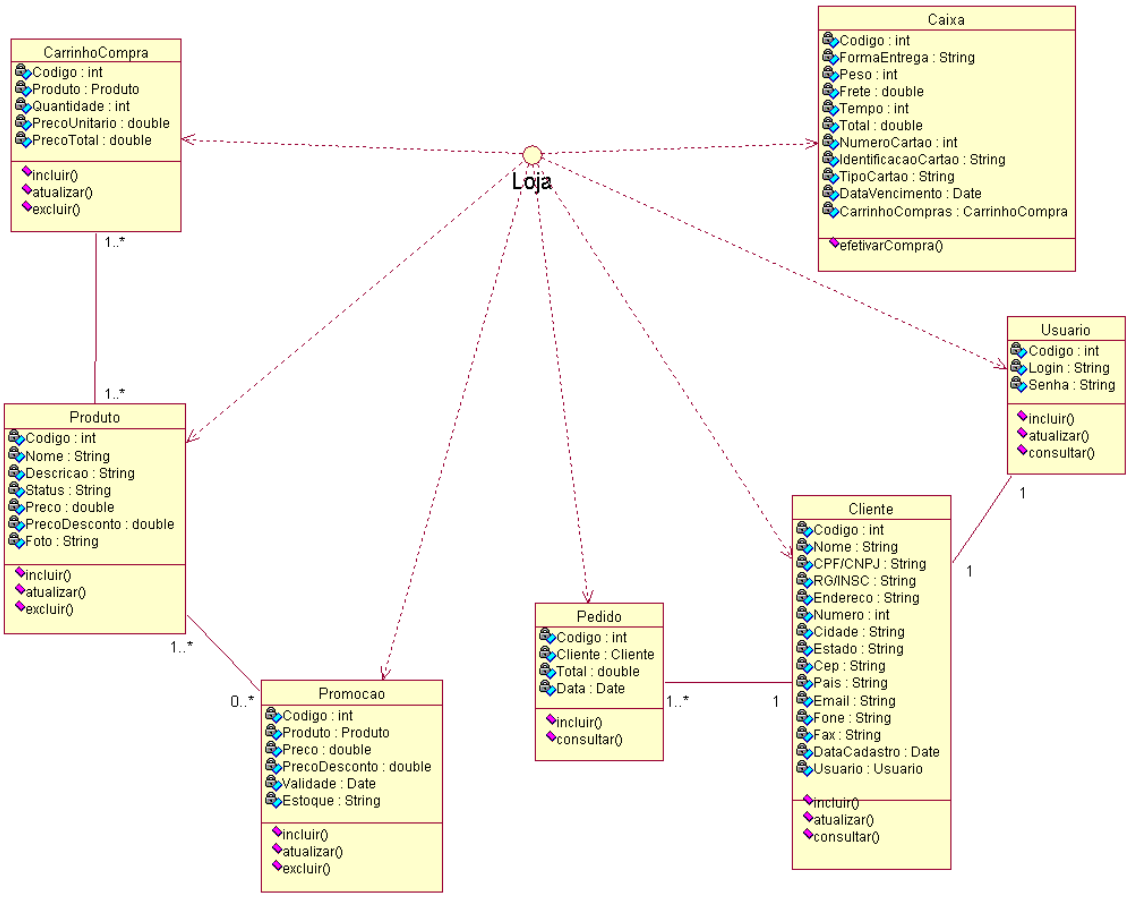


Figura 23 - Modelo simplificado do *WebSiteFactory*

O AMX Shopping Virtual <http://www.amx.com.br> é um portal que oferece aos clientes a possibilidade de divulgar os produtos através de um sistema de administração de lojas virtuais, feito *on-line* diretamente pelo cliente. A administração e o cadastro dos produtos são feitos pelo cliente, e a configuração da loja é feita pela equipe técnica do AMX, de acordo com o plano de loja comprado pelo cliente. Os planos variam dos mais simples, com número limite de produtos que podem ser cadastrados, até planos com um número de 1000 produtos cadastrados. As principais entidades levantadas no modelo analisado são: produtos, categorias, pedidos, clientes e carrinho de compras. Seu modelo de classes pode ser visto na figura 24:

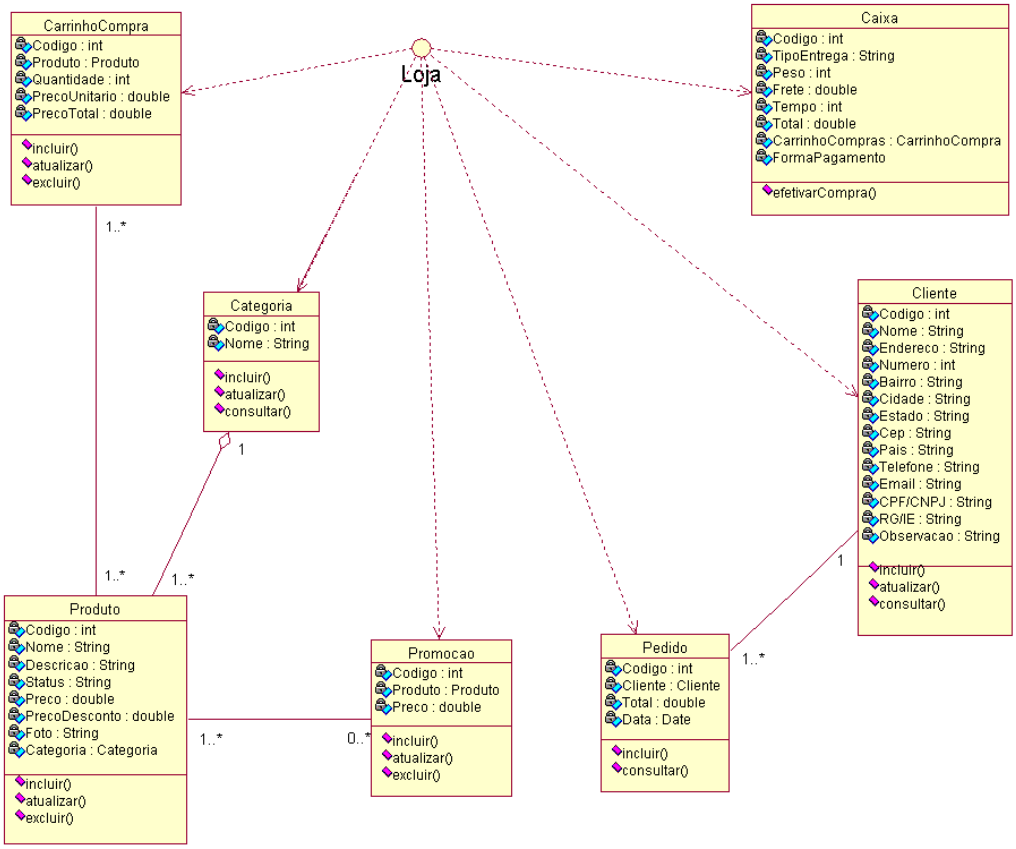


Figura 24 - Modelo simplificado do AMX Shopping Virtual

A Gigamax <http://www.gigamax.com.br> é uma empresa especializada no desenvolvimento de *softwares* e soluções aplicadas para negócios na Internet. Possui um serviço Loja Virtual que oferece ao cliente uma estrutura de loja segura com características e recursos solicitados pelo cliente. O controle da loja é feito pelo cliente através do acesso *on-line*, não havendo limite quanto ao número de produtos, nem cobrança sobre transações de venda. O serviço permite configurações de aparência, fontes, cores, imagens, mensagens e rodapés das páginas da loja virtual. Opções dos produtos, tais como: cor, tamanho, são configuráveis tanto na loja quanto no carrinho de compras. Possui um controle de estoque que notifica o lojista quando o estoque está baixo e retira o produto de exposição quando o estoque do produto acaba. As principais entidades são: carrinho de compras, produtos, características do produto, características da loja, promoções, produtos em destaque, cliente, pedido. Seu modelo de classes pode ser visto na figura 25:

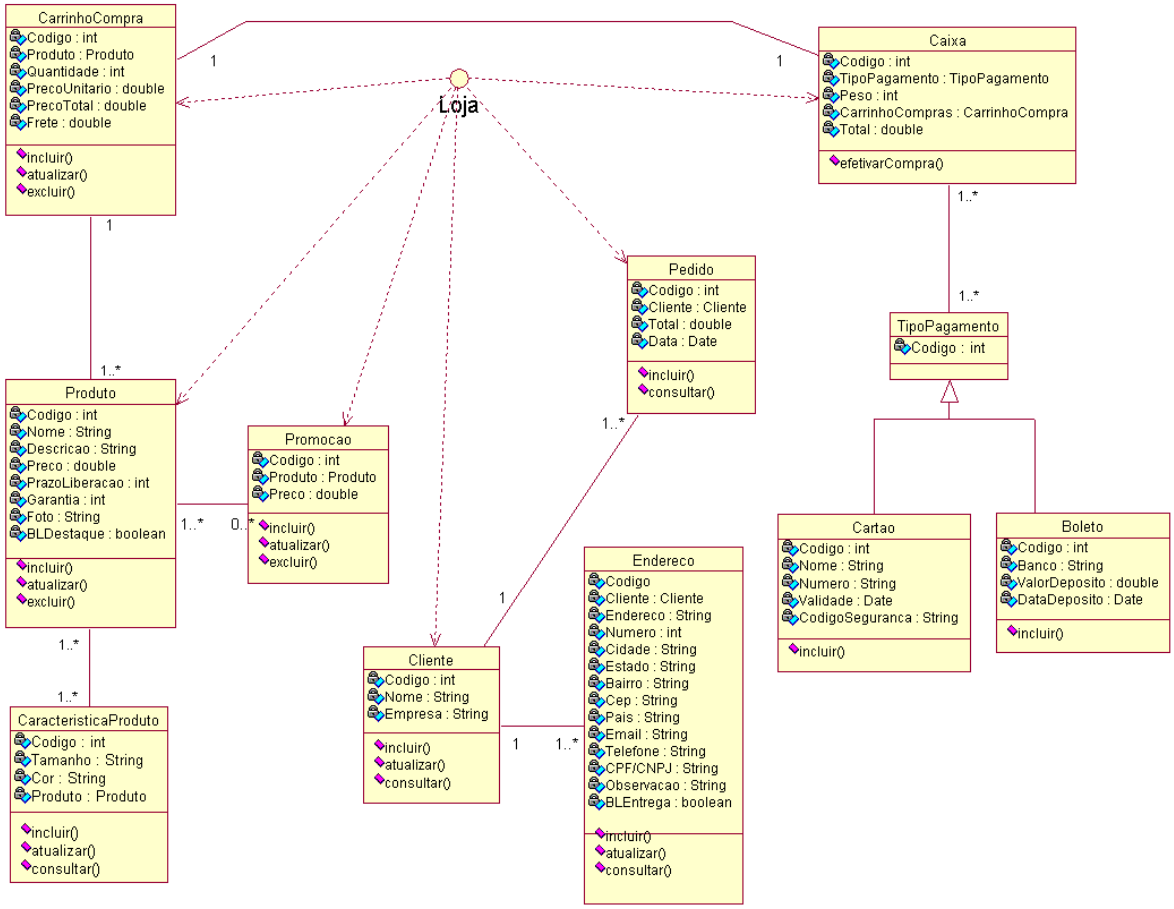


Figura 25 - Modelo simplificado do Gigamax Loja Virtual

A LojcommStore <http://www.lojcomm.com.br> é uma solução de comércio eletrônico desenvolvida pela Lojcomm para desenvolvimento de lojas virtuais. Oferece *layout* personalizado, não apenas na mudança de imagens e cores. Acesso ao menu em até três níveis (departamento, categoria e sub-categoria). Possui opção para mostra de produtos em destaque. Oferece meios de consulta para o cliente saber o andamento do(s) pedido(s), e o carrinho de compras é opcional na loja. As principais entidades observadas na análise são: produtos, promoções, pedidos, clientes, carrinho de compras. Seu modelo de classes pode ser visto na figura 26:

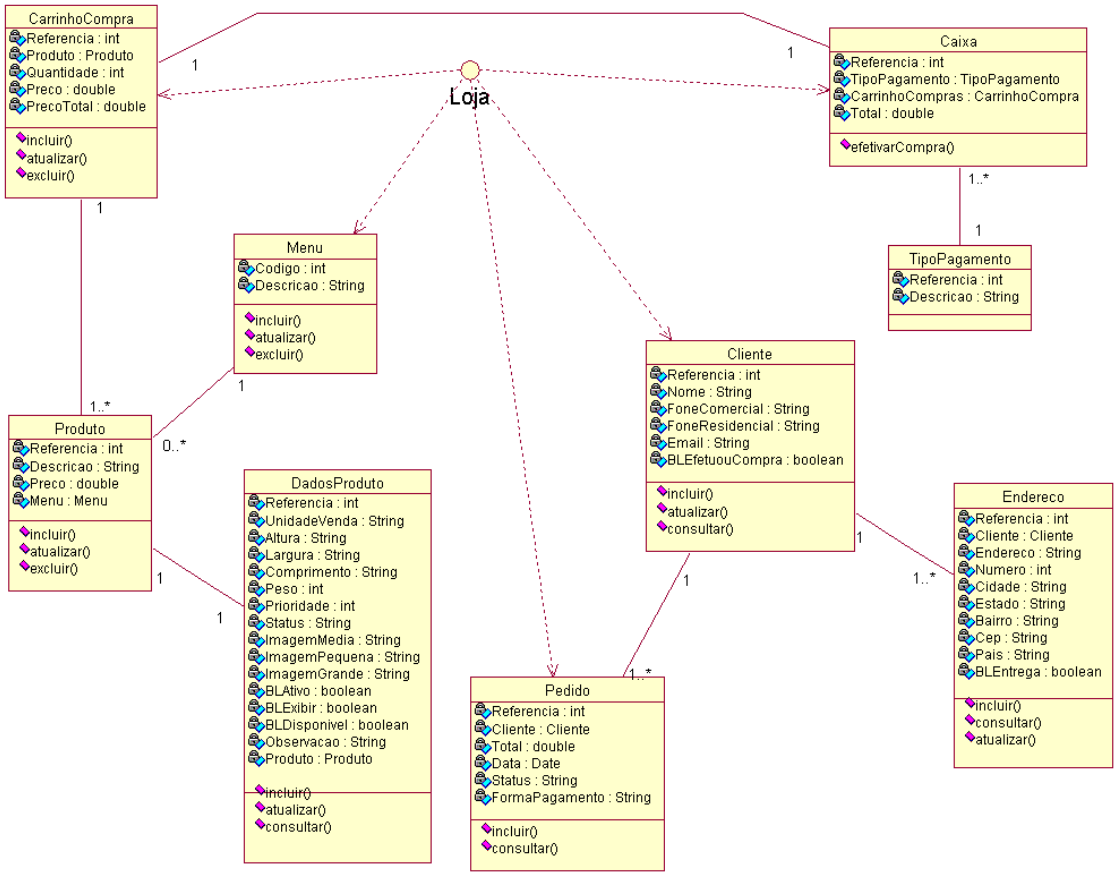


Figura 26 - Modelo simplificado do Lojcomm

O 001SHOP <http://www.001shop.com.br> é um produto desenvolvido pela empresa StudioVR. Para comprar o produto, o cliente necessita escolher um dos planos oferecidos pelo *site*, mas independente do plano, o produto oferece recursos básicos que todos os planos cobrem sem qualquer custo adicional. O produto oferece diferentes formas de pagamento, tais como: cartões de crédito, boletos bancários e operadoras de crédito. Recursos de vendas, tais como: lista de desejos, carrinho de compras, produtos, acompanhamento de pedidos, controle de estoque, gerenciamento de clientes, sistema de administração da loja via Internet, entre outros. Por se tratar de um produto, não é fornecido código-fonte do sistema para fins de customização da loja. Existe um limite máximo de 5.000 (cinco mil) produtos que podem ser cadastrados por loja. Seu modelo de classes pode ser visto na figura 27:

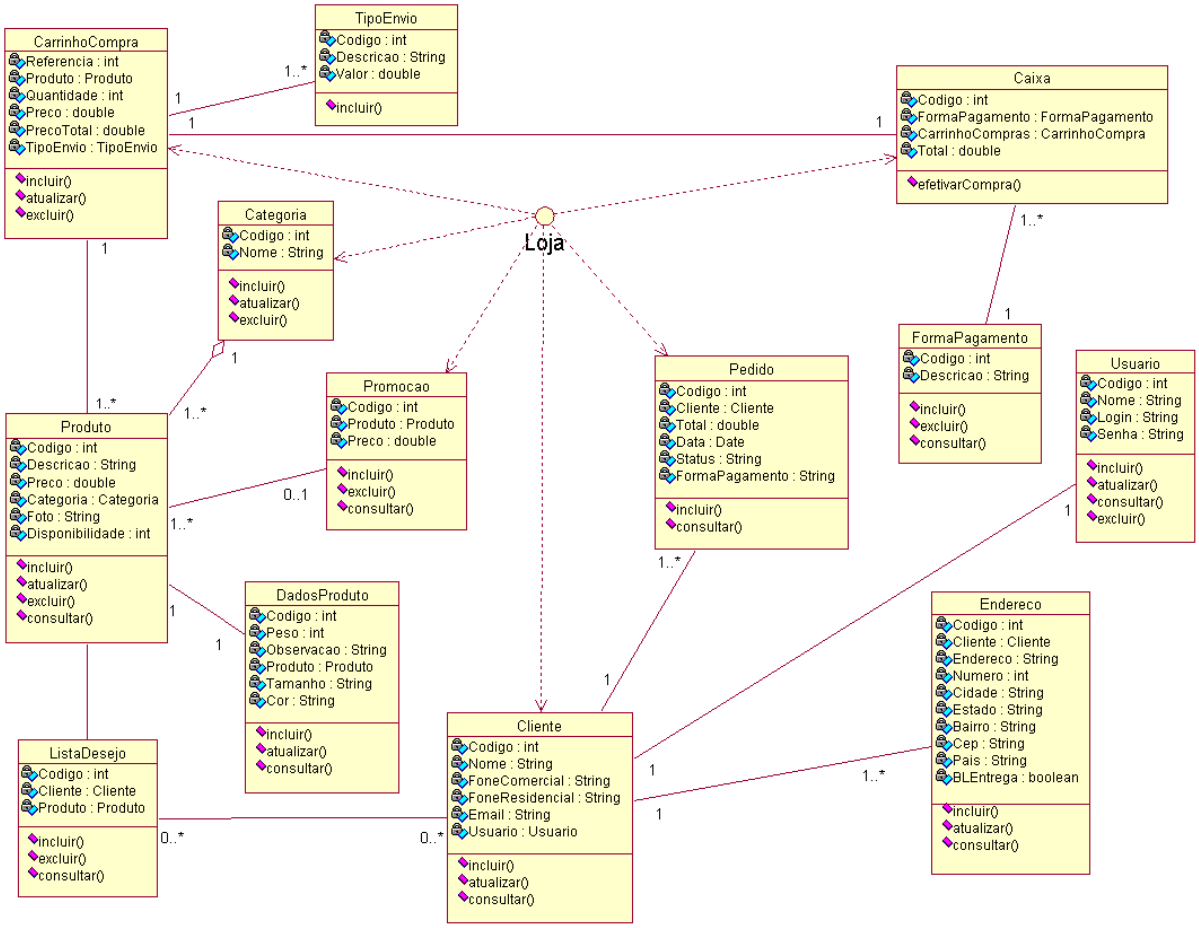


Figura 27 - Modelo simplificado do 001Shop

É possível perceber pontos em comum nos produtos e serviços descritos, bem como características que não estão presentes em todos eles, ou que se apresentam em alguns de formas diferentes. Esses pontos em comum devem fazer parte do *kernel* do *framework*, enquanto os *hot-spots* serão constituídos pelas características diferenciadas. O processo de engenharia reversa realizado resultou no desenvolvimento de um primeiro modelo, que serviu de base para a modelagem do *framework*.

