Universidade Estadual de Maringá
Centro de Tecnologia
Departamento de Informática

"IMPLEMENTAÇÃO DE LINHA DE PRODUTO DE SOFTWARE UTILIZANDO ASPECTOS: VARIABILIDADE DE RESTRIÇÃO DE CARTÕES"

Marcos Cláudio Ramos

TCC - 2009

Maringá - Paraná

2009

Universidade Estadual de Maringá
Centro de Tecnologia
Departamento de Informática

IMPLEMENTAÇÃO DE LINHA DE PRODUTO DE SOFTWARE UTILIZANDO ASPECTOS:VARIABILIDADE DE RESTRIÇÃO DE CARTÕES

Marcos Cláudio Ramos

TCC-INF-2009

Trabalho de Conclusão de Curso apresentado ao Curso de Informática do Centro de Tecnologia da Universidade Estadual de Maringá, como requisito parcial para a obtenção do título de Bacharel em Informática. Orientadora:*Prof.ª MSc. Thelma Elita Colanzi Lopes*

Maringá - Paraná
2009

Marcos Cláudio Ramos

Implementação de Linha de Produto de Software utilizando aspectos: Variabilidade de Restrição de Cartões

Este exemplar corresponde à redação final da monografia aprovada como requisito parcial para obtenção do título de Bacharel em Informática da Universidade Estadual de Maringá, pela banca examinadora formada pelos seguintes membros:

Orientadora:Prof.ª MSc. Thelma Elita Colanzi Lopes
Departamento de Informática, CTC, DIN

Prof.ª PhD. Itana Maria de Souza Gimenes
Departamento de Informática, CTC, DIN

Prof.ª PhD. Elisa Hatsue Moriya Huzita
Departamento de Informática, CTC, DIN

Maringá - Paraná
2009

Agradecimentos

Ao final de mais essa etapa de minha vida, agradeço a Deus em primeiro lugar que não me faltou em nenhum momento e sem o qual não teria conseguido superar os obstáculos impostos. Com semelhante importância, agradeço a minha esposa Paulina e minha filha Júlia que com enorme paciência e compreensão entenderam minha ausência em diversos momentos de suas vidas. Aos meus pais, senhor José e dona Neusa, que me guiaram pelos caminhos da vida com retidão de caráter e muita dignidade apesar de todas as dificuldades. Às minhas irmãs Marcia e Aline e ao meu irmão Edson que em suas orações sempre lembravam de mim. Aos professores e professoras do Departamento de Informática por tamanha dedicação e empenho em passar valiosos ensinamentos. Em especial à professora Thelma Elita por ter compreendido minhas aflições e pela tranquilidade transmitida, que foi de fundamental importância para a realização deste trabalho.

Resumo

A busca pela agilidade no processo de desenvolvimento de software tem sido alvo de diversos estudos. *Frameworks*, padrões de projeto, desenvolvimento de software baseado em componentes e outras técnicas são utilizadas para maximizar o reúso de software a fim de tornar o desenvolvimento de sistemas economicamente viável e consequentemente minimizar os esforços envolvidos. Nesse contexto, a abordagem de Linhas de Produtos de Software (LPS) é uma forma sistemática para a produção de software visando o reúso. Este trabalho tem por objetivo implementar variabilidades de uma LPS utilizando programação orientada a aspectos e analisar os resultados alcançados. A LPS utilizada foi a LPS-BET controle de Bilhetes Eletrônicos de Transporte (BET) que foi implementada utilizando componentes caixa-preta. Inicialmente foram escolhidas as variabilidades "Número de Cartões" e "Combinação de Cartões" para serem implementadas com o uso de aspectos. No decorrer do trabalho algumas limitações foram encontradas implicando em adaptações do projeto. Foi observado que o uso de aspectos para implementar requisitos funcionais nesta LPS não contribuiu positivamente mostrando que a implementação de variabilidades com componentes pode ser mais adequada no contexto da LPS-BET.

Palavras chaves: Engenharia de software, Programação Orientada a Aspectos, Linha de Produto de Software

Abstract

The search for the agility in the software development process has been the target of various studies in this sense. Frameworks, design patterns, component based software development and many other tecniques are used to improve the software reuse and to make system development economically feasible and therefore decrease effort involved. In this context, the software product line (LPS) are adopted as a systematic way for a software production. This project aims at implementing an LPS variability using aspect oriented programming and analyze the results obtained. The LPS used was LPS-BET. It is used to control of transport electronic tickets (BET) has been implemented using black box components is presented. Were initially chosen "Número de Cartões" and "Combinação de Cartões" variabilities to be implemented whith the use of aspect oriented programming. During the work some limitations had been found implying in adaptations of the project. It was observed that the use of aspects to implement functional requirements in this LPS, did not contribute positively showing that the implementation of variabilities with components can be more adjusted in the context of LPS-BET.

Keywords: Software Engineering, Aspect-Oriented Programming, Software Product Line

Lista de Figuras

Figura 1 Desenvolvimento do núcleo de artefatos com suas entradas e saídas 4

Figura 2 Desenvolvimento do produto 7

Figura 3 Representação da LMA por meio de formulários 9

Figura 4 LMA em formato XML 9

Figura 5 Adendo *before* 11

Figura 6 Adendo *after* 11

Figura 7 Adendo *around* 11

Figura 8 *Pointcut* não-nomeado 12

Figura 9 *Pointcut* nomeado 12

Figura 10 Anatomia de um aspecto 13

Figura 11 Diagrama de Característica do Núcleo da LPS-BET 15

Figura 12 Diagrama de Característica para a LPS-BET 16

Figura 13 Visão geral do processo de desenvolvimento da LPS-BET 17

Figura 14 Diagrama de Casos de Uso da LPS-BET 18

Figura 15 Arquitetura de componentes do núcleo da LPS-BET 19

Figura 16 Exemplo de separação de interesses utilizando componentes caixa-preta . . . 20

Figura 17 Arquitetura dos aspectos *Autenticacao* e *Autorizacao* e dos componentes entrecortados 21

Figura 18 Implementação do aspecto *Autenticacao* 22

Figura 19 Aspecto abstrato e aspectos contratos para representar a característica *Integracao* 22

Figura 20 Implementação do aspecto abstrato *IntegracaoCtrl* 23

Figura 21 Representação da Técnica Tunel SSH 29

Figura 22 Diagrama de classe representando a classe *GerenciaCartaoNumCartoes* . . . 30

Figura 23 Arquitetura parcial da LPS-BET - sem aspecto 30

Figura 24 Arquitetura parcial da LPS-BET - com aspecto 31

Figura 25 Classe *NumCartoesDAO* que implementa as interfaces alvo da interceptação 31

Figura 26 Aspecto *NumCartoesAspect* 32

Figura 27 Trecho de código interceptado por aspectos 35

Figura 28 Classe representando o padrão *template* 35

Figura 29 Classe concreta que implementa o método herdado 36

Lista de Tabelas

Tabela 1 Designadores da linguagem AspectJ 13

Tabela 2 Vantagens e desvantanges no uso de componentes do tipo caixa-branca ou caixa-preta 20