O diagrama representado na figura 28 ilustra o processo de engenharia reversa realizado.

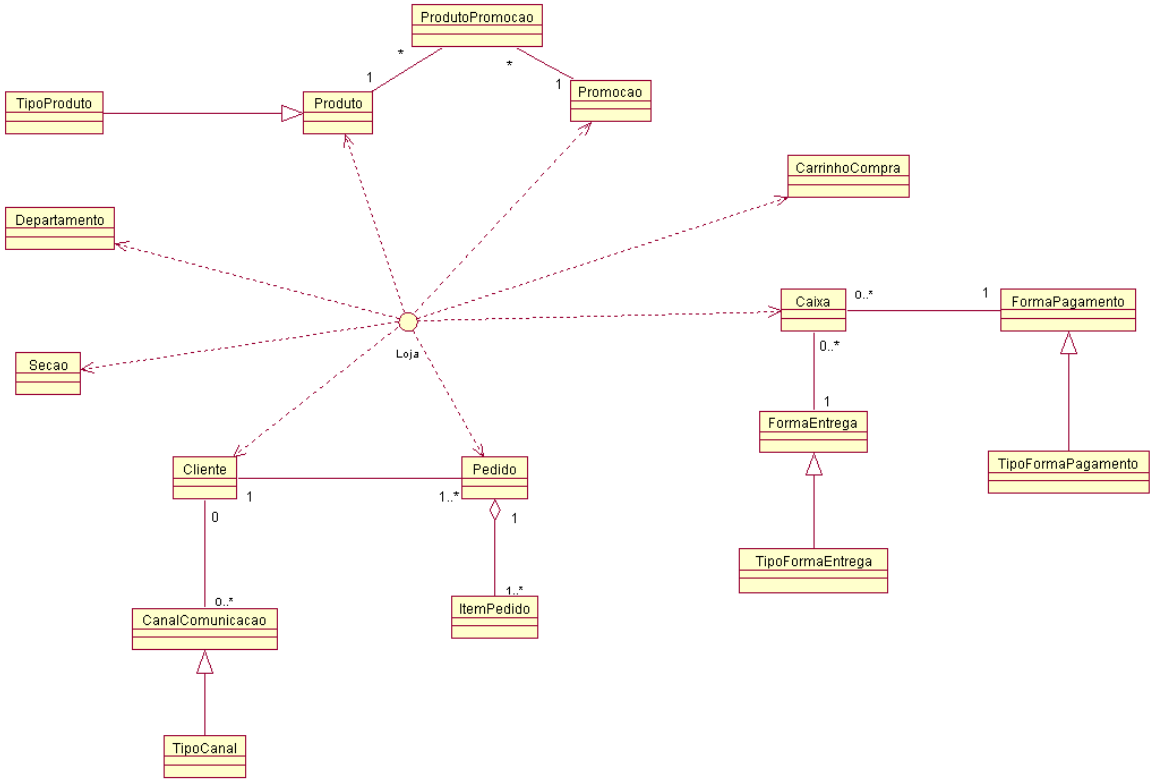


Figura 28 - Modelo genérico dos produtos e serviços analisados

Os produtos e serviços descritos constituem as ferramentas analisadas, utilizadas neste trabalho para melhor ilustrar o processo de engenharia reversa e definição dos requisitos necessários para o desenvolvimento do *framework*.

5.2 *Hot-spots* dos Ambientes Analisados

Com base nas informações obtidas, foi possível definir as classes do *kernel* e dos *hot-spots* do *framework*.

Como pontos a serem flexibilizados de um sistema de geração de scripts, temos:

- Definição de tipos de produtos: cada produto comercializado na loja possui características comuns com alguns produtos e diferentes em relação a outros, o que se faz necessário à implementação de tipos de produtos, conforme a necessidade. Entende-se por tipos de produtos, como sendo perfumes, livros, dvds, produtos de informática, filmes etc. Um produto do tipo livro possui características pertinentes a um livro, tais como: autor, editora, ISBN. Um produto do tipo Dvd pode ser categorizado, por exemplo, em dois tipos: DVD/Show que possue características como direção, produção artística, músicas, bônus, etc e DVD/Filmes possuindo características como ator, diretor, distribuidora, título original, legenda, etc.

- Definição de departamentos: uma loja pode ou não estar dividida em departamentos, isso pode ser flexível conforme a definição da loja. Pode-se tomar como exemplo de uma loja dividida em departamentos, uma loja que comercialize CD-Rom educacionais e de treinamento. Essa loja poderia ter departamentos como Informática e Infantil.

- Definição de seções: produtos comercializados numa loja podem estar agrupados em diferentes seções. Tomando como base o exemplo anterior de tipos de departamentos que uma loja pode conter, poder-se-ia subdividir o departamento de Informática em Aplicativos, *Design* e Internet, assim como o departamento Infantil em Alfabetização e Percepção Visual.

- Definição de canais de comunicação: um canal de comunicação é um meio de comunicação entre a loja e o cliente. Quando um cliente efetua um pedido, pode ser solicitada a confirmação do envio de uma mensagem automática, referente à efetivação da compra ou mensagens de propagandas publicitárias sobre ofertas ou novos produtos que a loja está oferecendo. Estas mensagens podem ser enviadas via *e-mail*, celular ou outro meio de comunicação.

- Definição do carrinho de compras: o carrinho de compras é uma peça fundamental que toda loja virtual deve implementar. É através dele que os clientes vão depositando seus itens de compra, de forma que quando desejam realmente efetuar a compra, devem encaminhá-los ao caixa para que possa somar os itens do carrinho de compras e proceder com a finalização da compra. O carrinho de compras é um utilitário que pode oferecer opções tais como: adicionar produtos, remover produtos, atualizar produto, limpar carrinho de compras, armazenar produtos para compras futuras, etc.

- Definição do caixa: o caixa oferece ao cliente a finalização da compra através do somatório de todos os produtos contidos no carrinho de compras. Ao ir para o caixa de uma loja virtual, o cliente pode selecionar a forma de pagamento da compra, indicar o endereço de entrega, escolher a forma de entrega dos produtos, informar seus dados para que possa tornar-se cliente efetivo da loja, efetivar o pedido para que a loja mantenha um histórico de todas as compras realizadas, etc.

- Definição de formas de pagamento: diversas formas de pagamento podem ser oferecidas por uma loja virtual, tais como: boleto, cartão de crédito, doc bancário, etc. Essas formas de pagamento podem ser configuradas pela loja virtual de acordo com as opções de crédito que deseja disponibilizar aos seus clientes.

- Definição de formas de entrega: formas de entrega dos produtos podem ser configuradas pela loja virtual, tais como: sedex, transportadora, entrega normal, etc. As formas de entrega dependem das parcerias que a loja virtual formalizar com prestadores de serviços.

- Definição da conexão com banco de dados: o banco de dados deve ser conhecido para acesso às informações, portanto este pode ser configurado através da definição do tipo de conexão que é desejada. A conexão para acesso aos dados pode ser efetuada, por exemplo, via JDBC, ou via JNDI.

- Definição do script para acesso a nova *sequence* no banco de dados: cada banco de dados estabelece um script para busca da *sequence*, assim, o script deve ser implementado na instanciação do *framework*

A seguir será apresentado o modelo de classes do *framework* proposto, levando-se em consideração os aspectos já elicitados dos diversos produtos e serviços analisados. O modelo será visto e apresentado em partes, de forma a proporcionar um melhor entendimento. Nesta descrição será destacado o *kernel* e os *hot-spots* do ambiente.

5.3 Descrição do *Framework*

Na figura 29 é apresentado em um tom mais claro as classes referentes ao *kernel* do *framework*, e num tom mais escuro são apresentados os *hot-spots*. Conforme definição dos pontos de flexibilização, as classes Secao, Departamento, CarrinhoCompra, Caixa, FormaPagamento, FormaEntrega, CanalComunicacao e TipoProduto representam os *hot-spots* do *framework*. As classes Produto, Cliente, Promoção, ProdutoPromocao, Pedido e ItemPedido representam o conceito de *kernel* do *framework*.

O modelo de classes é representado através do uso da notação padrão UML (Rumbaugh et. al) [14]. Para a representação das classes dos *hot-spots* do *framework*, optou-se por utilizar a notação UML-F (Fontoura et. al) [24], que é utilizada para modelagem de *frameworks*. UML-F utiliza os mecanismos de extensão básicos da UML para definir novas construções para modelar todos os aspectos relevantes de projetos de *frameworks* [25]. A *tag* {*extensible*} utilizada significa que a classe de interface depende da instanciação do *framework*, e novos métodos devem ser definidos para estender a funcionalidade da classe. A *tag* {*apply-class*} descreve que novas classes de aplicação devem ser definidas durante a instanciação do *framework*. A *tag* {*incomplete*} indica que novas classes devem ser adicionadas para um relacionamento de herança.

Na figura 29 é apresentada a relação entre os produtos (classe Produto) e os tipos de produtos que a loja pode comercializar (classe TipoProduto). A classe TipoProduto pode especializar diferentes tipos de produtos, para tanto utilizou-se o relacionamento de herança entre a classe Produto e TipoProduto, para que as novas classes possam herdar os atributos e comportamentos dessa classe existente, assim pode-se adicionar atributos e comportamentos ou anular comportamentos de superclasse para personalizar a classe especializada, por exemplo: todo produto possui atributos em comum com outros produtos, porém produtos especializados diferem em características, portanto, essas diferentes características são identificadas e informadas nas classes especializadas. Esse mecanismo de herança é aplicado também para as classes CanalComunicacao e TipoCanal, FormaEntrega e TipoFormaEntrega, FormaPagamento e TipoFormaPagamento.

As classes CarrinhoCompra, Caixa, Secao e Departamento devem ser especializadas na definição do domínio de aplicação do *framework*. Por serem classes abstratas devem ser implementadas na ocasião de instanciação do *framework*. Tirando a classe CarrinhoCompra, as demais serão classes de persistência, onde os atributos definidos para o objeto devem refletir a tabela mapeada no banco de dados. Somente atributos e métodos *get()* e *set()* devem ser implementados nessas classes, já que a implementação da camada de persistência será feita de maneira a não espalhar código de negócio em classes de persistência. Os aspectos AspectTipoProduto, AspectCarrinhoCompra, AspectCaixa, AspectSecao, AspectDepartamento, AspectFormaPagamento, AspectTipoFormaPagamento, AspectFormaEntrega e AspectTipoFormaEntrega são aspectos abstratos que devem ser implementados na instanciação do *framework*. Estes irão implementar o mapeamento dos atributos das tabelas do banco de dados em relação aos tipos dos atributos dos objetos.

As classes definidas no *kernel* do *framework* estão relacionadas a um aspecto correspondente, que serão apresentados na figura 33.

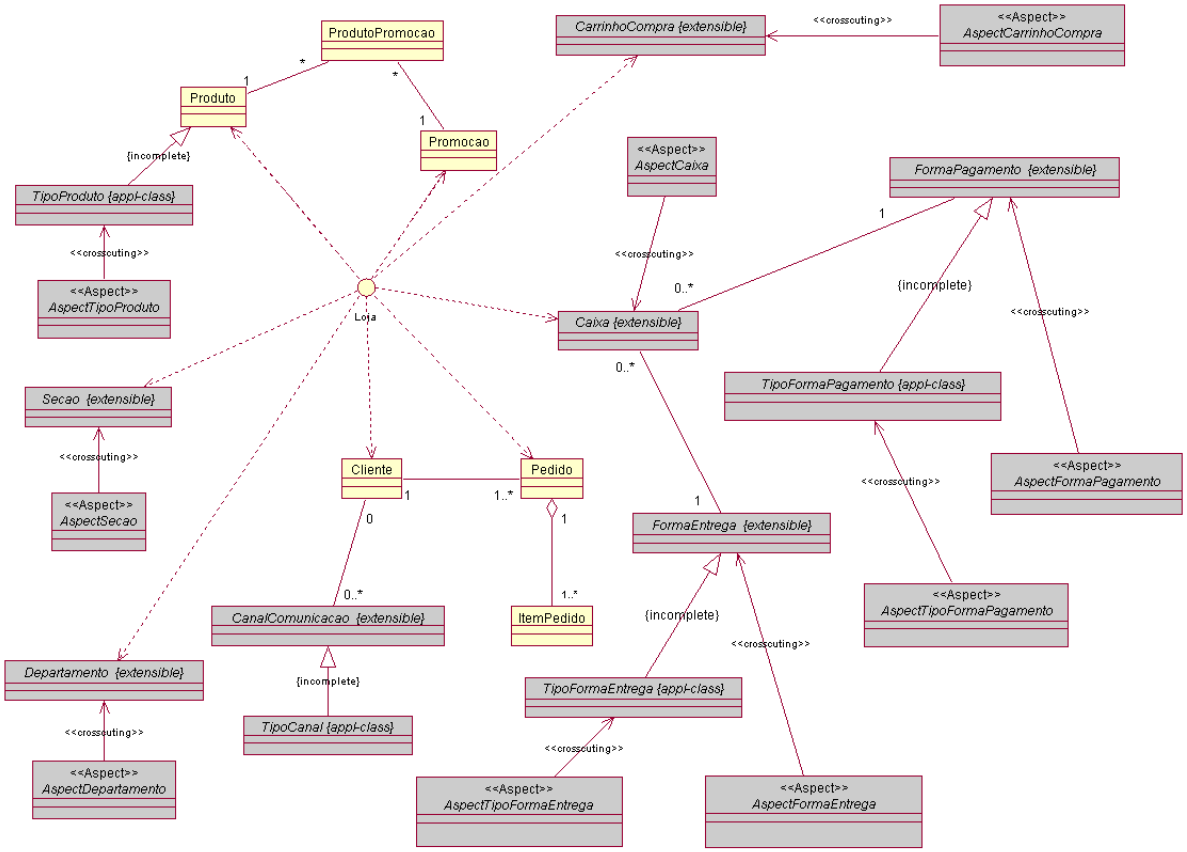


Figura 29 - Modelo de classes do *framework* com identificação dos *hot-spots*

Na figura 30 é apresentada a relação entre as classes de aplicação disponíveis no *framework*. Foi utilizado o padrão de projeto *Factory Method* [28] definindo-se uma interface para criar objetos de forma que as subclasses façam a instanciação. Esse padrão de criação permite que uma classe delegue a responsabilidade de instanciação às subclasses. Isso permite que qualquer classe de aplicação adicionada ao *framework* estenda a classe *PojoObject* para que seja responsável pela criação do objeto, não forçando colocar em código as classes concretas que serão instanciadas. Com a utilização da classe *PojoObject*, todas as classes persistentes da aplicação do *framework* devem estender esta classe que relaciona-se por associação com a classe *MappingManager*, onde os métodos responsáveis pelo mapeamento de atributos foram implementados.

A classe *MappingManager*, implementa o sub-padrão Métodos de Mapeamento de Atributos (*Atributte Mapping Methods*) do padrão Camada de Persistência (*Persistence Layer*) [62], responsável por mapear os atributos de um objeto e as suas respectivas colunas da tabela do banco de dados. Esse mapeamento é feito através da chamada ao método abstrato mapear() declarado na interface *PojoObject*. Cada classe que estender a classe *PojoObject*, deve implementar esse método.

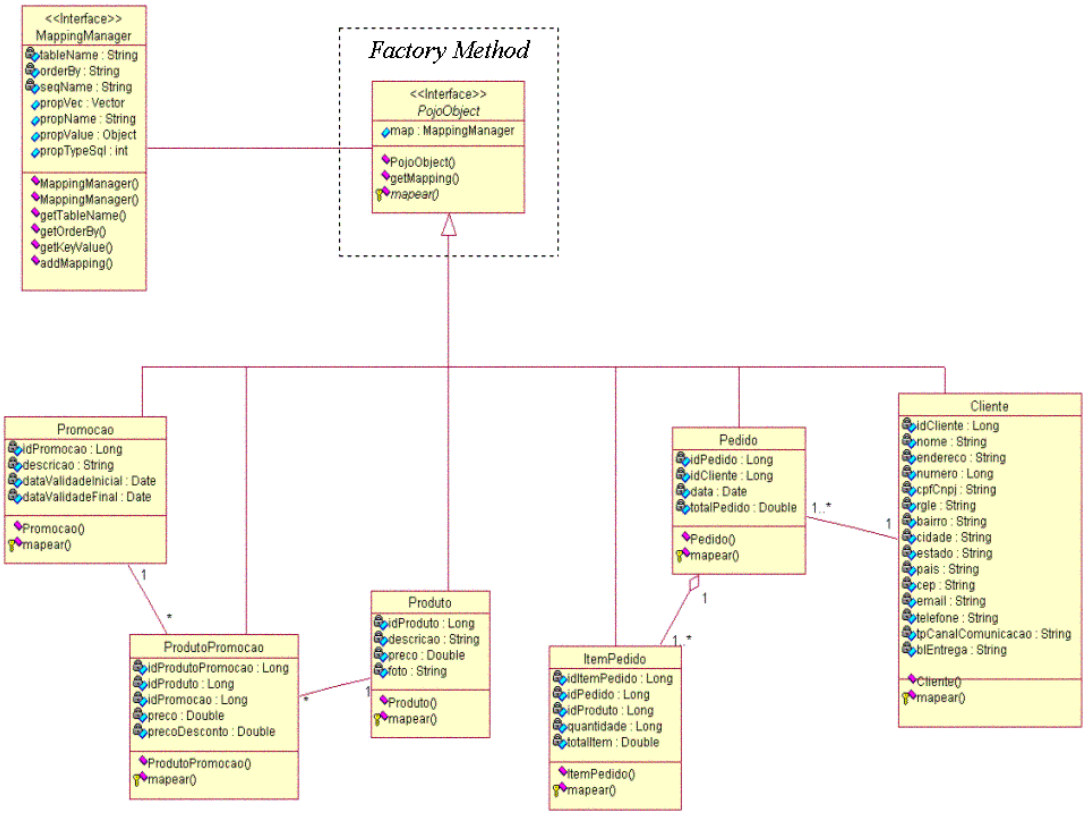


Figura 30 - Modelo de classes do *framework* da camada de aplicação

Na figura 31 é apresentado o diagrama de classes que será responsável pela camada de persistência do *framework*. As classes EJBs (*Enterprise Java Beans*) [5] [34] são utilizadas para chamadas de métodos que delegarão a execução dos eventos a uma classe de controle chamada *TableManager*. Foi utilizado o padrão de projeto *Façade* [28] para definir uma classe centralizadora das solicitações a objetos apropriados no sistema. Esse padrão estrutural de objeto oferece uma interface única para um conjunto de interfaces de um subsistema. Essa interface de nível mais elevado torna o subsistema mais fácil de usar, reduzindo complexidade e minimizando dependências [5].

Na classe *TableManager* são declarados os métodos conectarDB() e closeDB(), que implementam o sub-padrão Gerenciador de Transação (*Transaction Manager*) do padrão Camada de Persistência (*Persistence Layer*) [62], responsáveis por controlar as transações com o banco de dados, implementando mecanismos de *Commit* e *Rollback*, que melhoram o controle e a segurança dos dados.

No conjunto de classes Incluir, Alterar, Excluir, Consultar, foi implementado o sub-padrão CRUD do padrão Camada de Persistência (*Persistence Layer*) [62] e mais duas classes utilitárias ConsultarNivel e ExcluirNivel. Cada classe implementa uma operação do sub-padrão, fazendo com que o controle de manutenção seja delegado para a classe correspondente.

Para a geração dos scripts de código SQL, foi utilizada uma classe abstrata chamada MontaScriptSQL, que implementa em cada um dos seus métodos, o sub-padrão Descrição de Código SQL do padrão Camada de Persistência (*Persistence Layer*) [62]. A utilização desse sub-

padrão define a geração das cláusulas SQL para cada objeto persistente da aplicação, utilizadas nas chamadas aos métodos implementados pelo sub-padrão CRUD [62]. Nesta classe, o método abstrato montaScriptPK() deve ser implementado na instanciação do *framework* inserindo a cláusula SQL que fará a busca pela nova *sequence* da tabela, que deverá ser enviada por parâmetro. Este método implementa o *hot-spots* de definição do script para a nova *sequence* da tabela na base de dados.

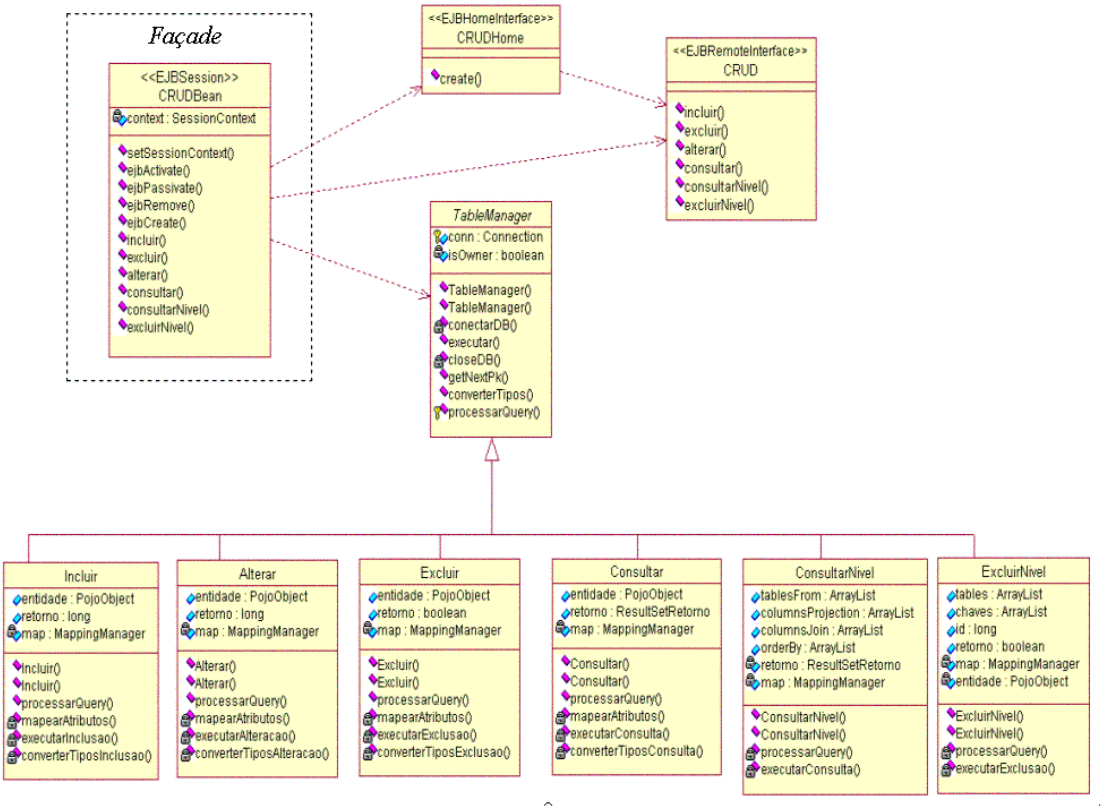


Figura 31 - Modelo de classes do *framework* para a camada de persistência

A figura 32 apresenta como a modelagem da conexão com o banco de dados foi implementada. A classe abstrata *TableManager* na implementação do método conectarDB(), faz referência ao método *getConnection()*, que está declarado como abstrato na classe abstrata *ConnectionManager*, e deverá ser implementado na instanciação do *framework*. Este método implementa o *hot-spots* de definição da conexão com o banco de dados. A classe *ConnectionManager* implementa o sub-padrão Gerenciador de Conexão (*Connection Manager*) do padrão Camada de Persistência (*Persistence Layer*) [62], responsável por efetuar a conexão do sistema com a base de dados, garantindo sua manutenção. A utilização desse sub-padrão traz alguns benefícios, tais como: conexão para todos os objetos persistidos, uma única classe para fazer manutenção caso as informações da base de dados venham a ser alteradas.

A classe *ConnectionManager* estabelece uma conexão com a base de dados baseada nas informações de requisição da classe *TableManager*. Quando *TableManager* invoca o método *getConnection()*, o aspecto *AspectConnection* intercepta a chamada ao método e busca a informação da conexão na classe *ConnectionManager* através do método *TableManager.getConnection()* que

deverá ser implementado na instanciação do *framework*. O uso de aspectos na conexão a base de dados centraliza as chamadas ao método *getConnection()*, não precisando a classe *TableManager* acessá-lo. Isso beneficia a manutenção do sistema, caso ocorram alterações nos métodos da classe *ConnectionManager*. Na representação do método *getConnection()* utilizou-se a notação UML-F [24]. A tag {*variable, dynamic*} indica que o método é variável, ou seja, sua implementação varia para cada aplicação instanciada, e {*dynamic*} complementa a definição do método de variação, indicando que uma instância é requerida em tempo de execução.

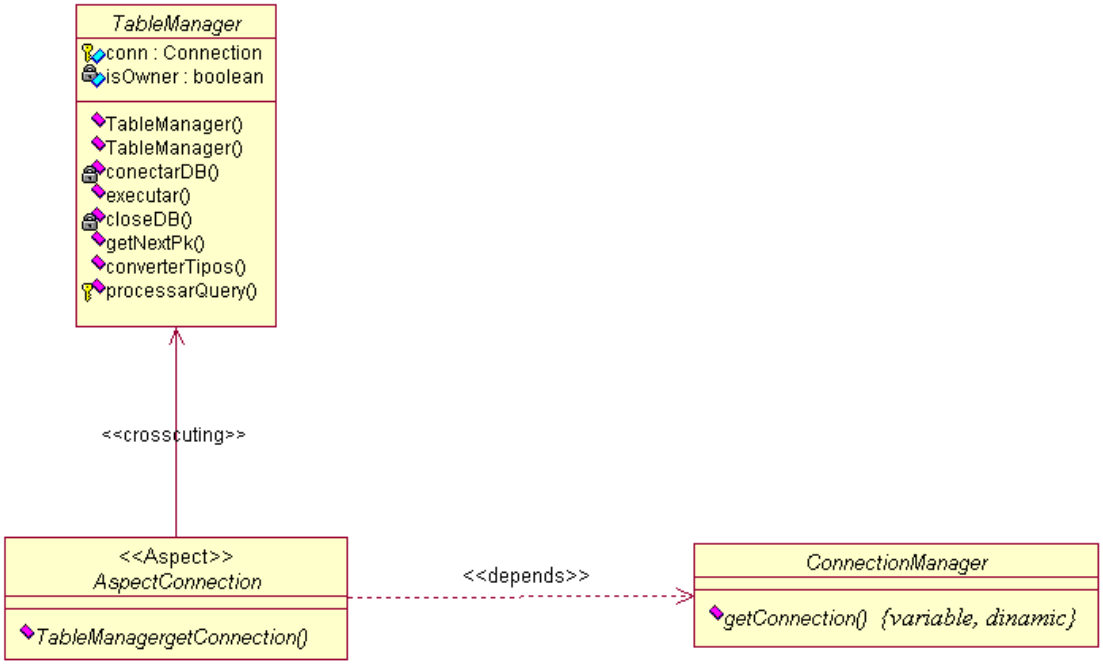


Figura 32 - Modelo de classes da conexão com banco de dados

A figura 33 apresenta o modelo de classes e aspectos. Este modelo consiste em apresentar como a modelagem do *framework* baseado em aspectos foi implementada. Para adaptar o uso de aspectos ao *framework*, foi necessário criar um aspecto para cada classe persistente da aplicação do *framework*. Pode-se tomar como exemplo o aspecto AspectProduto, que possui comportamento específico para a classe Produto, pois deve definir valores de persistência para essa classe, tais como: nome da tabela, nome da *sequence*, atributo de ordenação, e todos os atributos da tabela no banco de dados. O ponto de corte *doConstrutor()* desse aspecto possui como tarefa interceptar chamadas do construtor da classe Produto e definir valores para os atributos de mapeamento dessa classe. O ponto de corte *doMapping()* desse aspecto possui como tarefa interceptar chamadas do método mapear() da classe Produto e definir valores para os atributos de persistência dessa classe.

Na implementação de aspectos para as classes persistentes da aplicação, cada aspecto implementa o sub-padrão Conversão de Tipos (*Type Conversion*) do padrão Camada de Persistência (*Persistence Layer*) [62]. Nesta implementação, o ponto de corte *doMapping* implementa o método mapear() da classe persistente, atribuindo para cada coluna da tabela da base de dados, o tipo apropriado de objetos.

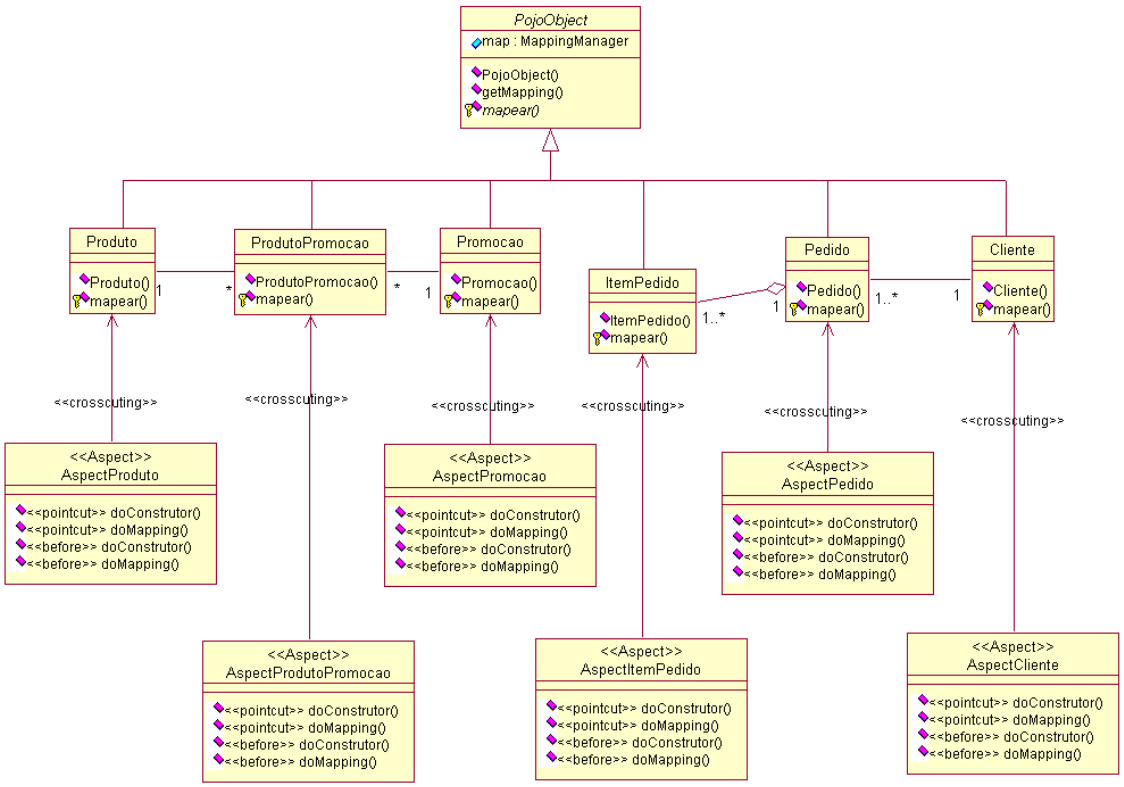


Figura 33 - Modelo de classes da aplicação e aspectos

Todos os *hot-spots* do modelo possuem métodos para sua criação e manipulação, e por serem classes abstratas devem ser implementados na ocasião da instanciação do *framework*.

A figura 34, na página a seguir, apresenta uma visão geral do modelo do *framework*. Esse modelo separa o *framework* em duas partes: uma formada pela camada de persistência, onde são implementadas as classes responsáveis pelo gerenciamento do acesso e manipulação dos dados no banco de dados, e uma outra formada pela camada de aplicação, onde contém a implementação das classes que fazem parte do *kernel* do *framework*.