Tabela 3 Lista de recursos utilizados durante o processo de construção 23

Tabela 4 Algumas métricas da implementação da variabilidade número de cartões . . . 34

Sumário

1 Introdução . . . . . p. 1

2 Revisão Bibliográfica . . . . . p. 3

2.1 Linha de Produto de Software (LPS) . . . . . p. 3

2.1.1 Desenvolvimento do Núcleo de Artefatos . . . . . p. 4

2.1.2 Desenvolvimento do produto . . . . . p. 6

2.1.3 Gerenciamento de LPS . . . . . p. 7

2.2 Captor . . . . . p. 8

2.3 Programação Orientada a Aspectos . . . . . p. 9

3 Desenvolvimento da LPS-BET . . . . . p. 14

3.1 LPS-BET . . . . . p. 14

3.2 Decisões de projeto da LPS-BET . . . . . p. 19

3.3 Tecnologias de Implementação da LPS-BET . . . . . p. 23

4 Desenvolvimento de Variabilidade da LPS-BET usando aspectos . . . . . p. 26

4.1 Considerações Iniciais . . . . . p. 26

4.2 Configurações do Ambiente de Desenvolvimento . . . . . p. 26

4.2.1 Configuração do SVN . . . . . p. 26

4.2.2 Configuração do Maven . . . . . p. 26

4.2.3 Obtendo os fontes da LPS-BET . . . . . p. 27

4.2.4 Configurando a IDE Eclipse para fazer a depuração do código . . . . . p. 27

4.2.5 Configurando o acesso ao banco de dados . . . . . p. 28

4.3 Considerações pré-implementação . . . . . p. 29

4.4 Implementação da variabilidade Número de Cartões utilizando aspectos . . . . . p. 29

4.4.1 Limitações e dificuldades encontradas . . . . . p. 32

4.5 Métricas da Variabilidade Número de Cartões . . . . . p. 33

4.6 Considerações Finais . . . . . p. 34

5 Conclusão . . . . . p. 37

Referências Bibliográficas . . . . . p. 38

1 Introdução

Diversas pesquisas na área da engenharia de software estão voltadas para o reúso de software. Pesquisas estas que são de primordial importância para o mercado de desenvolvimento de software, uma vez que um dos requisitos fundamentais para a criação, desenvolvimento e manutenção de um projeto de software são os custos envolvidos em todo esse processo. Diversas técnicas para garantir (ou melhorar) a reutilização de software vem sendo empregadas tais como *frameworks* orientados a objetos, desenvolvimento baseado em componentes (DBC), geradores de código, Linhas de Produto de Software (LPS), padrões de projeto e linguagens orientadas a aspectos (Donegan, 2008). Estas técnicas podem ser empregadas isoladamente ou em conjunto com outras técnicas.

O resultado de todas essas pesquisas permitem que sistemas sejam desenvolvidos com maior rapidez e qualidade. Essa rapidez se dá pela montagem de sistemas com partes de softwares já produzidos, devidamente testados e validados e consequentemente diminuem o tempo e o custo de desenvolvimento. A "montagem" de sistemas pode ser feita com partes de softwares de seguimentos distintos ou semelhantes. Quando os sistemas possuem mais características semelhantes do que distintas estes sistemas passam a fazer parte de uma "família de sistemas"(Parnas, 1979).

As famílias de sistemas, também conhecidas como LPS, consistem de um conjunto de sistemas de software compartilhando características comuns e gerenciadas que satisfazem necessidades específicas de um segmento particular de mercado ou de negócio que são desenvolvidas a partir de um conjunto de ativos centrais de forma sistemática(Clements e Northrop, 2001). Em uma LPS, pode-se encontrar pontos de variação que são pontos onde as características do produto podem se diferenciar. Como exemplo de LPS podemos citar o sistema de bilhetes eletrônicos de transporte denominado LPS-BET que será alvo deste trabalho. Os produtos dessa LPS foram desenvolvidos para as cidades de Fortaleza, Campo Grande e São Carlos. A LPS-BET inicialmente foi desenvolvida usando componentes caixa-preta.

Um dos objetivos deste trabalho é implementar e analisar o uso da programação orien-

tada a aspectos para a implementação de variabilidades da LPS-BET. Tal objetivo se justifica em virtude de haver implementações de variabilidades utilizando componentes e também para se obter indícios da eficácia do uso de aspectos para a mesma finalidade. Além do fato deste trabalho fazer parte de um projeto maior que visa implementar todas as variabilidades da LPS-BET utilizando aspectos.

O escopo desse trabalho restringe-se apenas às variabilidades "restrição de cartões" (Número de Cartões e Combinação de Cartões), encontradas nos produtos gerados para as cidades de Fortaleza, Campo Grande e São Carlos. Também é objetivo deste trabalho gerar novamente os produtos da LPS-BET usando a nova versão das variabilidades que foram implementadas usando aspectos. Este trabalho faz parte de um projeto de pesquisa que pretende gerar uma versão orientada a aspectos da LPS-BET, isto é, implementar todas as variabilidades utilizando aspectos.

A metodologia utilizada consiste inicialmente da investigação do trabalho Donegan(2008). Uma série de passos descritos por Donegan(2008), teve que ser realizada para a preparação do ambiente de desenvolvimento. Paralelamente ao estudo do trabalho de Donegan(2008) realizou-se o estudo do código da LPS-BET utilizando o ambiente integrado de desenvolvimento Eclipse. O estudo do código foi realizado inspecionando-se o sistema em tempo de execução linha a linha em diversos momentos.

Após esta etapa, foi feito um estudo sobre AspectJ e para tanto, buscou-se na documentação de seu mantenedor Eclipse o conhecimento necessário para a implementação dos aspectos. Estes estudos proveram a base necessária para a realização do projeto. As últimas etapas foram a implementação da variabilidade número de cartões com aspectos e a análise dos resultados alcançados.

Este trabalho está estruturado em capítulos. O Capítulo 2 apresenta os conceitos e termos utilizados por uma LPS e as etapas essenciais para sua construção. É feita também uma explanação sobre o gerador de aplicações Captor (Shimabukuro, 2006) e a programação orientada a aspectos. No Capítulo 3 aborda-se o desenvolvimento da LPS-BET, as decisões de projeto e os recursos utilizados. O Capítulo 4 trata das configurações do ambiente de desenvolvimento bem como a explicação para a utilização dos recursos, a implementação das variabilidades utilizando aspectos e as dificuldades encontradas. Por fim, o Capítulo 5 apresenta as conclusões.

2 Revisão Bibliográfica

Vários estudos focados na produção de software como parte de uma linha de montagem sempre abordam a importância do reúso de software. O reúso é uma técnica utilizada para minimizar o retrabalho e garantir qualidade do produto final, uma vez que as partes que são integradas ao novo software já foram testadas e validadas. Uma das maneiras sistemáticas de produção de software é a abordagem de LPS.

Nas próximas seções apresentam-se os conceitos básicos sobre os temas envolvidos nesse trabalho: LPS, gerador de aplicação (Captor) e programação orientada a aspectos.

2.1 Linha de Produto de Software (LPS)

Uma LPS permite à uma organização diminuir investimentos em diversas partes do seu processo produtivo como documentação, projeto, arquitetura, pessoas entre outras.

As empresas que adotam a LPS, visam entre outras coisas alcançar ou melhorar sua agilidade no mercado pois a abordagem possibilita que: seus produtos possam ser alterados ou criados com maior rapidez; a qualidade de seus produtos seja melhorada; a satisfação do cliente aumente; sua presença no mercado seja mantida(SEI, 2009).

A seguir, são apresentados alguns conceitos utilizados no decorrer deste trabalho.

* Núcleo ou ativo base: Artefato ou recurso que pode ser usado em um ou mais produtos de uma LPS. Pode ser uma arquitetura, um componente de software um modelo de processo, plano , documento ou qualquer outro resultado que possa ser utilizado para a construção de um sistema (SEI, 2009).

* Variabilidade: O conceito de variabilidade é um dos mais importantes no contexto de uma LPS. Todos os artefatos gerados dentro de uma LPS devem ser analisados com o intuito de detectar as variabilidades existentes entre um produto e outro. As representações das variabilidades devem ser uma preocupação constante da equipe, uma vez que ela pode

surgir em qualquer fase do desenvolvimento. Os produtos de uma LPS podem diferir uns dos outros pela arquitetura, componentes, comportamento etc.

* Característica: Característica de um produto que usuários e clientes consideram importante na descrição de membros de uma família de produtos(Griss, 2000 apud Donegan, 2008;p.8). As características podem se apresentar como comuns, opcionais ou alternativas.

Ao se construir uma LPS deve-se realizar 3 atividades essenciais(Clements e Northrop, 2001), descritas a seguir: Desenvolvimento do núcleo de artefatos, desenvolvimento do produto e gerenciamento da LPS. Nas subseções seguintes aborda-se cada uma delas.

2.1.1 Desenvolvimento do Núcleo de Artefatos

A atividade de desenvolvimento de núcleo de artefatos tem por objetivo estabelecer a capacidade de produção para os produtos e acontece dentro de um contexto situacional de restrições e recursos. A atividade do desenvolvimento do núcleo contempla as elicitações de requisitos, detecção das variabilidades, construção de artefatos reutilizáveis, modelo de domínio, arquiteturas a serem utilizadas dentre muitas outras decisões. Esta atividade é interativa e pode mudar o contexto. Como ilustrado na Figura 1, existem alguns fatores contextuais que influenciam no desenvolvimento do núcleo de artefatos.

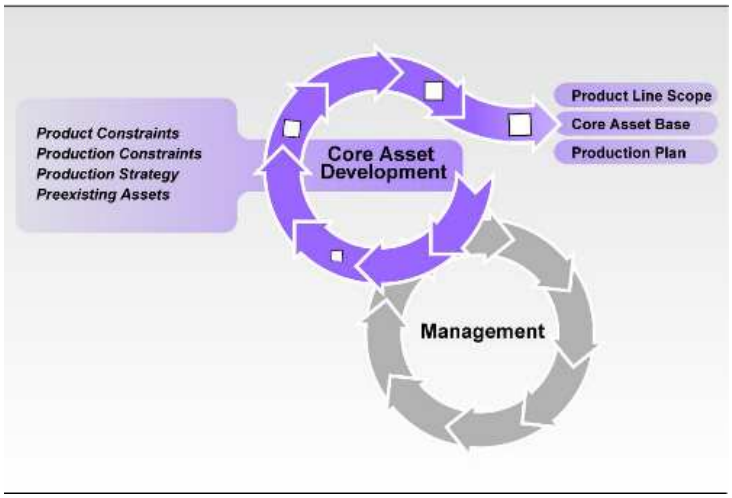

Figura 1: Desenvolvimento do núcleo de artefatos com suas entradas e saídas (SEI, 2009)

Para um melhor entendimento dos itens apresentados na Figura 1, a seguir apresenta-se algumas considerações.

Restrição de produto

A restrição de produto (*product constraint*) é o conjunto de características comuns e variantes e atributos comportamentais associados com os produtos, no âmbito da LPS que compreende questões como:

* Quais características comuns e variações existem entre os produtos que constituirão a LPS?
* Quais características comportamentais eles fornecem?
* Com quais sistemas externos o produto deve interagir?
* Quais restrições físicas devem ser observadas e quais requisitos de qualidade são impostos?

Estas restrições podem ser derivadas de um conjunto de produtos pré-existentes que irão formar a base para a LPS.

Restrição de produção

A Restrição de produção (*production constraint*) é qualquer restrição de tempo, ambiente de desenvolvimento, processo ou habilidade do desenvolvedo associado ao desenvolvimento de produtos em uma linha de produtos de software. Dentre as questões envolvidas nas restrições de produção estão:

* Em quanto tempo o produto será apresentado ao mercado (um dia, um mês ou um ano por exemplo)?
* Quem irá construir os produtos e que ambientes eles utilizarão?
* Que capacidade de produção deve ser dada aos engenheiros?

Estratégia de produção

A estratégia de produção (*production strategy*) é uma abordagem geral para a realização dos ativos base e produtos de uma LPS. Algumas questões encontradas nesta entrada são:

* A LPS será construída de forma pró-ativa ou reativa?

* Os componentes genéricos serão comprados ou desenvolvidos internamente?
* Como a produção do núcleo será gerenciada?
* Os produtos serão gerados automaticamente a partir do núcleo de ativos ou montados manualmente?

Ativos pré-existentes

Os ativos pré-existentes (*preexisting assets*) são soluções que podem ajudar no desenvolvimento de uma LPS. Produtos de código aberto, *frameworks*, *Web Services* e padrões são apenas alguns dos exemplos de ativos pré-existentes. A seguir são mostradas as questões discutidas nessa atividade:

* Que software e ativos organizacionais estão disponíveis?
* Existem bibliotecas, frameworks, ferramentas, componentes e serviços que podem ser utilizados?
* Existem processos de gerenciamento técnico e recursos de treinamento que podem ser adaptados facilmente à LPS?

2.1.2 Desenvolvimento do produto

É nesta atividade que os produtos são gerados. Os componentes são utilizados para se produzir um novo software. Nesta etapa, com base em um modelo de domínio, ajustes serão feitos para atender as necessidades específicas do cliente. Caso alguma necessidade não tenha sido coberta pelo modelo de domínio, será considerada como um novo requisito. O desenvolvimento do produto é dependente da saída da atividade de desenvolvimento do núcleo de artefatos juntamente com a descrição dos produtos a serem desenvolvidos como ilustra a Figura 2.