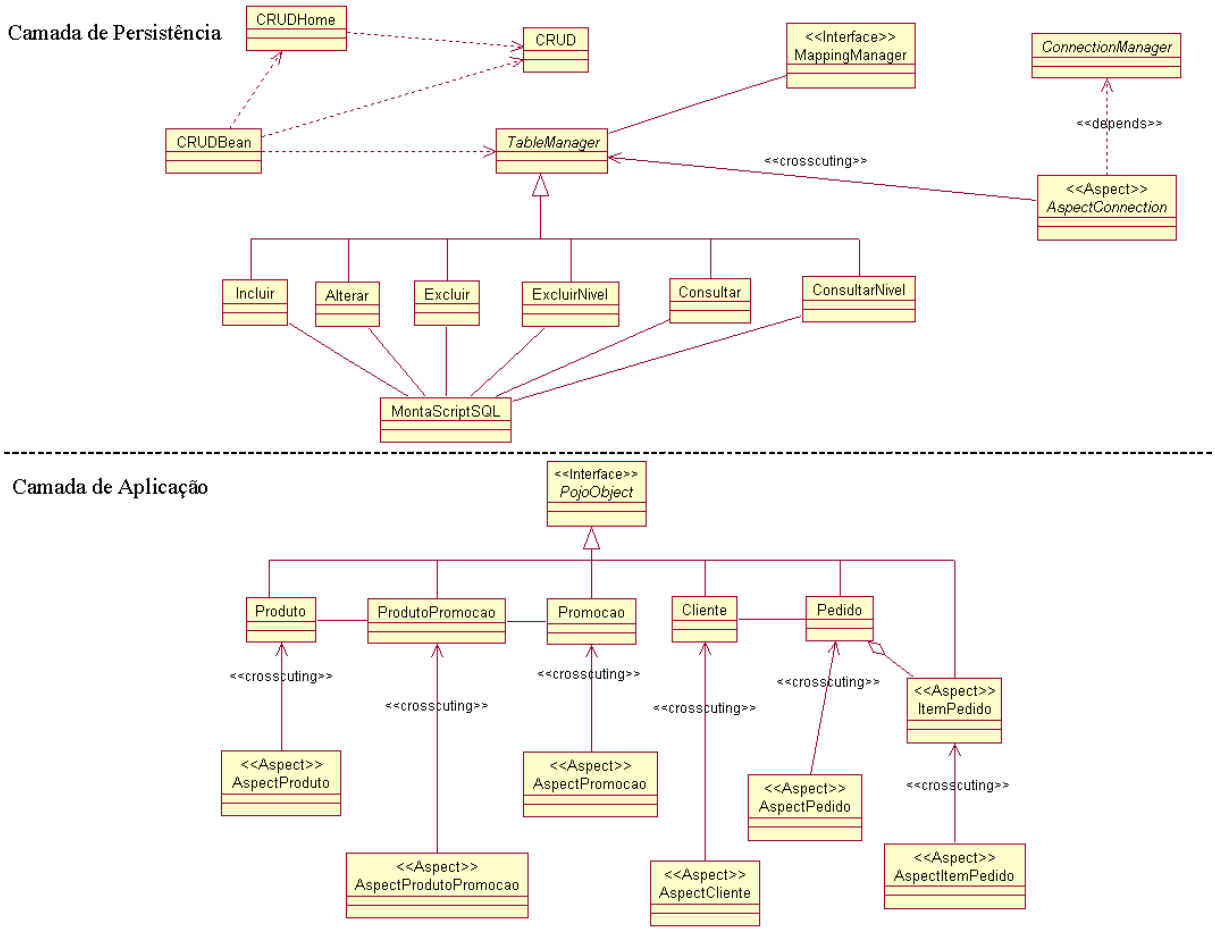


Figura 34 - Visão geral do modelo do *framework*

A seção seguinte justifica a utilidade do *framework* para a área de Comércio Eletrônico.

5.4 *Framework* e Comércio Eletrônico

A contribuição do *framework* é a possibilidade de automatizar a geração de aplicações para o Comércio Eletrônico, com características/funcionalidades específicas desta área. O *framework* através do seu *kernel* possibilita a reutilização de classes já implementadas para um domínio específico de aplicação direcionado para Comércio Eletrônico, deixando a escolha de que classes especializar para o desenvolvedor da loja virtual.

Podem ser criadas aplicações de lojas virtuais que atendam um certo público com características diferenciadas, ou aplicações simples que atendem necessidades básicas dos clientes de uma loja virtual. Operações relacionadas às transações de dados no banco de dados não precisam ser codificadas, pois o *framework* oferece a implementação de classes específicas para a camada de persistência.

Assim, podem-se obter resultados mais proveitosos com o uso de aplicações desenvolvidas por este *framework*, principalmente por atender a dois interesses específicos: ambientes de Comércio Eletrônico e persistência de dados.

6 IMPLEMENTAÇÃO DO *FRAMEWORK*

Com base na modelagem do *framework*, foi realizada sua implementação. Para isto foram utilizadas as seguintes tecnologias (linguagens e ferramentas) de desenvolvimento:

- Java, mais especificamente o JDK (*Java Development Kit*) na sua versão j2sdk1.4.2_01 encontrado para *download* gratuito no endereço <http:// www.java.sun.com>;
- J2EE, *Java 2 Platform Enterprise Edition* na sua versão 1.3.1 encontrado para *download* gratuito no endereço <http://www.java.sun.com/j2ee/>;
- AspectJ – Compilador - versão 1.2.1 encontrado para *download* gratuito no endereço <http://www.eclipse.org/downloads/index.php>;
- JBoss – Servidor de aplicação - versão 4.0.1, encontrado para *download* gratuito no endereço <http://www.jboss.com>;
- Eclipse SDK – Ferramenta de desenvolvimento – versão 2.1, encontrado para *download* gratuito no endereço <http://www.eclipse.org/downloads/index.php>;
- AJDT – *AspectJ Development Tools* – versão 1.1.4, encontrado para *download* gratuito no endereço <http://www.eclipse.org/downloads/index.php>;
- PostgreSQL8.0 - *Object-relational database management system* (ORDBMS) – versão 8.0, encontrado para *download* gratuito no endereço <http://www.postgresql.org>;
- Rational Rose 2000 Enterprise Edition – Ferramenta de modelagem.

Na etapa de desenvolvimento do *framework* tornou-se necessário o estudo de recursos da linguagem *AspectJ* [39] [40] especialmente recursos de pontos de junção, adendos e declaração de inter-tipos.

Na tabela 7, são apresentados, de forma compacta, os pontos de escolha de tecnologia, as opções consideradas e a justificativa pela escolha.

A familiarização com algumas destas tecnologias já utilizadas em outros projetos profissionais também contribuiu para a determinação das escolhas realizadas, levando-se em consideração, principalmente, o prazo para a realização do trabalho.

Pontos	Opções consideradas	Escolha	Critérios
Linguagem de programação	Java C++	Java	100% Orientada a Objetos. Portabilidade (Sistema em evolução).
Banco de dados	Relacional OO	Relacional (PostgreSQL)	SGBD (Sistema Gerenciador de Banco de Dados) objeto relacional de código aberto e de uso livre para aplicações comerciais.
Servidor de aplicação *Frontend*	JBoss TomCat	JBoss	Executa JSP, J2EE. *Freeware*.
Linguagem *Frontend*	JSP ASP	JSP	Ambiente Java.
Ferramenta de desenvolvimento	IBM VisualAge for Java NetBeans Eclipse	Eclipse	Ambiente de desenvolvimento com integração a AJDT. *Open Source*.

Tabela 7 - Tecnologias utilizadas no desenvolvimento do *framework*

6.1 Arquitetura do *Framework*

A figura 35 apresenta uma visão geral da arquitetura do *framework*. Essa arquitetura foi desenvolvida baseada na plataforma *Web*, que trabalha a apresentação dos dados no lado do cliente e o processamento no lado do servidor. A seguir é apresentada uma definição de cada uma das camadas utilizadas.

- Camada cliente: o cliente visualiza as informações através de um *browser*;
- Camada de apresentação: o cliente (ou sistema externo) interage com a aplicação. Nesta camada, são fornecidos serviços de aplicações cliente/servidor com a finalidade de fornecer interface para aplicações no servidor. Essa interação pode ser limitada pelo tipo de aplicação e servidor *Web* utilizados. Alguns exemplos de tecnologias utilizadas são: CGI (Common Gateway Interface), ASP (Active Server Pages), Servlets, JSP (Java Server Pages);
- Camada de negócios: são executados os processos relacionados com a lógica do negócio além de permitir a interação com outros sistemas (internos ou externos);
- Camada de integração: é feita a modelagem dos componentes persistentes transacionais;

- Camada de recursos: todos os recursos utilizados para armazenagem e recuperação de dados estão contidos nesta camada. Regras de integridade de dados, servidores de mensagens e servidores de impressão podem estar contidos nesta camada de recursos.

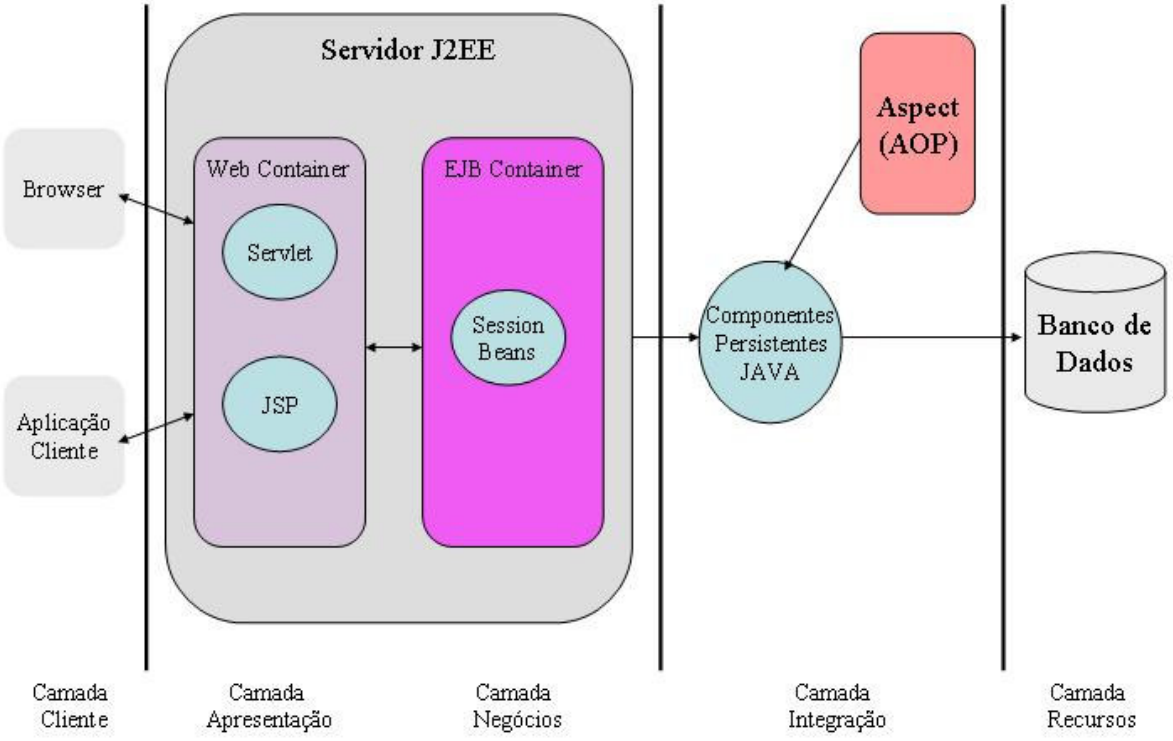


Figura 35 - Arquitetura do *framework*

Para o desenvolvimento do *framework* foi utilizada a tecnologia J2EE [5] [34], baseada no desenvolvimento de componentes EJBs (*Enterprise Java Beans*) do tipo *Stateless Session Beans*.

Session Beans são direcionados para uso de um cliente somente. São componentes que provêem lógica de negócio, tal como o cálculo de retorno de um investimento, ou podem salvar um estado, por exemplo, um carrinho de compras para um cliente *Web*. O ciclo de vida de um *Session Bean* não deverá ser mais longo do que o do cliente. Quando um cliente deixa um *Web site* ou a aplicação é desligada, o *Session Bean* desaparece [5].

O tipo de componente *Stateless Session Bean* é um componente de negócio que não mantém conversação com o cliente, não possuindo garantias de que chamadas sucessivas de métodos remotos vão ser feitas no mesmo objeto. A escolha deste tipo de componente se deu por ser possível modelar serviços reutilizáveis da camada de persistência, prevendo um serviço genérico para acesso às requisições. As operações relativas ao sub-padrão CRUD [62] não foram implementadas diretamente na classe EJB, visto que essa forma não seria a melhor forma de promover reuso e

manutenção do código, mas, sim, em uma classe de controle responsável pelas chamadas às operações requisitadas. As classes EJBs terão somente chamadas de métodos que delegarão a execução à classe de controle, chamada *TableManager*.

J2EE é um modelo de componentes do lado do servidor, conforme apresentado na figura 35, onde o Servidor J2EE contém um EJB Container que é responsável por armazenar todo componente EJB. Para desenvolver um componente EJB é preciso seguir algumas regras, pois existem interfaces responsáveis por tarefas definidas, regras de herança, etc.

O JCommerceNet é um *framework* totalmente *white-box*, ou seja, exige que se conheça o código-fonte para realizar sua instanciação. A instanciação pode ser feita estendendo as classes do *framework* ou complementando o código de classes já existente.

A seguir serão listados os principais métodos das classes mais importantes – Componentes EJBs e Classes Aplicação – assim como métodos das principais classes associadas a elas, de forma a fornecer mais detalhes sobre como proceder na instanciação do *framework*.

6.2 Componente EJB

As classes que formam a estrutura do componente EJB e que serão descritas a seguir são: CRUD, CRUDBean, CRUDHome.

6.2.1 Classe CRUD

A interface CRUD (interface Remota) herda da interface *javax.ejb.EJBObject* que por sua vez herda de *java.rmi.Remote*. Toda interface que define chamadas remotas deve herdar da interface *java.rmi.Remote*. Na interface remota encontram-se os métodos que o componente vai publicar para chamada por clientes remotos.

A figura 36 apresenta a interface CRUD definindo as seis operações definidas para o sub-padrão CRUD [62] que podem ser chamadas remotamente.

```
public interface CRUD extends EJBObject {

    public Long incluir(PojoObject pojo) throws RemoteException, Exception;
    public Boolean excluir(PojoObject pojo)
                            throws RemoteException, Exception;
    public Long alterar(PojoObject pojo) throws RemoteException, Exception;
    public ResultSetRetorno consultar(PojoObject pojo)
                                        throws RemoteException, Exception;
    public ResultSetRetorno consultarNivel(ArrayList tablesFrom,
                                            ArrayList columnsProjection,
                                            ArrayList columnsJoin,
                                            ArrayList orderBy) throws
                                            RemoteException, Exception;
    public Boolean excluirNivel(ArrayList tables,
                                ArrayList chaves,
                                Long id) throws RemoteException, Exception;
}
```

Figura 36 - Classe CRUD

6.2.2 Classe CRUDHome

A interface CRUDHome (interface Home) define como se pode criar o componente CRUDHome, apresentado na figura 37. É através dessa interface que o cliente obtêm a interface Remota do *Bean*.

```
public interface CRUDHome extends EJBHome {
       public CRUD create() throws RemoteException, CreateException;
}
```

Figura 37 - Classe CRUDHome

6.2.3 Classe CRUDBean

A classe CRUDBean pode ser utilizada para implementação da lógica de negócio, ou seja, das operações definidas para o sub-padrão CRUD [62]. Porém, nesta classe optou-se por somente fazer a chamada aos métodos que delegarão a execução a uma classe de controle. Assim, por exemplo, o método incluir() é responsável apenas pela chamada à implementação que foi desenvolvida na classe Incluir. A figura 38 apresenta como é feita essa chamada.

```
public Long incluir(PojoObject pojo) throws RemoteException, Exception {
       Long retorno = null;
       try {
              Incluir incluir = new Incluir();
              incluir.entidade = pojo;
              incluir.executar();
              retorno = incluir.retorno;
       } catch (Exception e) {
              throw new Exception(e.getMessage());
       }
       return retorno;
}
```

Figura 38 - Método incluir() da classe CRUDBean

6.3 Controle e Persistência

As principais classes que formam a estrutura de controle das chamadas aos métodos do sub-padrão CRUD [62] são: *TableManager* e suas subclasses Incluir, Alterar, Excluir, Consultar, ConsultarNivel e ExcluirNivel.

6.3.1 Classe *TableManager*

O construtor da classe *TableManager* apenas inicializa o estado da conexão ao banco de dados da aplicação (Figura 39).

```
public TableManager(Connection conn) throws Exception {
        this.conectarDB(conn);
}
```

Figura 39 - Construtor da Classe *TableManager*

O método conectarDB(), apresentado na figura 40, é responsável por verificar se a conexão passada por parâmetro ao construtor da classe é nula ou não. Se for nula, obtem uma nova conexão ao banco de dados, através da chamada *getConnection()*, caso contrário, mantem a mesma referência a conexão.

```
private void conectarDB(Connection conn) throws Exception {
     try {
             if (conn == null) {
              this.conn = getConnection();
             this.conn.setAutoCommit(false);
             } else {
                this.conn = conn;
             }
     } catch (Exception e) {
           ---
    }
}
```

Figura 40 - Método conectarDB() da classe *TableManager*

A chamada *getConnection(),* da figura 40, é interceptada pelo aspecto *AspectConnection*, apresentado na figura 41, que é responsável por executar a chamada a implementação do método *getConnection()* pela classe *ConnectionManager*. Esta chamada deverá se implementada na instanciação do *framework*, já que o método *getConnection()* é abstrato na classe *ConnectionManager*.

```
public abstract aspect AspectConnection {
     public abstract void TableManagergetConnection();
     /*Aqui deve ser implementada a chamada ao método getConnection da classe
ConnectionManager, retornando a conexão para a classe TableManager*/
}
```

Figura 41 - Método abstrato *TableManager.getConnection()* no aspecto *AspectConnection*

A classe *ConnectionManager* possui um método abstrato *getConnection(),* apresentado na figura 42, que deve ser implementação no momento de instanciação do *framework*.

```
public abstract class ConnectionManager {
      public abstract void getConnection(){
      /*Deverá implementar como pegar a conexão, por exemplo, via JNDI, ou   JDBC e
retorná-la*/
       }
}
```

Figura 42 - Método abstrato *getConnection()* da classe *ConnectionManager*

O método executar(), apresentado na figura 43, é responsável por encapsular as chamadas que fazem parte do fluxo de execução de um evento. O método *processarQuery()* é um método abstrato que deverá ser implementado por cada classe que estende a classe *TableManager*.

```
public void executar() throws Exception {
        boolean success = false;
        try {
            this.processarQuery();
        } catch (Exception e) {
                throw new Exception(e.getMessage());
        } finally {
            try {
                this.closeDB(success);
            } catch (SQLException se) {
                throw new Exception(se.getMessage());
            }
   }
}
```

Figura 43 - Método executar() da classe *TableManager*

O método closeDB(), apresentado na figura 44, é responsável por finalizar a conexão com o banco de dados da aplicação, após a execução de uma das operações implementadas pelo sub-padrão CRUD [62].

```
private void closeDB(boolean success) throws SQLException {
  try{
      if (this.isOwner) {
        if (this.conn != null) {
          if (success) {this.conn.commit();} else {this.conn.rollback();}
        }
       }
 } finally{
      this.conn.close();
  }
}
```

Figura 44 - Método closeDB() da classe *TableManager*

6.3.2 Classe Incluir

Todas as classes que estendem a classe *TableManager*, seguem um mesmo padrão de implementação que a classe *TableManager* controla. Nesta seção, será apresentada apenas a classe Incluir, já que as demais seguem as mesmas seqüências de implementação, porém com alguns detalhes diferentes.