As entradas para a atividade de desenvolvimento do produto são:

* A descrição do produto para um produto particular;
* O escopo da LPS;

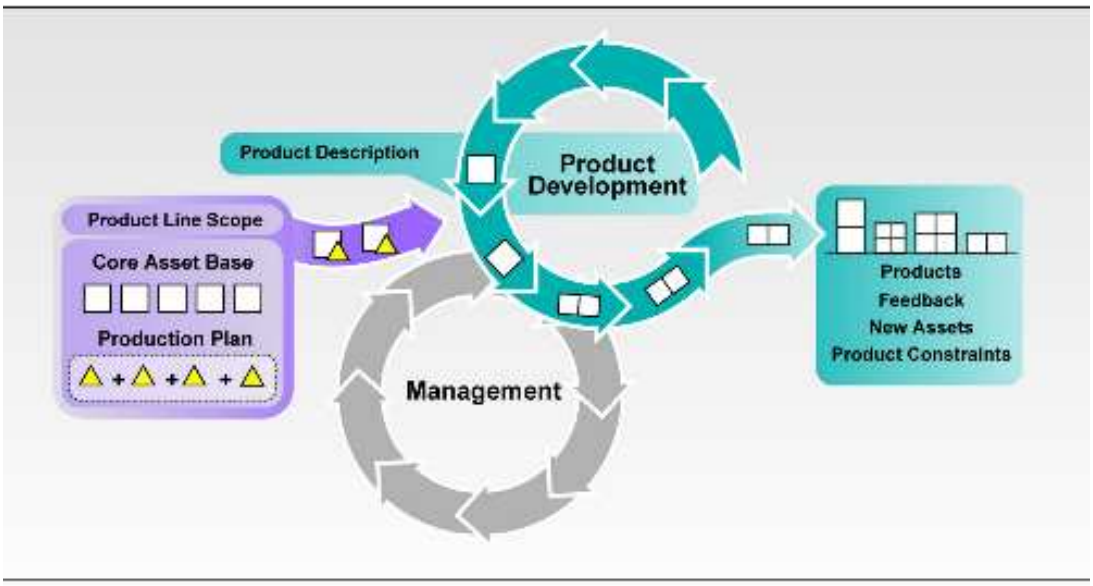

Figura 2: Desenvolvimento do produto (SEI, 2009)

* O núcleo de artefatos a partir do qual os produtos são construídos;

* O plano de produção.

O escopo da LPS (*Product Line Escope*) é uma descrição dos produtos que constituem uma linha de produto ou que a linha de produto é capaz de incluir; O núcleo de ativos base (*Core Asset Base*) é o conjunto completo dos ativos essenciais associados a uma determinada linha de produtos de software; O plano de produção (*Production Plan*) é um quia de como os produtos em uma LPS serão construídos a partir dos ativos base da linha de produto; O gerenciamento (*Management*) é a atividade de uma LPS que visa garantir que todos os recursos técnicos sejam aplicados de forma coordenada.

2.1.3 Gerenciamento de LPS

Para que uma LPS tenha sucesso efetivo, a atividade de gerenciamento deve ser desenvolvida continuamente durante toda a sua vida sob pena de se perder altos investimentos, tanto econômicos quanto de tempo. O gerenciamento técnico e o gerenciamento organizacional constituem os níveis que devem estar comprometidos com a continuidade da LPS. O gerenciamento organizacional é responsável por prover recursos de maneira eficiente para as unidades organizacionais envolvidas na manutenção da linha de produto de software. Para tanto, uma estrutura organizacional deve ser criada. O gerenciamento técnico é responsável por supervisionar as atividades de desenvolvimento do núcleo de artefatos e de produtos de maneira a garantir que

os requisitos definidos para a LPS estejam sendo obedecidos.

2.2 Captor

Um gerador de aplicação é uma ferramenta responsável por produzir artefatos que farão parte de uma aplicação para algum domínio. Por outro lado, um gerador de aplicação configurável pode ser configurado de maneira a atender mais de um domínio. O gerador de aplicação configurável Captor (do inglês *Configurable Application Generator*) foi inicialmente projetado para ser um gerador configurável por linguagens de padrões e, na primeira fase do desenvolvimento, seus requisitos foram definidos com base no gerador de aplicação específico GREN-Wizard (Braga e Masiero, 2003, apud Donegan, 2008;p.28).

> Terminado o desenvolvimento do gerador configurável por linguagem de padrões, os autores decidiram que a ferramenta deveria ser configurada não apenas por linguagens de padrões, mas também por outras linguagens de modelagem de aplicações que podem ser definidas para domínios específicos. Esse processo levou a uma segunda fase de desenvolvimento, que culminou na criação do gerador de aplicação configurável Captor e na definição das atividades das etapas de engenharia de domínio e engenharia de aplicação com geradores configuráveis (Shimabukuro, 2006;p.58).

O Captor foi projetado para que sua GUI (*Graphical User Interface*) assuma uma aparência de acordo com as especificações do domínio. Sua aparência é definida por informações contidas em um arquivo XML produzidas pelo engenheiro de domínio. Neste arquivo são definidos a quantidade de formulários que poderão ser apresentados, quais são as regras de validação para cada campo e também a organização dos formulários. O Captor lê o arquivo XML e apresenta os elementos (campos, caixas de texto, caixas de seleção entre outros) por meio de formulários (Figura 3) que poderão ser editados.

A partir deste momento, o engenheiro de aplicação utiliza as informações apresentadas pela GUI para gerar uma linguagem de modelagem de aplicações (LMA) no formato XML, como pode ser visto na Figura 4.

O Captor foi utilizado para gerar os produtos da LPS-BET. Em um primeiro momento, o Captor foi configurado para produzir um núcleo com as características comuns da LPS. Em seguida, foram identificadas as características específicas para cada uma das aplicações-referência (Campo Grande, Fortaleza e São Carlos).

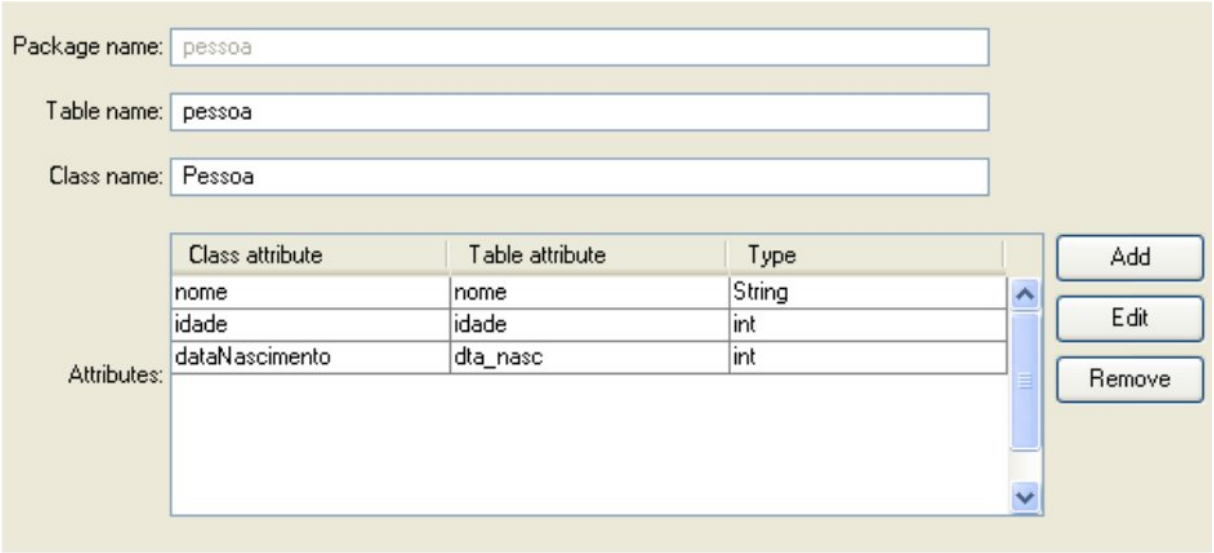

Figura 3: Representação da LMA por meio de formulários (SHIMABUKURO, 2006;p.42)

```
<persistencia>

  <entidade>
    <nome_tabela>pessoa</nome_tabela>
    <nome_classe>Pessoa</nome_classe>
    <atributos>
      <atributo tipo="string">
        <att_tabela>nome</att_tabela>
        <att_classe>nome</att_classe>
      </atributo>
      <atributo tipo="int">
        <att_tabela>idade</att_tabela>
        <att_classe>idade</att_classe>
      </atributo>
      <atributo tipo="date">
        <att_tabela>dta_nasc</att_tabela>
        <att_classe>dataNascimento</att_classe>
      </atributo>
    </atributos>
  </entidade>

</persistencia>
```

Figura 4: LMA em formato XML (SHIMABUKURO, 2006;p.43)

2.3 Programação Orientada a Aspectos

A programação orientada a aspectos torna o desenvolvimento de software mais modularizado principalmente quanto aos requisitos não-funcionais. Linguagens orientadas a objeto enfrentam dificuldades para cobrir todos os interesses do sistema, sobretudo aqueles relacionados aos requisitos não-funcionais. Como exemplo, pode-se citar as operações de *log* onde o código é espalhado por todo o sistema a fim de realizar tal operação. Supondo uma classe cliente com atributos típicos de um cliente, em algum momento da execução do programa, seu

estado é atualizado e, toda vez que isso acontece, deseja-se que seja gravado em um arquivo de *log* seu novo estado. Não é atribuição da classe cliente realizar tal operação porém ela o faz.

Ao se utilizar a programação orientada a aspectos pode-se encapsular os "interesses" de forma a realizar as operações por meio de construções específicas. Dessa maneira o código fica inteiramente limpo. Os aspectos especificam pontos bem definidos onde a interceptação da execução do programa deve ser feita e o que fazer quando ela ocorrer. Com esse recurso, cada classe do sistema contém apenas o que é de fato seu e dessa maneira permite uma melhor manutenção.

AspectJ (Kiczales *et. al*.,2001) é uma extensão orientada a aspectos para a linguagem orientada a objetos Java e introduz o conceito de pontos de junção e alguns novos elementos de programa para o Java. A Linguagem AspectJ é dotada de algumas construções que permitem ao programador especificar em quais pontos do programa as execuções irão ocorrer sem ter que mexer em uma única linha de código do programa alvo.

Os aspectos são similares a classes Java, porém, com algumas características que os distinguem. Eles basicamente se diferenciam das classes Java por possuirem adendos (*advice*), pontos de atuação ou conjuntos de junção (*pointcut*) e pontos de junção (*join point*). O conjunto destas construções dão sentido ao aspecto onde estão inseridas. A combinação deste conjunto de comandos, pode ser feita com o uso de alguns operadores lógicos já conhecidos de algumas linguagens como "`||`" (OR), "`&&`" (AND) e "!" (NOT). A flexibilidade é aumentada com o uso de *wildcards* como "*", que permite referenciar os pontos de junção, contidos em pontos de atuação, de forma coletiva (dando um significado de "qualquer"). Cada uma destas construções com os operadores e o *wildcard*, estão devidamente explicadas e exemplificadas a seguir:

* Ponto de junção (*join point*): Os pontos de junção são pontos bem definidos dentro do programa e que determinam dentro de um aspecto, o exato momento onde deve haver a interceptação do fluxo de execução do programa. Pode-se considerar pontos de junção chamadas a métodos, execução de métodos, inicialização de objetos etc.

* Ponto de atuação (*pointcut*): Os pontos de atuação tem a função de agrupar os pontos de junção dentro de um aspecto. Estes pontos de atuação podem ser nomeados ou não. A seguir, as Figuras 8 e 9 mostram exemplos para esclarecer o exposto neste item;

* Adendo (*Advice*): Usado para implementar os comportamentos transversais. É composto por ponto de atuação e o código que é executado assim que cada ponto de junção é alcançado. Os adendos podem ser do tipo *before*, *after* e *around* e determinam o momento em que a execução do código ocorrerá. A seguir, alguns exemplos mostrando a

sintaxe dos adendos utilizando cada um dos adendos anteriormente mencionados:

```
1  before(): move() {
2      System.out.println("about to move");
3  }
```

Figura 5: Adendo *before* (Eclipse, 2009)

A Figura 5 mostra de forma simples a sintaxe de um adendo *before*. Pode-se ler o código na Figura 5 da seguinte maneira: Antes de executar os comandos contidos dentro do ponto de atuação nomeado 'move()', execute o comando `System.out.println("about to move")`.

Em alguns casos pode ser necessário que ao término do processamento de um método em que haja algum retorno, seja iniciado outro processamento. Neste caso, os adendos podem ser usados como mostrado na Figura 6. A sintaxe é muito simples e intuitiva pois pode-se ler como o exemplo mostra: 'Após ter executado todos os comandos que estão dentro do ponto de atuação nomeado move(), execute o comando System.out.println("just successfuly moved")';

```
1  after() returning: move(){
2      System.out.println("just successfully moved");
3  }
```

Figura 6: Adendo *after* (Eclipse, 2009)

O adendo *around* mostrado na Figura 7 tem a função de direcionar o fluxo de execução quando ocorrer uma chamada para o método "void setX(int)". Quando a chamada ocorrer, será avaliada a condição "p.assertX(x)" que permitirá que o método "void setX(int)" seja executado ou não. Enquanto os adendos citados anteriormente executam o código referido pelo ponto de junção marcado como *before* ou *after*, o adendo *around* pode devolver ou não a execução para o ponto de junção dependendo das condições impostas em seu código. A execução do ponto de junção especificado, será feita apenas se houver uma chamada para o comando *proceed*.

```
1  void around(Point p, int x): target(p)
2              && args(x)
3              && call(void setX(int)){
4      if(p.assertX(x)) proceed(p,x);
5      p.releaseResources();
6  }
```