O construtor da classe Incluir apenas inicializa o estado da conexão ao banco de dados da aplicação (Figura 45), fazendo referência à superclasse.

```
public Incluir(Connection conn) throws Exception {
       super(conn);
}
```

Figura 45 - Construtor da classe Incluir

O método *processarQuery()*, apresentado na figura 46, é implementado nesta classe, pois é um método específico, responsável pela execução do evento. O método mapearAtributos() faz chamada ao método mapear() da classe *PojoObject*, que é implementado por cada classe da camada de aplicação, onde o mapeamento de cada atributo do objeto é referenciado conforme seu tipo a uma coluna do banco de dados.

```
public void processarQuery() throws Exception {
        try {
             this.mapearAtributos();
             this.executarInclusao();
        } catch (Exception e) {
             throw e;
        }
}
```

Figura 46 - Método processarQuery() da classe Incluir

O método executarInclusao(), apresentado na figura 47, é responsável por executar a inclusão de um registro no banco de dados. Neste método é efetuada toda a transação para inclusão de um registro. Observa-se que não existe nenhum código de script no método, visto que isso deixaria o método não reutilizável, assim, optou-se por criar uma classe de script a qual contêm todos os métodos necessários para a montagem dos scripts utilizados pelas operações. O método montaScriptSQLIncluir(), recebe um map que contêm todas as colunas do objeto mapeado, para montar o script. Após isso, o método converteTiposInclusão(), faz a conversão de cada coluna mapeada no banco de dados para o objeto correspondente em Java, de forma que o atributo *psmt* do tipo *PreparedStatement* possa fazer o *executeUpdate*().

```
private void executarInclusao() throws Exception {
        PreparedStatement pstmt = null;
        try {
             String sql = MontaScriptSQL.montaScriptSQLIncluir(this.map);
             pstmt = super.conn.prepareStatement(sql);

            Long chave;
            if (map.getKeyValue() == null) {
                chave = getNextPk(super.conn, this.map.getSeqName());
             this.map.setKeyValue(chave);
            } else {
                chave = map.getKeyValue();
            }

            converterTiposInclusao(pstmt, this.map);
           pstmt.executeUpdate();
           this.retorno = chave;
        } finally {
            if (pstmt != null) {
                pstmt.close();
            }
        }
}
```

Figura 47 - Método executarInclusao() da classe Incluir

6.4 Camada da Aplicação

O modelo do *kernel* do *framework* apresenta algumas classes que são comuns às diferentes aplicações de lojas virtuais que podem ser implementadas. Para exemplificar como o modelo foi implementado, serão apresentadas as classes *PojoObject*, Produto e o aspecto AspectProduto e seus métodos respectivamente.

6.4.1 Classe *PojoObject*

A classe *PojoObject* é uma classe abstrata na qual todas as classes que fazem parte da camada de aplicação devem estendê-la, assim como as novas classes que devem ser implementadas no momento da instanciação do *framework*.

O construtor da classe *PojoObject* é responsável por instanciar um objeto da classe *MappingManager* (Figura 48).

```
public PojoObject() {
	map = new MappingManager();
}
```

Figura 48 - Construtor da classe *PojoObject*

O método mapear(), apresentado na figura 49, é um método abstrato que deve ser implementado pela classe que estende a classe *PojoObject*.

```
protected abstract void mapear();
```

Figura 49 - Método abstrato mapear() da classe *PojoObject*

O método *getMapping*(), apresentado na figura 50, é responsável por executar as chamadas aos métodos, que fazem parte da execução do mapeamento dos atributos dos objetos para as colunas no banco de dados. O método *clearMap*() inicializa o vetor utilizado para fazer o mapeamento. O método mapear() é chamado através da classe que o implementa.

```
public MappingManager getMapping() {
	this.map.clearMap();
	this.mapear();
	return this.map;
}
```

Figura 50 - Método *getMapping()* da classe *PojoObject*

6.4.2 Classe Produto

A classe Produto é uma classe de aplicação onde contêm os atributos referentes ao objeto e seus métodos *get()* e *set()*. A figura 51 apresenta a classe com seus atributos e métodos.

```
public class Produto extends PojoObject {
	private Long   idProduto;
	private String descricao;
	private Double preco;
	private String foto;

	public Produto() {}
	protected void mapear() {}

	public Long getIdProduto() {     return idProduto;}
	public String getDescricao() {return descricao;}
	public String getFoto() {return foto;}
	public Double getPreco() {return preco;}

	public void setIdProduto(Long idProduto) {
		this.idProduto = idProduto;
	}
	public void setDescricao(String descricao) {
		this.descricao = descricao;
	}
	public void setFoto(String foto) {
		this.foto = foto;
	}
	public void setPreco(Double preco) {
		this.preco = preco;
	}
}
```

Figura 51 - Classe Produto

6.4.3 AspectProduto

O aspecto AspectProduto, apresentado na figura 52, é responsável por interceptar chamadas ao construtor da classe Produto e também por implementar o método abstrato da classe *PojoObject* mapear(). O ponto de corte *doConstrutor()* intercepta as execuções do construtor normal da classe Produto. Essa interceptação fornece detalhes específicos da classe da aplicação, responsável por inicializar três parâmetros da classe. O outro ponto de corte doMapping() intercepta chamadas ao método mapear() da classe Produto, responsável por definir os atributos de persistência de classe.

```
public aspect AspectProduto {
    pointcut doConstrutor(Produto produto): target (produto) && execution
(Produto.new());
    pointcut doMapping(Produto produto): target (produto) && execution (protected void
Produto.mapear());

before (Produto produto): doConstrutor(produto){
    produto.map.setTableName("PRODUTO");
    produto.map.setOrderBy("IDPRODUTO");
    produto.map.setSeqName("PRODUTO_SEQ");
}
before (Produto produto): doMapping(produto) {
    produto.map.addMapping("IDPRODUTO",produto.getIdProduto(),Types.INTEGER);
    produto.map.addMapping("DESCRICAO",produto.getDescricao(),Types.VARCHAR);
    produto.map.addMapping("PRECO",produto.getPreco(), Types.DOUBLE);
    produto.map.addMapping("FOTO",produto.getFoto(), Types.VARCHAR);
    produto.map.addMapping("IDDEPARTAMENTO",produto.getIdDepartamento(),
Types.INTEGER);
    produto.map.addMapping("IDSECAO",produto.getIdSecao(),Types.INTEGER);
}
}
```

Figura 52 - Aspecto AspectProduto

6.5 Como Implementar *hot-spots*

Os *hot-spots* são classes abstratas ou de interface que possuem métodos sem implementação. Quando queremos utilizar um *hot-spot* na criação de uma aplicação, devemos estender sua classe ou implementar sua interface, conforme for o caso. Além disso, um *hot-spot* também pode ser um método abstrato dentro de uma classe concreta que não possui implementação. Na instanciação do *framework*, esse método deve ser implementado de acordo com o requisito.

Um *hot-spot* existente no modelo do JCommerceNet é o *getConnection()*, um método abstrato dentro da classe *ConnectionManager*. Para adicionar código a esse método, é preciso codificar o método existente, tornando-o *static* e retornando um objeto *Connection*.

A seguir é apresentado o código da classe *ConnectionManager* e a estrutura do método *getConnection*() criada a partir de sua implementação, que já faz parte do *framework* (Figura 53).

```
/** Classe abstrata responsável por gerenciar as conexões com banco de dados
 * Deve ser implementada na instanciação do framework
*/
public abstract class ConnectionManager {
      /**
        * Método abstrato getConnection que deverá ser implementado na
          instanciação do framework
       */
       public abstract void getConnection() {
       }
}
/*Exemplo de instanciação*/
public class ConnectionManager {
       public static Connection getConnection() throws SQLException, Exception {
              (..) // O código desejado vai aqui
          return Connection;
       }
}
```

Figura 53 - Classe *ConnectionManager* e método *getConnection()*

7 ESTUDO DE CASO

Este capítulo apresenta um estudo de caso no qual será instanciado o *framework* para a criação de uma loja virtual de livros. Este estudo de caso cobre os principais *hot-spots* levantados durante a análise de requisitos de uma loja virtual.

7.1 Introdução

O sistema desenvolvido para a construção da loja virtual de livros deve dar suporte a manutenção dos cadastros básicos necessários para o funcionamento da loja, que são os seguintes: Produtos, Promoções, Produtos em promoção, Cliente, Seção, Departamento, Livros; além disso, deve disponibilizar os produtos destinados à venda ao cliente no momento que acessar o *site* da loja na Internet. Ao navegar pela loja, o cliente deverá realizar suas compras e adicionar os produtos no Carrinho de Compras. Quando desejar finalizar uma compra, deverá estar disponível o Caixa da loja responsável por efetivar o Pedido do cliente.

A loja de livros está estruturada de forma que seus produtos possam estar organizados e separados por departamentos e seções, para que os clientes possam encontrar de uma forma mais organizada os livros que lhes interessam. Exemplos de departamentos e seções que a loja pode conter na sua estruturação são: departamento de Informática, seção de *Hardware*, de Internet, de Linguagens de Programação; departamento de Literatura Nacional, seção de Poesia, Romance, Suspense e de Terror, Contos e Crônicas; deparatmento Científico, seção de Informática e Cultura Geral, etc.

O responsável pela loja de livros é o gerente, cuja principal função é cadastrar os departamentos, as seções, os produtos, as promoções e os produtos em promoção. Toda alteração necessária em algum desses cadastros deve ser realizada para que o cliente, ao acessar o *site* da loja, possa encontrar a informação referente ao produto sempre atualizada. Após a manutenção dos produtos o cliente poderá acessar o *site* da loja e comprar os produtos disponíveis.

O sistema desenvolvido para o estudo de caso tem por objetivo validar as funcionalidades que o *framework* propõe, ficando de fora requisitos adicionais que uma loja virtual necessita na sua implementação, tais como: interface de comunicação com empresas de cobrança, login, interface de comunicação com empresas de entrega de mercadorias, entre outros. Assim, a interface desenvolvida apenas ilustra como deve funcionar o cadastro de um Produto, a adição de produtos no Carrinho de Compras, os dados que devem ser informados ao efetivar o pedido de compras, o cadastro do Cliente, não fornecendo suporte à cobrança e nem entrega das mercadorias, por exemplo.

As principais funcionalidades que o *framework* propõe são ilustradas na figura 54 através do diagrama de casos de usos UML [14]:

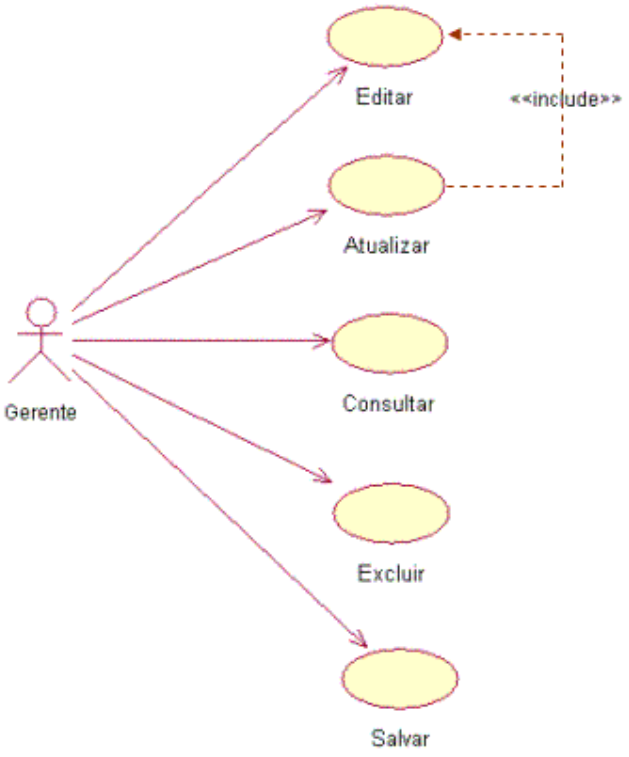


Figura 54 - Diagrama de casos de uso

7.2 Implementação

O sistema teve sua implementação desenvolvida para ambiente *Web*, utilizando banco de dados relacional PostgreSQL8.0 e JSP para a implementação das interfaces. A comunicação entre cliente-servidor foi implementada por meio de *servlets*, uma biblioteca da linguagem Java, ideal para sistemas baseados na *Web*. Um *servlet* estende a funcionalidade de um servidor, sendo eficiente para desenvolver soluções baseadas na *Web* que ajudam a fornecer acesso seguro a um *site* da *Web*, interagir com banco de dados em favor de um cliente, permitindo também manter sessão exclusiva de cada cliente.

O padrão Camada de Persistência (*Persistence Layer*) [62] foi implementado no paradigma OO. POA foi utilizada na implementação da definição dos mapeamentos dos atributos das classes persistentes e também na chamada ao método que faz a referência de conexão com o banco de dados.

Para criação da loja, foram estendidos os *hot-spots*: Departamento, Secao, TipoProduto, CarrinhoCompra, Caixa e FormaPagamento.

O diagrama de classes para a implementação dos *hot-spots* pode ser visualizado na figura 55. As classes persistentes de aplicação devem estender a classe *PojoObject*. Para cada *hot-spots* estendidos, foi criado um aspecto para fazer o mapeamento dos atributos do objeto com as tabelas e colunas respectivas ao banco de dados. A classe CarrinhoCompra não é apresentada no modelo

porque é uma classe somente utilizada para instanciar um objeto CarrinhoCompra, sendo que seus dados não serão persistidos ao banco de dados diretamente.

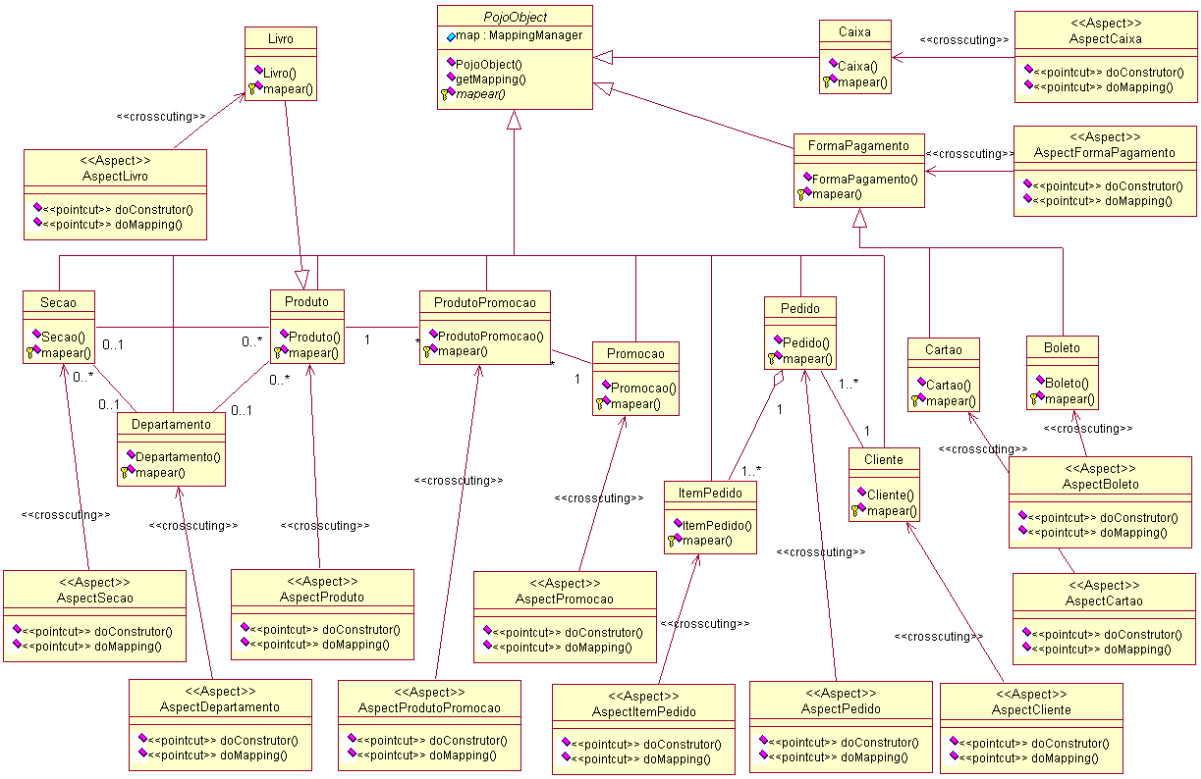


Figura 55 - Modelo de classes da aplicação com *hot-spots* estendidos

A figura 56 apresenta um trecho de código do *hot-spot* TipoProduto especializado para a classe Livro. Todo TipoProduto especializado deve estender a classe Produto, de forma que haja reaproveitamento de código através do mecanismo de herança. Foram definidos alguns atributos que seriam necessários para informação dos dados ao cliente.

```
public class Livro extends Produto {
	private Long   idLivro;
	private String autor;
	private String editora;
	(..) // mais atributos

	public Livro() {}
	protected void mapear(){}

	public String getAutor() {return autor;}
	public String getEditora() {return editora;}
	public Long getIdLivro() {return idLivro;}
	(..)//mais métodos

	public void setAutor(String autor) {this.autor = autor;}
	public void setIEditora(String editora) {this.editora =editora;}
	public void setIdLivro(Long idLivro) {this.idLivro = idLivro;}
}
```

Figura 56 - Classe Livro

A figura 57 apresenta um trecho de código do aspecto AspectLivro. Para cada classe persistente na aplicação, deve ser criado um aspecto para que este implemente o método mapear() da classe, e também intercepte chamadas do construtor.

```
public aspect AspectLivro {
	pointcut doConstrutor(Livro livro): target (livro) && execution (Livro.new());
	pointcut doMapping(Livro livro): target (livro) && execution (protected void Livro.mapear());

	before (Livro livro): doConstrutor(livro){
		livro.map.setTableName("LIVRO");
		livro.map.setOrderBy("IDLIVRO");
	}
	before (Livro livro): doMapping(livro) {
		livro.map.clearMap();
		livro.map.addMapping("IDLIVRO",livro.getIdLivro(),Types.INTEGER);
		livro.map.addMapping("AUTOR",livro.getAutor(), Types.VARCHAR);
		livro.map.addMapping("EDITORA",livro.getEditora(),Types.VARCHAR);
	}
}
```

Figura 57 - AspectLivro

O *hot-spots* do método *getConnection()* também foi implementado de forma que a aplicação conecte-se com o banco de dados. A figura 58 apresenta a implementação do método *getConnection()*.

```
public static Connection getConnection() throws SQLException, Exception {
     try {
       initialContext = new InitialContext();
       dataSource = (javax.sql.DataSource) initialContext.lookup("java:jdbc/postgresqlDS");
      } catch (Exception e) {
       e.printStackTrace();
      }
       return dataSource.getConnection();
}
```

Figura 58 - Implementação do método *getConnection*() da classe *ConnectionManager*

O método montaScriptPK() foi implementado com o uso do banco de dados PostgreSQL8.0. A figura 59 apresenta sua implementação.

```
public static String montaScriptPK(String pkName) {
      StringBuffer sb = new StringBuffer();
      sb.append("SELECT nextVal ('"+pkName+"') AS CHAVE");
      System.out.print(sb.toString());
      return sb.toString();
}
```

Figura 59 - Método montaScriptPK da classe MontaScriptSQL

Na implementação do método *getConnection()*, optou-se por utilizar JNDI (*Java Naming and Directory Interface*) da API J2EE, que fornece uma interface padrão para localização de serviços, usuários, máquinas, objetos e redes, além de permitir acessá-los através de nomes e diretórios através de aplicações JAVA. Assim, o controle do usuário e senha para acesso aos dados no banco de dados pode ser configurado no servidor de aplicação diretamente, nos arquivos de configuração do *JBoss*.