Figura 7: Adendo *around* (Eclipse, 2009)

A Figura 8 mostra um conjunto de *pointcuts* unidos por operadores lógicos "**OU**". Neste caso, a interceptação no código acontecerá no momento da chamada à qualquer um dos métodos especificados (`FigureElement.setXY(int,int)`, `Point.setX(int)`, `Point.setY(int)`, `Line.setP1(Point`, `Line.setP2(Point)`). O método chamado, deve ter a mesma assinatura descrita no *pointcut*.

```
1  call(void FigureElement.setXY(int,int)) ||
2  call(void Point.setX(int))              ||
3  call(void Point.setY(int))              ||
4  call(void Line.setP1(Point))            ||
5  call(void Line.setP2(Point));
```

Figura 8: *Pointcut* não-nomeado (Eclipse, 2009)

A organização do código dentro de um aspecto é facilitada com a utilização de pontos de atuação nomeados. Porém, o fato de a linguagem oferecer este recurso, não desobriga o desenvolvedor de criar os aspectos de acordo com as características comuns onde devem atuar.

```
1  pointcut move():
2     call(void FigureElement.setXY(int,int)) ||
3     call(void Point.setX(int))              ||
4     call(void Point.setY(int))              ||
5     call(void Line.setP1(Point))            ||
6     call(void Line.setP2(Point));
```

Figura 9: *Pointcut* nomeado (Eclipse, 2009)

Além das construções já citadas, os aspectos também contam com um conjunto de palavras reservadas mais comumentes conhecidas como designadores, cada um com um propósito específico. A seguir, a Tabela 1 mostra alguns dos designadores e sua respectiva função.

Os aspectos, como as classes Java, são instanciados para serem usados, porém, a instanciação do aspecto fica a cargo da linguagem AspectJ. No caso das classes Java, a instanciação é feita utilizando-se a palavra reservada `new`. Para que os aspectos possam atuar em conjunto com um programa escrito em Java, é necessário fazer a combinação dos aspectos com as classes que sofrem as interceptações. Esse processo de combinação é denominado *Weaver* ou combinador aspectual. *Weaver* é o mecanismo responsável por dirigir o fluxo de controle normal para um adendo, quando o ponto de execução do programa está em um ponto de junção. A Figura 10 mostra a estrutura de um aspecto. O aspecto representado pela Figura 10 contém dois métodos (linhas 5 e 9) denominados `reportFault` e `fixServer`. Além dos métodos, também fazem parte do aspecto um *pointcut* (linha 13), um adendo *before* (linha 15) e um adendo *after* (linha

Tabela 1: Designadores da linguagem AspectJ

Designador	**Característica**
call(*signature*)	chamada a métodos ou construtores referentes a assinatura *signature*
execution(*signature*)	execução de cada método ou construtor referentes a assinatura *signature*
get(*signature*)	acesso ao atributo identificado em *signature*
set(*signature*)	atribuição de valor ao atributo indenfiticado em *signature*
this(*type pattern*)	referencia ao objeto que contém a execução naquele instante
target(*type pattern*)	seleciona o ponto de junção onde o objeto alvo é indicado por *type*
args(*type pattern*)	seleciona o ponto de junção onde os argumentos são indicados por *type*
within(*type pattern*)	seleciona o ponto de junção onde a execução do código é definida pelo tipo indicado por *type*

19). Este aspecto fará a interceptação sempre que um método com visibilidade *public* dentro do projeto for chamado. O adendo *before* fará a interceptação antes da chamada aos métodos publicos do projeto. Após ter seu código executado, o adendo *before* devolve a execução para o método chamado. Se durante a execução do método ocorrer alguma exceção, o adendo *after* fará a interceptação, executando seu código.

```
 1 aspect FaultHandler {
 2
 3   private boolean Server.disabled = false;
 4
 5   private void reportFault() {
 6     System.out.println("Failure! Please fix it.");
 7   }
 8
 9   public static void fixServer(Server s) {
10     s.disabled = false;
11   }
12
13   pointcut services(Server s): target(s) && call(public * *(..));
14
15   before(Server s): services(s) {
16     if (s.disabled) throw new DisabledException();
17   }
18
19   after(Server s) throwing (FaultException e): services(s) {
20     s.disabled = true;
21     reportFault();
22   }
23 }
```

Figura 10: Anatomia de um aspecto (Eclipse, 2009)

3 Desenvolvimento da LPS-BET

Este capítulo está estruturado em seções. A Seção 3.1 apresenta brevemente a LPS-BET, sua finalidade e seu funcionamento. Também é apresentada nesta seção uma visão de como se deu o processo de desenvolvimento da LPS-BET. A Seção 3.2 trata das decisões de projeto da LPS-BET, vantagens e desvantagens da utilização de componentes caixa-branca e caixa-preta além de trechos de código de aspectos implementados por Donegan(2008). Por fim, na Seção 3.3 são apresentados os recursos utilizados para a implementação da LPS-BET.

3.1 LPS-BET

Donegan(2008) aborda em sua pesquisa a geração de uma família de produtos de software com arquitetura baseada em componentes, onde os componentes podem ser do tipo caixa preta ou caixa branca. Donegan (2008) ainda expõe as implicações de se desenvolver produtos de software utilizando tais componentes e em certo ponto de seu estudo faz uso de aspectos juntamente com componentes para implementar algumas das variabilidades encontradas. Uma LPS não é algo simples de se desenvolver, exige muito trabalho de seus engenheiros e desenvolvedores e não fogem das boas práticas para o desenvolvimento de software como a elicitação de requisitos e a geração de artefatos em cada fase do ciclo de desenvolvimento.

A LPS-BET tem por finalidade facilitar a utilização do transporte coletivo de uma cidade visando o conforto e a comodidade para os munícipes. O software consiste de um computador central que se comunica com os terminais que possuem leitores de cartão magnético, localizados nos ônibus em circulação por meio de rádio-frequência. Neste computador estarão armazenados todos os dados referentes aos passageiros e também sobre os itinerários dos ônibus. Os usuários interagem com o sistema por meio de cartões magnéticos e, com isso, usufruem de vantagens como: recarregar seu cartão em pontos específicos da cidade determinados pela empresa de transporte e também fazer integração pagando apenas uma passagem utilizando-se de dois ônibus, obviamente obedecendo certas regras previamente estipuladas. O desenvolvimento da LPS-BET foi focado no sistema BET das cidades de São Carlos(SP),

Fortaleza(CE) e Campo Grande(MS).

Donegan (2008) decidiu por utilizar um processo de engenharia avante uma vez que não foram encontradas aplicações para o mesmo domínio que pudessem ser tomadas como referência. Quanto a estratégia de produção, foi utilizado a evolução pró-ativa.

> No caso de uma evolução pró-ativa, a organização pode usar um processo baseado em engenharia reversa ou em engenharia avante que diferem basicamente na sua primeira fase. No processo baseado em engenharia avante uma nova LPS é desenvolvida e as funcionalidades comuns podem ser determinadas antes das funcionalidades variáveis. Na engenharia reversa existem sistemas que estão disponíveis para análise e modelagem e são candidatos para modernização e inclusão em uma LPS (Gomaa,2004 apud Donegan,2008,p.9).