Um exemplo de configuração desse arquivo é apresentado na figura 60.

```
<datasources>
 <local-tx-datasource>
  <jndi-name>jdbc/postgresqlDS</jndi-name>
  <connection-url>jdbc:postgresql://localhost:5432/PROJETOTC</connection-url>
  <driver-class>org.postgresql.Driver</driver-class>
  <user-name>postgres</user-name>
  <password>1234</password>
    <exception-sorter-class-name>
       org.jboss.resource.adapter.jdbc.vendor.PostgresqlExceptionSorter
    </exception-sorter-class-name>
</local-tx-datasource>
</datasources>
```

Figura 60 - Exemplo do arquivo *datasource.xml* no *JBoss*

A seguir, será apresentado como a implementação das páginas Web e classes JAVA foram programadas. Como exemplo, para ilustrar como funciona a requisição de dados de um cliente ao servidor, utilizou-se o cadastro de departamentos de loja virtual de livros. A Figura 61 apresenta uma página JSP criada para o cadastro de departamentos da loja.

```
<%@page contentType="text/html" %>
<%@page import="java.io.*"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<title>JSP Page</title>
<link rel="stylesheet" href="../styles/projeto.css" type="text/css">
<script src="../lib/forms.js" type="text/javascript"></script>

<SCRIPT>
 function salvar(){
     document.forms[0].action = "../ServletDepartamento?acao=SALVAR";
     document.forms[0].submit();
 }
</SCRIPT>

</head>
<body leftmargin="0" topmargin="0">
<form name="frmTela" method="post">
Nome:
<input type="text" size="30" name="nome" id="nome" maxlength="60" value="">
Descrição:
<input type="text" size="30" name="descricao" id="descricao" maxlength="100" value="">
<INPUT TYPE="Button" value="Salvar" onClick="salvar()">
</form>
</body>
</html>
```

Figura 61 - Página JSP de entrada de dados para o ServletDepartamento

O código apresentado na figura 61 mostra uma pequena parte da implementação da página do cadastro de departamentos da loja. A interface gerada na execução deste código é apresentada na figura 62.

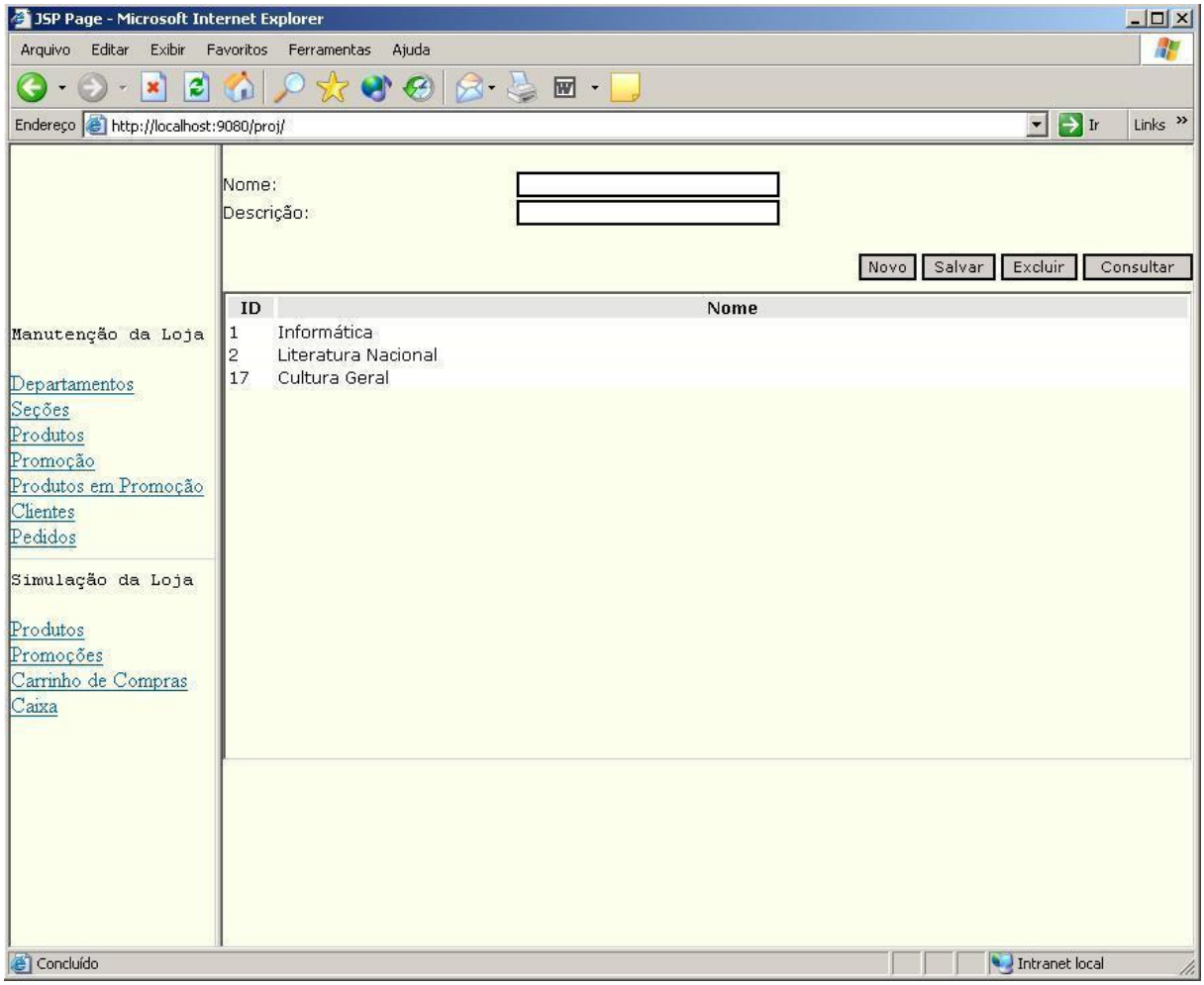


Figura 62 - Modelo de interface do cadastro de departamentos da loja virtual

Cada botão na interface da figura 62 é responsável por uma ação na tela, por exemplo: botão Novo, limpa os campos do formulário; botão Salvar submete uma ação ao *servlet*, para inserir ou alterar dados; botão Excluir submete uma ação ao *servlet* que efetuará a exclusão do registro; e botão Consultar submete uma ação ao *servlet* que efetuará a consulta dos registros inseridos no banco de dados.

Quando é feita uma solicitação para inserir um registro no banco de dados, através do botão Salvar, a função salvar() submete uma requisição para o ServletDepartamento, que tem sua implementação conforme apresentada na figura 63. Além do recebimento das requisições da interface do cliente, o *servlet* tem como função fazer as chamadas à camada de persistência para que os dados recebidos sejam persistidos no banco de dados, e também enviar uma confirmação para a interface do cliente se a operação ocorreu com sucesso ou não.

```
public class ServletDepartamento extends HttpServlet {
public void init (ServletConfig config) throws ServletException {
      super.init(config); }
public void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
Departamento departamento = new Departamento();
HttpSession session = request.getSession();
if (request.getParameter("acao")!=null){
      if (request.getParameter("acao").toUpperCase().equals("SALVAR")){
      departamento.setIdDepartamento(Utils.reqL("idDepartamento",request));
      departamento.setNome(Utils.req("nome",request));
      departamento.setDescricao(Utils.req("descricao",request));
     if (departamento.getIdDepartamento()!=null){
         alterar(departamento);
     }else incluir(departamento);
}else if (request.getParameter("acao").toUpperCase().equals("EXCLUIR")){
      departamento.setIdDepartamento(Utils.reqL("idDepartamento", request));
      excluir(departamento);
      }}
departamento = new Departamento(); consultar(departamento,session);
response.sendRedirect("./jsp/cadastroDepartamento.jsp");}
private void alterar(PojoObject pojo) throws Exception {
      Alterar alterar = new Alterar();
      alterar.entidade = pojo;
      alterar.executar();}
private void incluir(PojoObject pojo) throws Exception {
      Incluir incluir = new Incluir();
      incluir.entidade = pojo;
      incluir.executar();}
private void excluir(PojoObject pojo) throws Exception {
      Excluir excluir = new Excluir();
      excluir.entidade = pojo;
      excluir.executar();}
private void consultar(PojoObject pojo, HttpSession session) throws Exception{
      Consultar con = new Consultar();
      con.entidade = pojo;
      con.executar();
      ResultSetRetorno rs = con.retorno;
      session.setAttribute("ResultSetDepartamento",rs);
}
}
```

Figura 63 - *Servlet* que recebe as requisições da página JSP

Os métodos criados no *servlet*, conforme apresentado na figura 63, são utilizados para envio dos dados para os métodos do padrão Camada de Persistência, responsáveis por executar todo o serviço de persistência dos dados.