Os requisitos elicitados para a LPS-BET foram minuciosamente analisados e comparados para poder representar as possíveis variabilidades que viriam a ocorrer. Foram utilizados ciclos de incrementos horizontais durante o processo de desenvolvimento com o objetivo de se ter uma aplicação completa logo no início do processo. Com base nos requisitos elicitados, um diagrama de características foi modelado permitindo que uma definição do cronograma e das estimativas de tempo, esforço e escopo dos incrementos pudessem ser elaboradas mais facilmente. Este diagrama de características inicialmente foi modelado para o núcleo da LPS-BET (Figura 11) e no decorrer das execuções dos incrementos, modelou-se o diagrama de características da LPS-BET como um todo, como mostra a Figura 12.

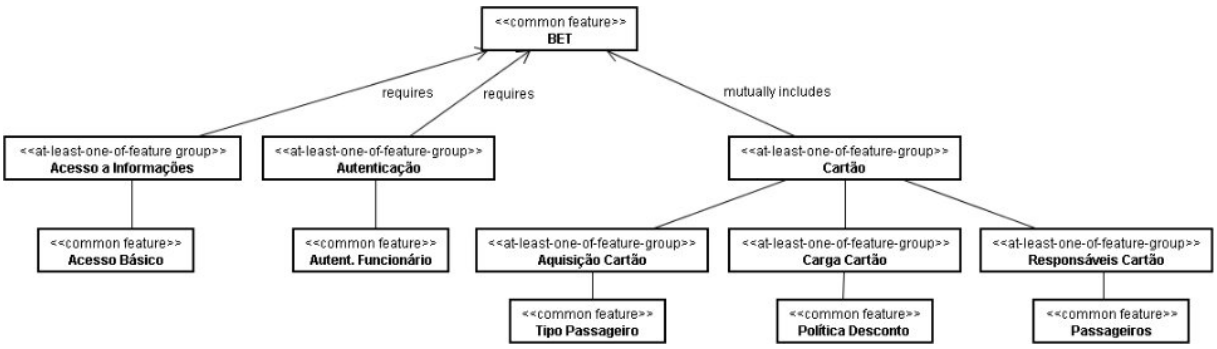

Figura 11: Diagrama de Característica do Núcleo da LPS-BET (Donegan, 2008;p.59)

A Figura 13 mostra os incrementos realizados durante o desenvolvimento. O incremento 1 contempla os requisitos elicitados das três aplicações-referência e um planejamento geral do desenvolvimento da LPS. O núcleo, por sua vez, foi planejado de forma detalhada. Desse modo, baseando-se nos requisitos elicitados, foi possível modelar o diagrama de casos de uso da LPS como mostra a Figura 14.

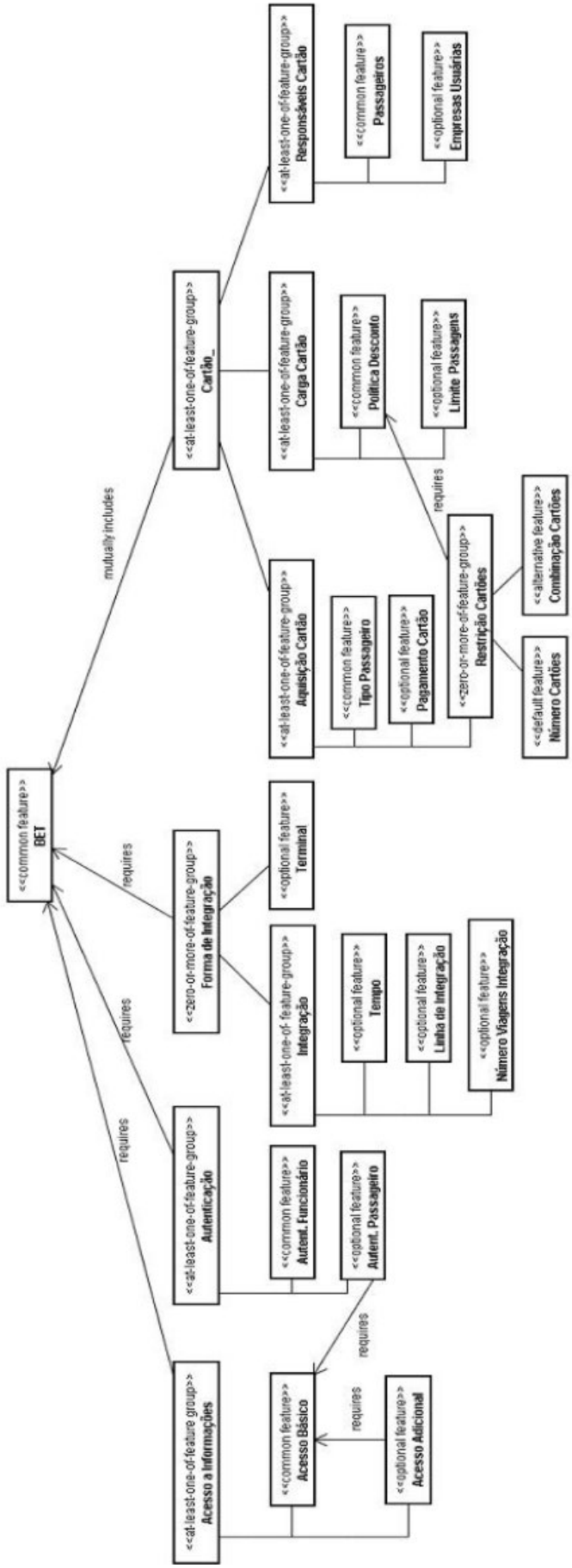

Figura 12: Diagrama de Característica para a LPS-BET (Donegan, 2008;p.157)