Cada *servlet* utilizado durante o desenvolvimento deve estar declarado no descritor *Web* (*web.xml*) do servidor de aplicação. A figura 64 apresenta como descrever o *servlet* no arquivo.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app>
  <servlet>
       <servlet-name>ServletDepartamento</servlet-name>
       <servlet-class>framework.servlet.ServletDepartamento</servlet-class>
  </servlet>
  <servlet-mapping>
        <servlet-name>ServletDepartamento</servlet-name>
        <url-pattern>/ServletDepartamento</url-pattern>
  </servlet-mapping>
</web-app>
```

Figura 64 - Descritor *Web* (*web.xml*) contendo o ServletDepartamento

A figura 65 apresenta uma interface da loja virtual de livros. Esta interface serve apenas para ilustrar como os dados podem ser apresentados para o cliente, no momento que acessar o *site* da loja virtual. Nesta figura são apresentados os produtos disponíveis para venda na loja virtual.

A implementação da interface, o acesso à camada de persistência e banco de dados podem ser feitos de modo flexível, tais como: escolha do banco de dados, *layout* da interface da loja, uso ou não de *servlet* e escolha do servidor de aplicação. Dessa forma, não criam-se dependências, nem limitações referentes à escolha de recursos e apresentação dos dados para o cliente.

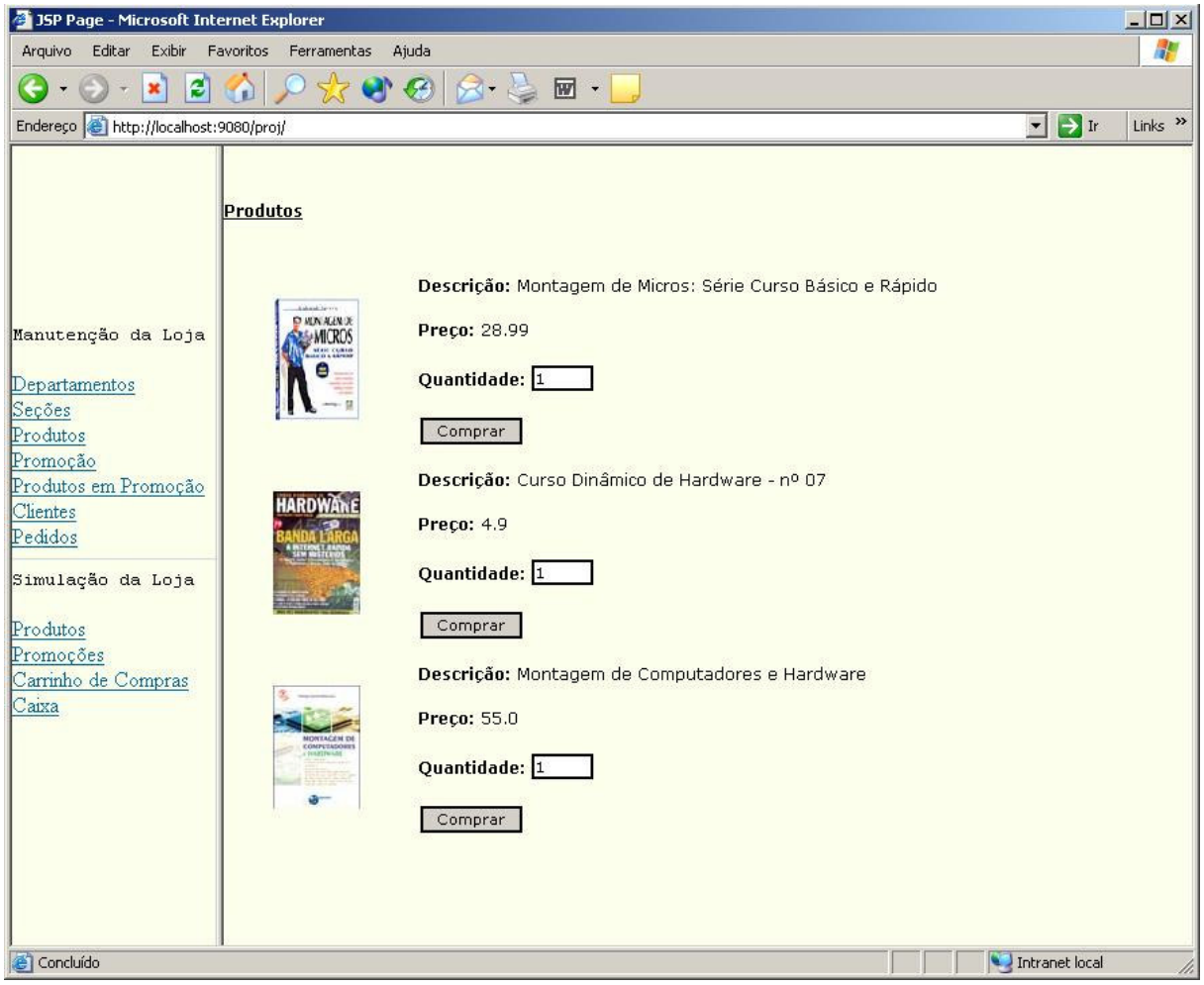


Figura 65 - Modelo de interface dos produtos da loja virtual

A loja virtual disponibiliza produtos em promoção, que são cadastrados na interface de Produtos em Promoção. Estes produtos são apresentados na figura 66. A apresentação é similar a de produtos apresentada na figura 65, porém, é apresentado para o cliente o preço real do produto e o preço de desconto oferecido. A loja pode optar de cadastrar produtos em promoção, isso irá variar conforme os objetivos de cada loja virtual.

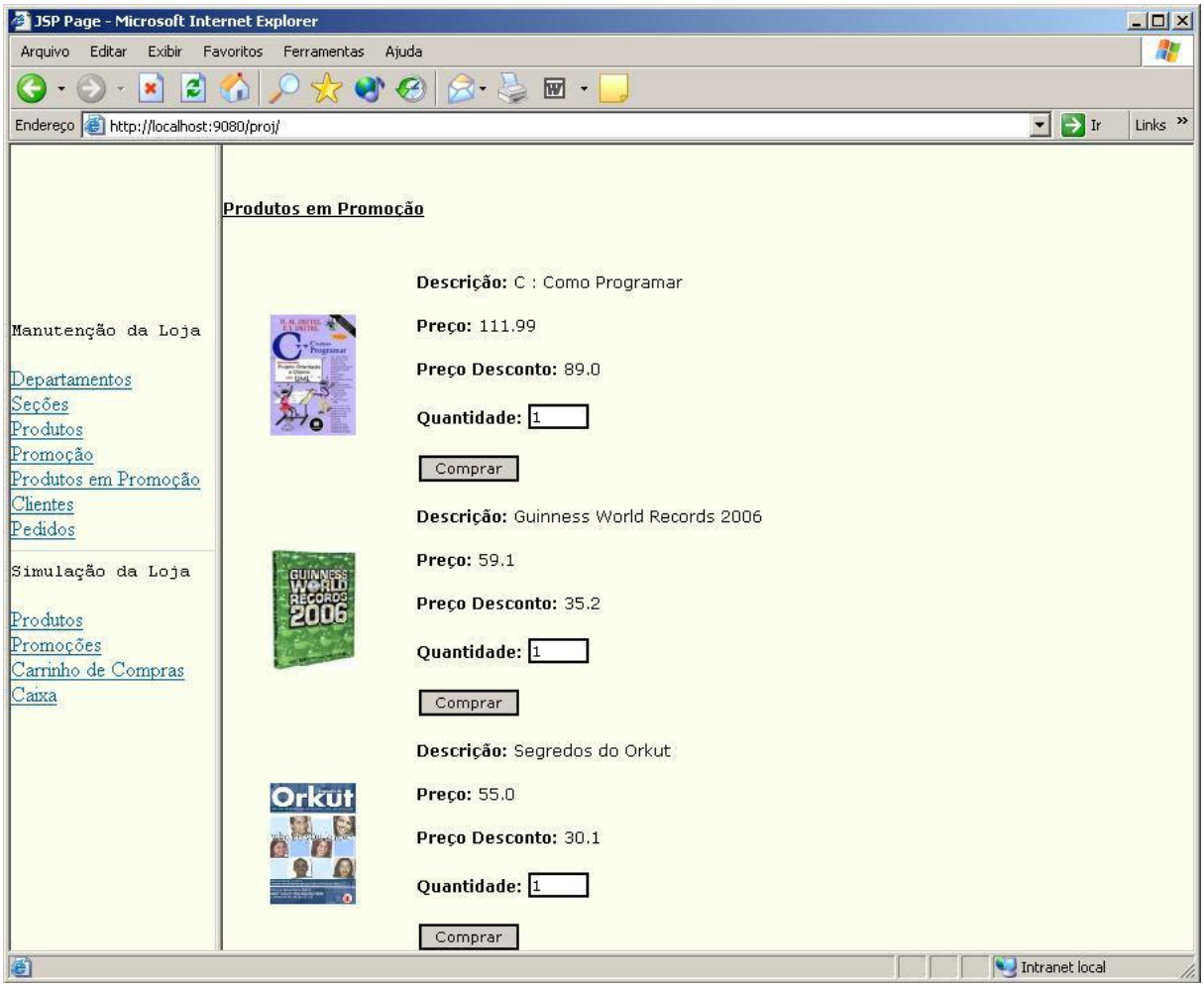


Figura 66 - Modelo de interface dos produtos em promoção da loja virtual

O cliente ao navegar pela loja virtual, adiciona os produtos que deseja comprar no carrinho de compras, clicando no botão Comprar, conforme apresenta a figura 66. Estes produtos serão adicionados no carrinho de compras e, quando o cliente desejar visualizá-los, o *link* disponível na interface da loja, Carrinho de Compras, ao ser clicado irá apresentar todos os produtos que constam no carrinho de compras, permitindo ao cliente sua visualização, atualização e retirada dos produtos. A figura 67 apresenta alguns produtos adicionados no carrinho de compras para apresentação da interface.

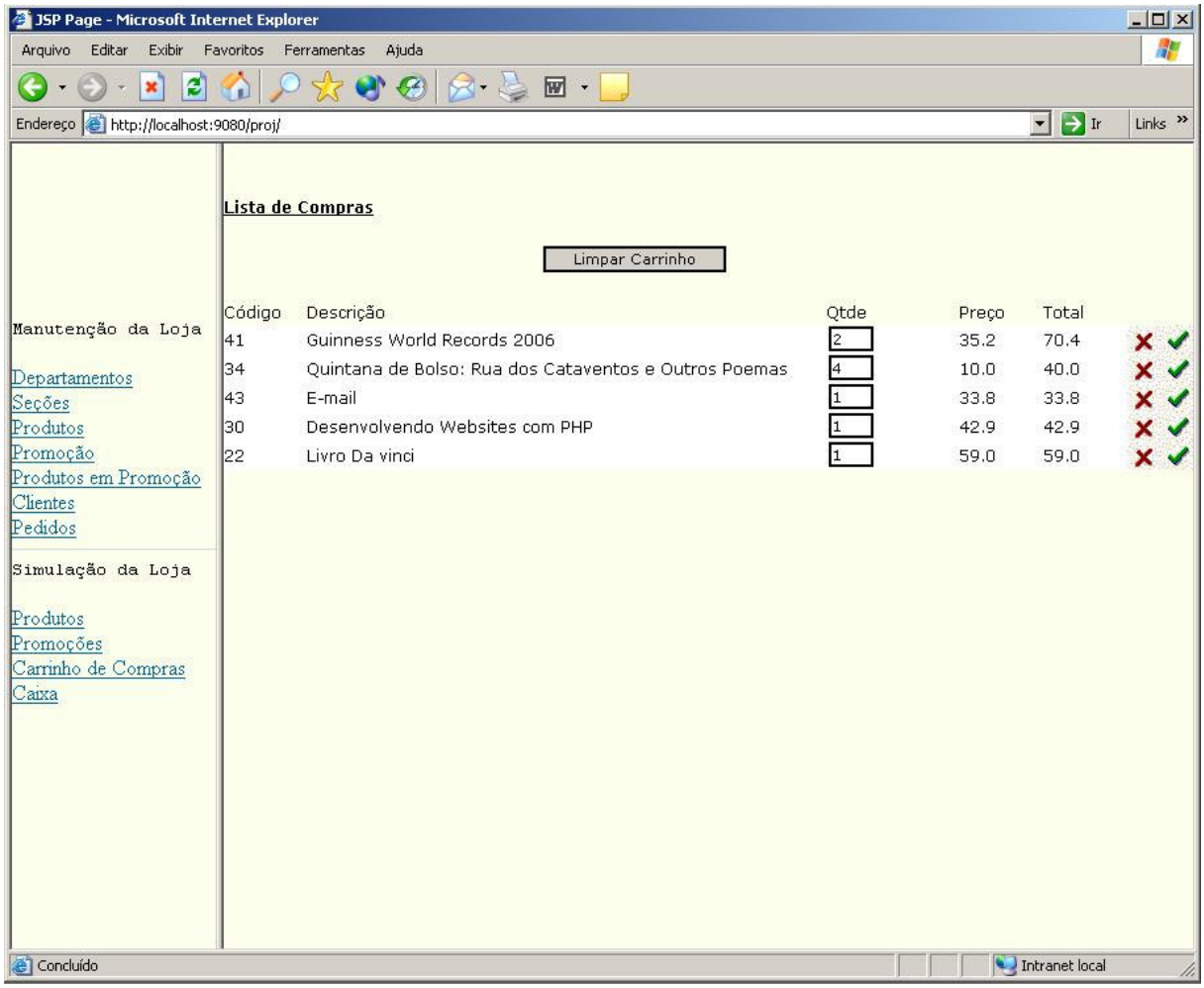


Figura 67 - Modelo de interface do carrinho de compras da loja virtual

Quando o cliente desejar finalizar a compra, o *link* Caixa na interface da loja, encaminha o cliente a uma interface onde deverá informar os dados necessários para finalizar a compra. A figura 68 apresenta como foi implementada a estrutura dessa interface.

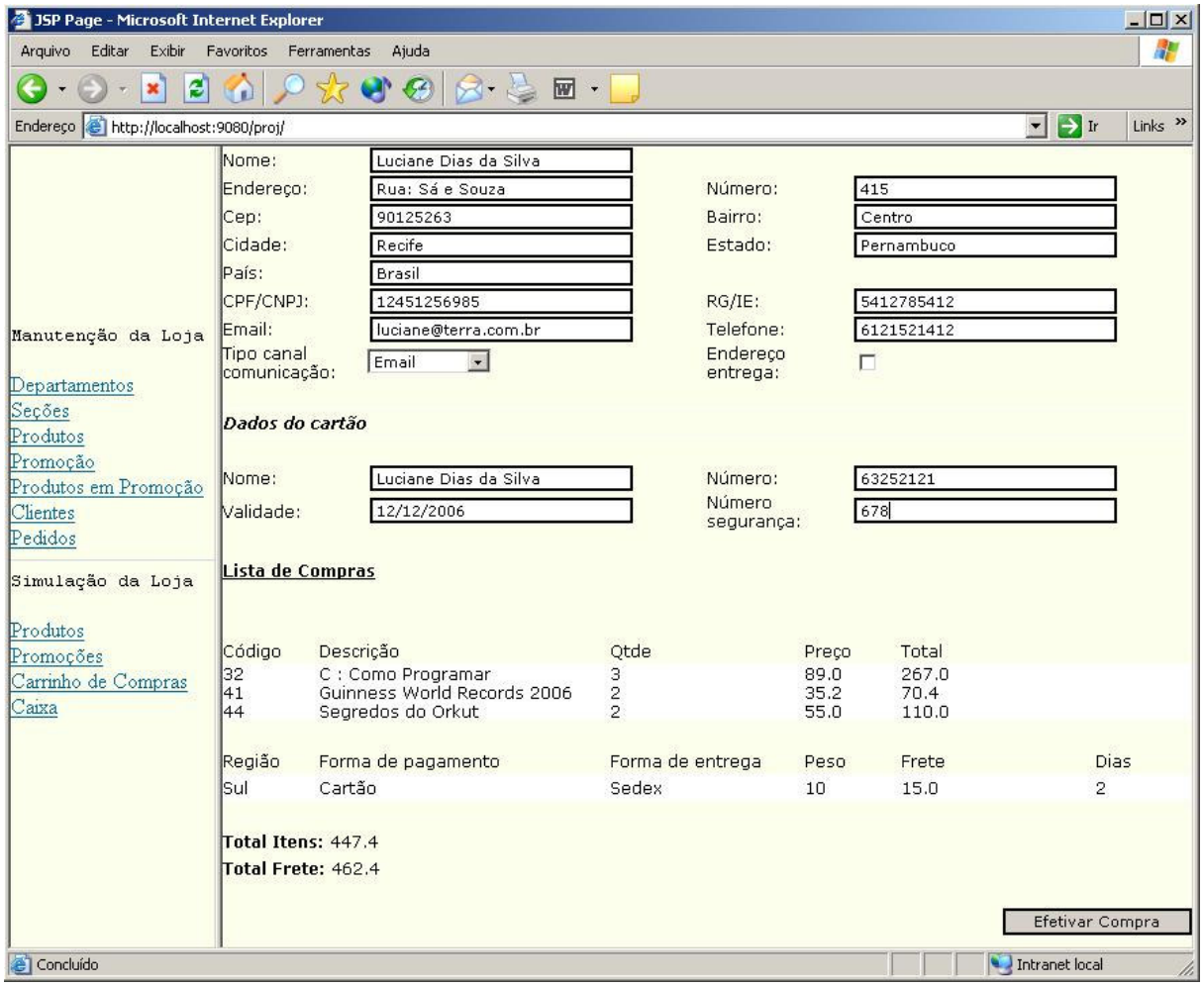


Figura 68 - Modelo de interface para finalização do pedido de compra da loja virtual

8 CONCLUSÃO

O JCommercerNet é um *framework* orientado a objetos utilizando aspectos, desenvolvido especificamente para a área de comércio eletrônico, que pode ser utilizado para automatizar o desenvolvimento de lojas virtuais.

A metodologia utilizada (processo baseado na análise de domínio) mostrou-se consistente para o desenvolvimento do JCommercerNet, uma vez que havia experiência já adquirida no domínio do problema ao qual o *framework* está relacionado.

O processo de análise de requisitos não apresentou problemas no que diz respeito à estrutura da aplicação, por se tratar de um sistema conhecido. A busca de informações junto a produtos e serviços que constróem lojas virtuais, bem como a busca de aplicações que tivessem o maior número de características aproximadas a esse domínio, foi essencial para complementar a visão dos requisitos necessários ao *framework*.

A principal contribuição do trabalho é o desenvolvimento de um *framework* para a área de comércio eletrônico, que permitirá a geração de scripts para a camada de persistência em seus diferentes tipos de instanciação promovendo flexibilização e reutilização de *software* através do uso de *frameworks*, Padrões de Projeto e aspectos. Buscou-se dotar essas ferramentas, que existem já a muito tempo, de recursos/características para o comércio eletrônico, de forma que possam ser utilizadas em ambiente *Web* de maneira mais proveitosa.

O ambiente foi desenvolvido segundo uma metodologia de desenvolvimento em espiral. Nas fases de implementação e instanciação do protótipo utilizou-se a linguagem Java, AspectJ, Servlet, JSP e HTML.

O desenvolvimento deste *framework* foi de um grande aprendizado e propiciou o contato com áreas que despertaram um grande interesse. O conhecimento adquirido com o estudo de *frameworks*, Padrões de Projeto e aspectos foi importante. Foi possível perceber na prática que, apesar da complexidade que há na criação deste tipo de ferramenta, os benefícios obtidos são muito grandes e vale a pena investir em sua criação.

Por ser aplicado à área de comércio eletrônico, o trabalho tornou-se ainda mais proveitoso. O aprendizado foi muito grande, tendo em vista o pouco contato anterior com assuntos relacionados a comércio eletrônico. Ficou clara a importância do desenvolvimento de uma loja virtual e principalmente hoje, quando se tem na *Web* uma grande fonte de recursos e oportunidades. Sem dúvida, é um tema de muita relevância e que merece toda a atenção de comerciantes, pesquisadores e instituições. A implementação de um exemplo de aplicação do *framework* foi trabalhosa, mas propiciou um melhor entendimento do desenvolvimento de *frameworks* e uso de Padrões de Projeto e aspectos, e um maior conhecimento da linguagem Java, que se mostra adequada para o desenvolvimento de diversos tipos de aplicações.

8.1 Considerações Finais

O avanço da Internet proporcionou o desenvolvimento do comércio eletrônico em todas as partes do mundo. O comércio eletrônico envolve a realização de negócios por meio da Internet, incluindo venda de produtos e serviços. Pesquisas indicam que o crescimento do comércio eletrônico é uma tendência natural decorrente do aumento da base de internautas (usuários da Internet) e da descoberta das vantagens da compra *on-line*, sendo que esse crescimento deve continuar em ritmo forte pelo menos até o final desta década.

A loja virtual é um segmento de comércio eletrônico onde as empresas implementam *sites* na Internet, com a intenção de divulgar seus produtos e fornecer informações aos clientes na forma de catálogos *on-line* e também pelo prestígio proporcionado pela vitrine *on-line*. Desenvolver uma loja virtual que efetivamente venda será uma ferramenta para o sucesso no comércio eletrônico, uma vez que essa irá replicar, no ambiente virtual, uma loja e seus vendedores.

A tecnologia de *frameworks* aliada ao uso de padrões de projeto para desenvolvimento de sistemas proporciona redução de tempo, custo e reuso de componentes, com o aumento da qualidade do *software* produzido. Capturar os conceitos mais gerais de um domínio de aplicações não é uma tarefa fácil, é necessária uma boa experiência no domínio da aplicação. Pode-se obter diferentes aplicações (pertencentes a um mesmo domínio) sem a necessidade de novo trabalho de análise, projeto e codificação de todo o ambiente. Apenas se define o comportamento dos *hot-spots* desejados para aquela aplicação. Esses expressam aspectos do domínio do *framework* que não podem ser antecipados, mas que são passíveis de customização e extensão. Além do uso de *frameworks* e Padrões de Projeto proporcionarem benefícios ao desenvolvimento de sistemas, o uso de aspectos também proporciona benefícios ao desenvolvimento, principalmente na separação de interesses de um sistema. Definir esses interesses e separá-los no projeto de um sistema proporcionam melhor reuso e manutenção de código.

A união dessas tecnologias se mostra adequada para áreas onde os requisitos mudam rapidamente, como o comércio eletrônico. Integrar novas funcionalidades a uma ferramenta torna-se mais fácil do que nos modos tradicionais de desenvolvimento orientado a objetos.

8.2 Trabalhos Futuros

O JCommerceNet, na forma como foi desenvolvido, representa a primeira implementação referente ao seu desenvolvimento.

Em relação ao desenvolvimento do *framework*, pretende-se dar continuidade ao projeto, de forma a aprimorar a primeira versão disponibilizada. Alguns aperfeiçoamentos futuros surgiram do decorrer do seu desenvolvimento, são eles:

- Aprimorar o *framework* elaborado ampliando-o com a implementação de novos *hot-spots*; entre eles, destaca-se alguns: *hot-spots* de *login*, de interface de comunicação com empresas de cobrança, de interface de comunicação com empresas de entrega de mercadorias;

- Ampliar o *framework* de forma que possa disponibilizar recursos de geração de interfaces para a construção de lojas virtuais, conforme configurações fornecidas pelos clientes;

- Adaptar as classes do *framework* orientado a objetos para gerar um *framework* totalmente orientado a aspectos;

- Disponibilizar, após a ampliação do *framework* para a geração de interfaces da loja, um serviço de geração e publicação da loja virtual via Internet. Este serviço poderá contar com alguns exemplos modelos (*templates*) de lojas virtuais, oferecendo uma possível simulação da loja para o cliente, antes de efetivar a compra do serviço;

9 REFERÊNCIAS BIBLIOGRÁFICAS

[1] Apple Computer Inc. ***MacAppII Programmer´s Guide***, 1989.

[2] ALBERTIN, Alberto Luiz. **Comércio Eletrônico: benefícios e aspectos de sua aplicação**. Revista de Administração de Empresas – RAE, EAESP/FGV, v.38, n° 1, pp. 52-63, Jan/Mar 1998.

[3] ALBERTIN, Alberto Luiz. **Comércio Eletrônico: modelo, aspectos e contribuições de sua aplicação**. 4.ed. São Paulo: Atlas, 2002, 292p.

[4] ALEXANDER, Christopher; ISHIKAWA, Sara; SILVERSTEIN, Murray; JACOBSON, Max; FIKSDAHL-KING, Ingrid; ANGEL Shlomo. ***A Pattern Language***. Oxford: Oxford University Press, New York, 1977.

[5] ALLAMARAJU, S. et. al. ***Professional Java Server Programming J2EE Edition***. Reading: Wrox Press, 2000.

[6] ***AspectC Team*** [*on-line*]. Disponível em: <http://www.cs.ubc.ca/labs/spl/projects/aspectc.html>. Acessado em: 03 ago. 2005.

[7] ***AspectC#*** [*on-line*]. Disponível em: <http://www.dsg.cs.tcd.ie/index.php?category_id=169>. Acessado em: 03. ago. 2005.

[8] AspectJ Team [*on-line*]. **Site do projeto inicial**. Disponível em: <http://www.parc.com/research/projects/aspectj/default.html>.Acessado em: 11 abr. 2005.

[9] AspectJ Team [*on-line*]. ***The AspectJ Programming Guide***. Disponível em:<http://www.eclipse.org/aspectj/>. Acessado em: 11 abr. 2005.

[10] ***AspectPHP*** [*on-line*]. Disponível em: <http://www.cs.toronto.edu/~yijun/aspectPHP/>. Acessado em: 03 ago. 2005.

[11] BLOCH, M.; PIGNEUR, Y. ; SEGEV, A. ***On the road of eletronic commerce: a business value framework, gaining competitive advantage and some research issue***. Lausanne: Institut D'Informatique et Organization, Ecole des Hautes Etudes Commerciales, Université de Lausanne, 1996.

[12] BERGMANS, Lodewijk; AKSIT, Mehmet. ***Composition Crosscutting Concerns Using Composition Filters***. *Communications of the ACM*, vol.44, n° 10, pp 51-57, October 2001.

[13] BOOCH, Grady. ***Object-Oriented Analysis and Design with Applications***. Second edition. Reading: Addison-Wesley, 1994, 589p.

[14] BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. ***The Unified Modeling Language User Guide***. Reading: Addison-Wesley, 1999.

[15] BOLIN, Sherrie. ***E-Commerce: A Market Analysis and Prognostication***. *Communications of the ACM*, vol. 6, nº 3, pp 97-105, September 1998.

[16] BUSCHMANN, F.. ***The Master-Slave Pattern***. Proceedings of the First Conference on Pattern Languages and Programming, Addison-Wesley, 1994.

[17] CRESPO C. S. Pinto, Sérgio. **Composição em WebFrameworks**. Tese de Doutorado, PUC-Rio, Departamento de Informática, 2000.

[18] DIJKSTRA, E. W. ***A Discipline of Programming***. 1.ed. New Jersey – USA: Prentice-Hall, 1976, 217p.

[19] ***Eclipse.org.*** [*on-line*] 2005. Disponível em: <http://eclipse.org>. Acessado em: 11 abr. 2005.

[20] ELRAD, Tzilla; FILMAN, Robert E.; BADER Atef. ***Aspect-oriented programming: Introduction***. *Communications of the ACM*, v. 44, n° 10, pp. 29-32, 2001a.

[21] ELRAD, Tzilla; KICZALES, Gregor; AKSIT, Mehmet; LIEBERHERR, Karl; OSSHER, Harold. ***Discussing Aspects of AOP***. *Communications of the ACM*, v. 44, n° 10, pp. 33-38, 2001b.

[22] FAYAD, Mohamed E.; SCHMIDT, Douglas C.; JOHNSON, Ralph E.. ***Building Application Frameworks: Object-Oriented Foundations of Framework Design***. New York: John Wiley & Sons, 1999, 664p.

[23] FIRESMITH, D.G.; ***Frameworks: the golden path to object Nirvana***, Journal of Object-Oriented Programming, vol. 7, nº 8, 1994.

[24] FONTOURA, Marcus; PREE, Wolfgang; RUMPE, Bernhard. ***UML-F: A Modeling Language for Object-Oriented Frameworks***, ECOOP´2000, Springer-Verlag, LNCS 1850, pp. 63-82, 2000.

[25] FONTOURA, Marcus; LUCENA, Carlos, J.; ANDREATTA, Alexandre; CARVALHO, Sérgio, E.; RIBEIRO, Celso, C..***Using UML-F to Enhance Framework Development: a Case Study in the Local Search Heuristics Domain***. Journal of Systems and Software 57. pp. 201-206, 2001.

[26] FORTUNATO, Luigi X..**2BuyNet – Um Framework para Instanciação e Administração de Lojas para a Internet**. Tese de Mestrado, PUC-Rio, Departamento de Informática, 1999.

[27] FROEHLICH, Garry; HOOVER, James; LIU, Ling; SORENSON, Paul. ***Designing Object-Oriented Frameworks***. University of Alberta, Canada.

[28] GAMMA, Erich; HELM, Richard; JOHNSON, Ralph; VLISSIDES, John. **Padrões de Projeto: Soluções Reutilizáveis de Software Orientado a Objetos**. Porto Alegre: Bookman, 2000, 364p.

[29] GHEZZI, Carlo; JAZAYERI, Mehdi; MANDRIOLI, Dino. ***Fundamentals of Software Engineering***. Prentice-Hall Inc, 1991.

[30] HARRISON, Willian; OSSHER, Harold. ***Subject-Oriented Programming (a critique of pure objects)***. In OOPSLA'93, pages 411-428. ACM Press, September 1993.

[31] IBGE. **As micro e pequenas empresas comerciais e de serviços no Brasil 2001**. Disponível em: <http://www.ibge.gov.br/home/estatistica/economia/microempresa/default.shtm>. Acessado em: 22 jul. 2005.

[32] **Java** [*on-line*]. Disponível em: <http://java.sun.com>. Acessado em: 23 out. 2005.

[33] **JavaWorld** [*on-line*]. Disponível em: <http://www.javaworld.com/javaworld/jw-01-2002/jw-0118-aspect-p4.html>. Acessado em: 30 jul. 2005.

[34] **J2EE** [*on-line*]. Disponível em: <http://java.sun.com/j2ee/>. Acessado em: 23 out. 2005.

[35] JOHNSON, Ralph E.; FOOTE Brian. ***Designing reuseable classes***. Journal of Object-Oriented Programming, Junho/Julho 1988.

[36] JOHNSON, Ralph E. ***Reusing Object-Oriented Design***. University of Illinois, Technical Report UIUCDCS 91-1696, 1991.

[37] KAPLAN, Steven; SAWHNEY, Mohanbir. ***E-Hubs: The New B2B Marketplaces***. *Harvard Business Review* (May-June 2000).

[38] KICZALES, Gregor; LAMPING, John; MENDHEKAR, Anurag; MAEDA, Chris; LOPES, Cristina V.; LOINGTIER, Jean-Marc; IRWIN, John. ***Aspect-Oriented Programing***. In Proceedings of the 11th European Conference on Object-Oriented Programming (ECOOP '97), Finland, Springer-Verlag, LNCS 1241, pp. 220-242, 1997.

[39] KICZALES, Gregor; HISLSDALE, Erik; HUGUNIN, Jim; KERSTEN, Mik A.; PALM, Jeffrey; GRISWOLD, Willian G.. ***An Overview of AspectJ***. European Conference on Object-Oriented Programming (ECOOP), Budapest, Hungary. Springer-Verlag, LNCS 2072, pp. 327-353, 2001.

[40] KICZALES, Gregor; HISLSDALE, Erik; HUGUNIN, Jim; KERSTEN, Mik A.; PALM, Jeffrey; GRISWOLD, Willian G.. ***Getting Started with AspectJ***. *Communications of the ACM*, vol. 44, n° 10, pp 59-65, October 2001.

[41] LAJOIE, R.; KELLER, R.K.; **Design and Reuse in Object-oriented Frameworks : Patterns, Contracts and Motifs**, in Concert, Proceedings of the 62nd Congress of the Association Canadienne Francaise pour I'Avancement des Sciences, Montreal, Canada. May 1994.

[42] LAUDON, Kenneth C.; TRAVER, Carol G.. ***E-commerce: business, technology, society***. Reading: Addison Wesley, 2002, 762p.

[43] LEE, Ken W. K.. ***An Introduction to Aspect-Oriented Programming***. COMP 610E 2002. Spring Software Development of E-Business Applications.

[44] LIEBERHERR, Karl; ORELANS, Doug; OVLINGER, Johan. ***Aspect-Oriented Programming with Adaptive Methods***. *Communications of the ACM*, vol.44, n° 10, pp 39-41, October 2001.

[45] LIMEIRA, Tani M. V. **E-Marketing - O Marketing na Internet com casos brasileiros**. Editora Saraiva, 2003.

[46] LOPES, Cristina, KICZALES, Gregor. ***D: A Language Framework form Distributed Programming***. Technical Report. Xerox PARC, February, 1997.

[47] LUCENA, Carlos J. P. de; MARKIEWICZ, Marcus E.. ***Understanding Object-Oriented Framework Engineering***. PUC – Rio. Inf.MCC38/00, October, 2000.

[48] MATTSSON, M. ***Object-oriented Frameworks – A survey of methodological issues***. Licentiate Thesis, Department of Computer Science, Lund University, CODEN: LUTEDX/(TECS-3066)/1-130/(1996), also as Technical Report, LU-CS-TR: 96-167, Department of Computer Science, Lund University, 1996.

[49] MATTSON, M.; BOSCH, J.; FAYAD, M.. ***Framework Integration: Problems, Causes and Solutions***. *Communications of the ACM*, vol. 42, nº 10, pp 80-87, October 1999.

[50] MATTSON, M.. ***Evolution and Composition Object-Oriented Frameworks***. PhD Thesis. University of Karlskrona/Ronneby, Department of Software Engineering and Computer Science, 2000.

[51] MURPHY, Gail C.; WALKER, Robert J.; BANIASSAD, Elisa L.A.; ROBILLARD, Martin P.; LAI, Albert; KERSTEN, Mik A.. ***Does Aspect-Oriented Programming Work?*** *Communications of the ACM*, vol.44, nº 10, pp 75-77, October 2001.

[52] OSSHER, H; TARR, P. ***Using subject-oriented programming to overcome common problems in object-oriented software development/evolution***. In International Conference on Software Engineering, ICSE'99, pp 687-688, 1999.

[53] OSSHER, H; TARR, P. ***Using Multidimensional Separation of Concerns to (Re)shape Evolving Software***. *Communications of the ACM*, vol.44, nº 10, pp 43-55, October 2001.

[54] PREE, Wolfgang. ***Design Patterns for Object-Oriented Software Development***. Reading: Addison-Wesley, 1995.

[55] SCHMIDT, D.S.; STEPHENSON, P.. ***Experience Using Desing Patterns to Evolve Communication Software Across Diverse OS Platforms***. Proceddings of the 9th European Conference on Object-Oriented Programming, Aarbus, Debmark, 1995.

[56] SCHIMIDT, D.; FAYAD, M.; JOHNSON, Ralf.. ***Software Patterns***. *Communications of the ACM*, 39 (10), October 1997.

[57] SIMONYI, Charles. ***The Death of Computer Languages***, the Birth of Intentional Programming. Technical Report MSR-TR-95-52, Microsoft Research, September 1995.

[58] SOMMERVILLE, Ian. ***Software Engineering***. 6 ed. Harlow: Addison-Wesley, 2001.

[59] WEINAND, A.; GAMMA, E.E.; MARTY, R..***Design and Implementation of ET++, a Seamless Object-Oriented Application Framework***. Structured Programming, vol. 10, nº 2, July 1989.

[60] WEINSTEIN, A. **Segmentação de Mercado**. São Paulo: Atlas, 1995, pp. 222 e 223.

[61] YESIL, Magdalena. **Criando a loja virtual – Faça negócios com seu site na Internet**. Rio de Janeiro: Infobook, 1999, 339p.

[62] YODER, Joseph W.; JOHNSON, Ralph E.; WILSON, Quince D..***Connecting Business Objects to Relational Databases***. In Proceedings of the 5th Conference on the Pattern Languages of Programs, Monticello-IL-EUA, August, 1998.

Anexo A Documentação do *Framework*

- Classificação: *white-box framework;*
- Processo de desenvolvimento: baseado na análise do domínio e de aplicações já existentes (tais aplicações não foram desenvolvidas anteriormente pelo projetista do *framework*). Utilizou-se de recursos da Engenharia Reversa para definição das características pertinentes ao *kernel* do *framework* e as que constituíram os *hot-spots*;
- Instanciação: estendendo as classes abstratas, adicionando novas operações e atributos;
- Documentação: o *framework* se destina ao desenvolvimento de lojas virtuais de pequeno porte na Internet, usando aspectos para definição de *hot-spots* na geração de scripts para automação dessas lojas;
- Como utilizar o *framework*: para uso do *framework* é necessário instanciar as classes abstratas:
    - TipoProduto;
    - Seção;
    - Departamento;
    - CanalComunicacao;
    - TipoCanal;
    - Caixa;
    - CarrinhoCompra;
    - FormaPagamento;
    - TipoFormaPagamento;
    - FormaEntrega;
    - TipoFormaEntrega;
    - e, implementar alguns métodos abstratos, por exemplo: montaScriptPK da classe MontaScriptSQL, *getConnection()* da classe *ConnectionManager*, e implementar a chamada ao método *TableManager.getConnection()* no aspecto *AspectConnection*.

Anexo B Manual de Configuração da Aplicação

OBS: As configurações são válidas para ambiente Windows, demais ambientes deve ser consultado guia de instalação de cada ferramenta.

Instalação de ferramentas:

- Java - j2sdk1.4.2_01 – download gratuito no endereço <http://www.java.sun.com>;
- J2EE - j2sdkee1.3.1 – download gratuito no endereço <http://www.java.sun.com/j2ee>;
- AspectJ 1.2. – download gratuito no endereço <http://www.eclipse.org/downloads/index.php>;
- Jboss 4.0.1 – download gratuito no endereço <http://www.jboss.com>;
- PostgreSQL - 8.0 – download gratuito no endereço <http://www.postgresql.org>.

Instruções de configuração das variáveis de ambiente

Informações que devem ser adicionadas no *classpath*:

- c:\j2sdk1.4.2_01;
- c:\aspectj1.2\lib\aspectjrt.jar;
- c:\j2sdk1.4.2_01\jre\lib\ext\postgresql-8.0-310.jdbc3.jar.

Informações que devem ser adicionadas no *path*:

- c:\j2sdk1.4.2_01\bin;
- c:\aspectj1.2\bin.

Informações que devem ser adicionadas como variáveis de ambiente:

- Java_home - c:\j2sdk1.4.2_01
- Jboss_home - c:\jboss-4.0.1

Material que consta no CD-Rom:

- Arquivo com o estudo de caso, pasta ProjetoFramework. Deverá ser efetuada a cópia para a raiz ou outro diretório do disco;

- Arquivo com o servidor de aplicação *JBoss,* pasta jboss-4.0.1. Deverá ser efetuada a cópia para a raiz do disco. No servidor de aplicação encontram-se todos os arquivos *.class* e páginas JSPs criadas no desenvolvimento da aplicação;

- Arquivos de documentação JavaDoc gerado para o *framework*, JCommerceNet/doc e classes do *framework* para utlização no desenvolvimento de aplicação, pasta JCommerceNet/framework;

- Cópia do banco de dados para simulação de dados, pasta Postgresql.

Estrutura da aplicação *Web* no servidor de aplicação:

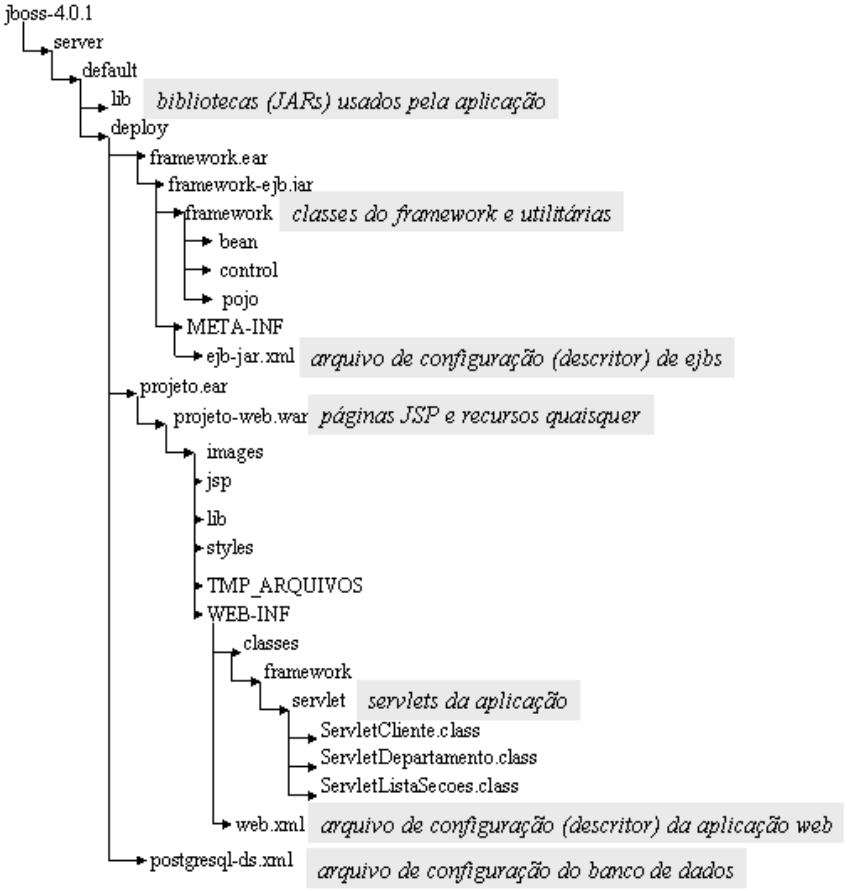


Figura 69 - Estrutura dos arquivos no servidor de aplicação