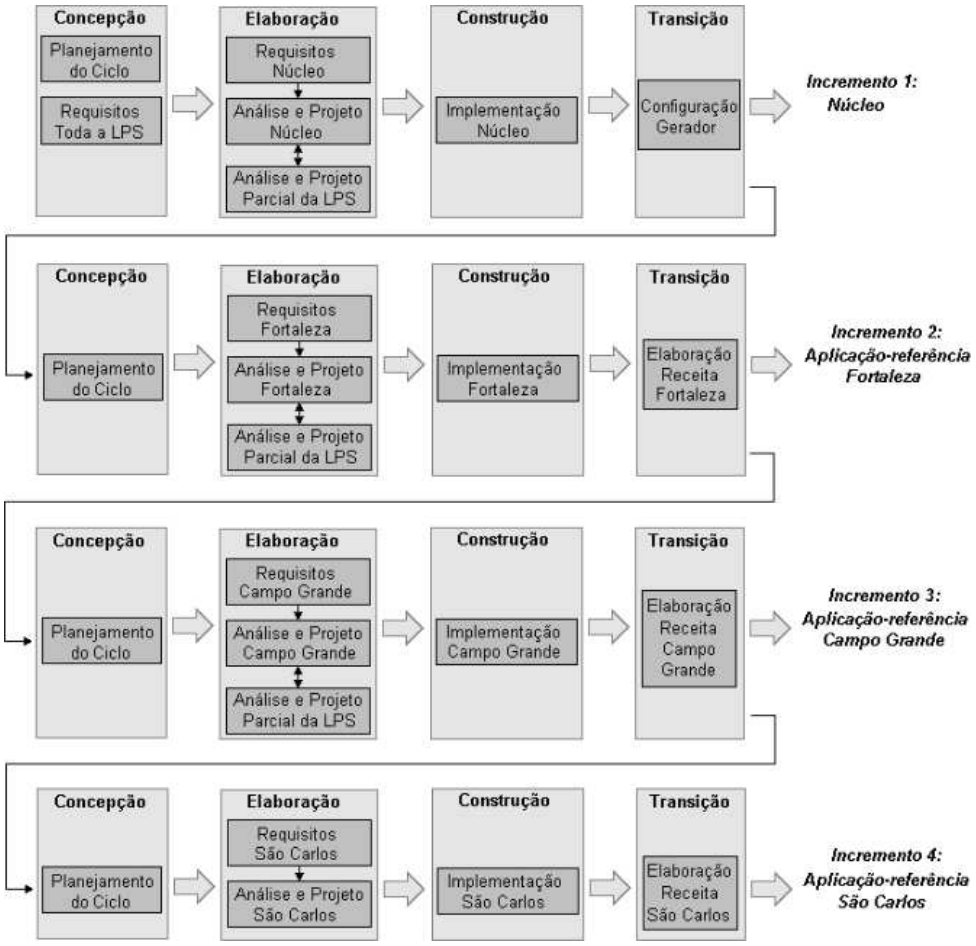

Figura 13: Visão geral do processo de desenvolvimento da LPS-BET (Donegan, 2008;p.57)

No incremento 2 o planejamento do desenvolvimento pôde ser feito em detalhes fazendo reúso do incremento 1 tornando o processo mais ágil. A partir daí, adotou-se o mesmo processo para a realização dos incrementos posteriores, ou seja, reusando os incrementos anteriores e refinando as necessidades para o incremento atual.

Ao final de cada incremento, na etapa de transição, elaborou-se uma receita para que os produtos fossem gerados utilizando o gerador de aplicações configurável Captor.

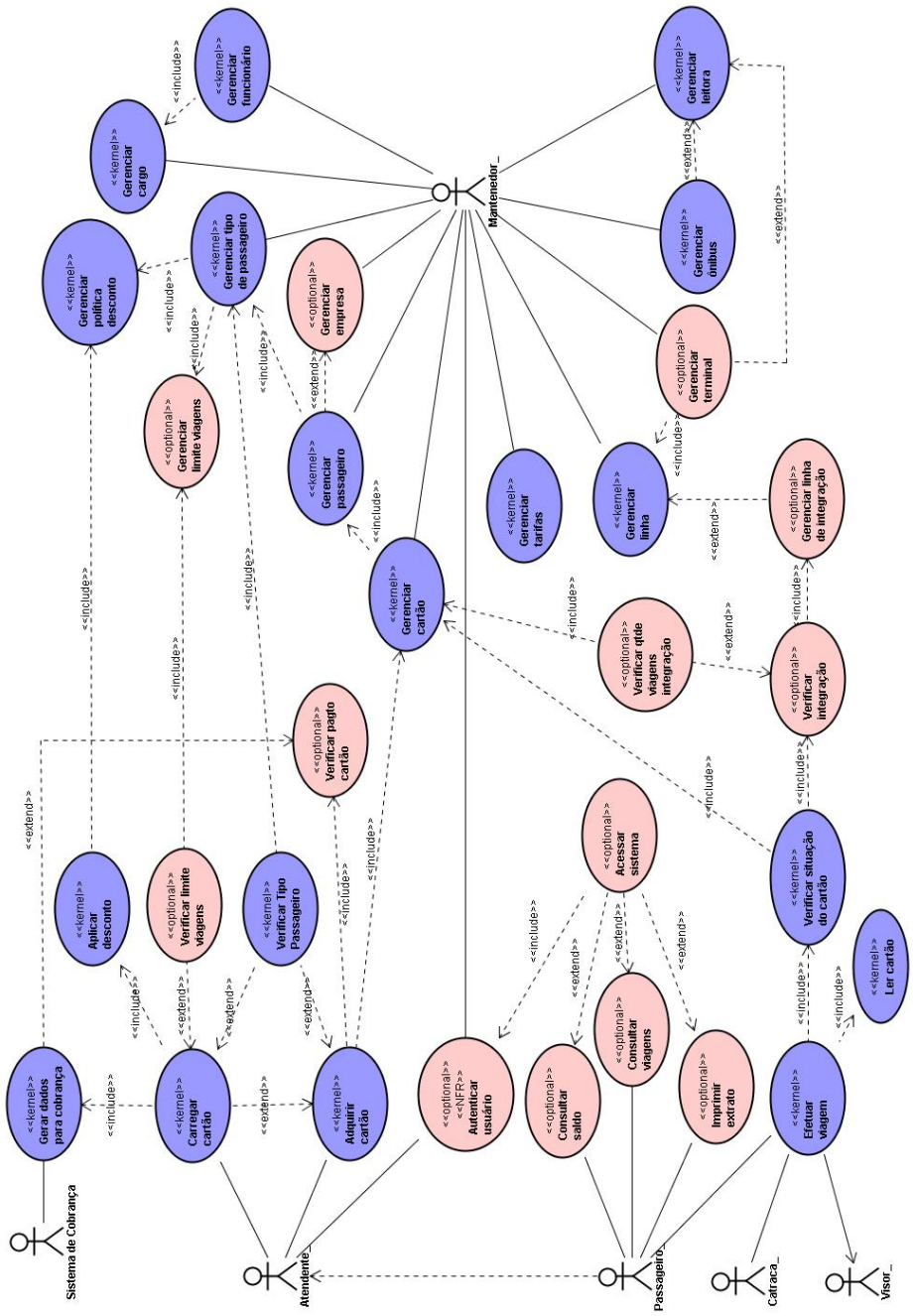

Figura 14: Diagrama de Casos de Uso da LPS-BET Donegan (Donegan,2008;p.156)

A Figura 15 mostra a arquitetura dos componentes do núcleo da LPS-BET e as ligações entre eles. Os componentes no sistema são identificados como sendo de controle (possuem terminação "Ctrl") (Gomma, 2004 apud Donegan, 2008, p. 61) e de negócio (possuem terminação "Mgr") (Chessman e Daniels, 2001 apud Donegan, 2008, p. 61). A variabilidade alvo deste trabalho (Número de Cartões) é parte integrante do componente Aquisição Cartão.

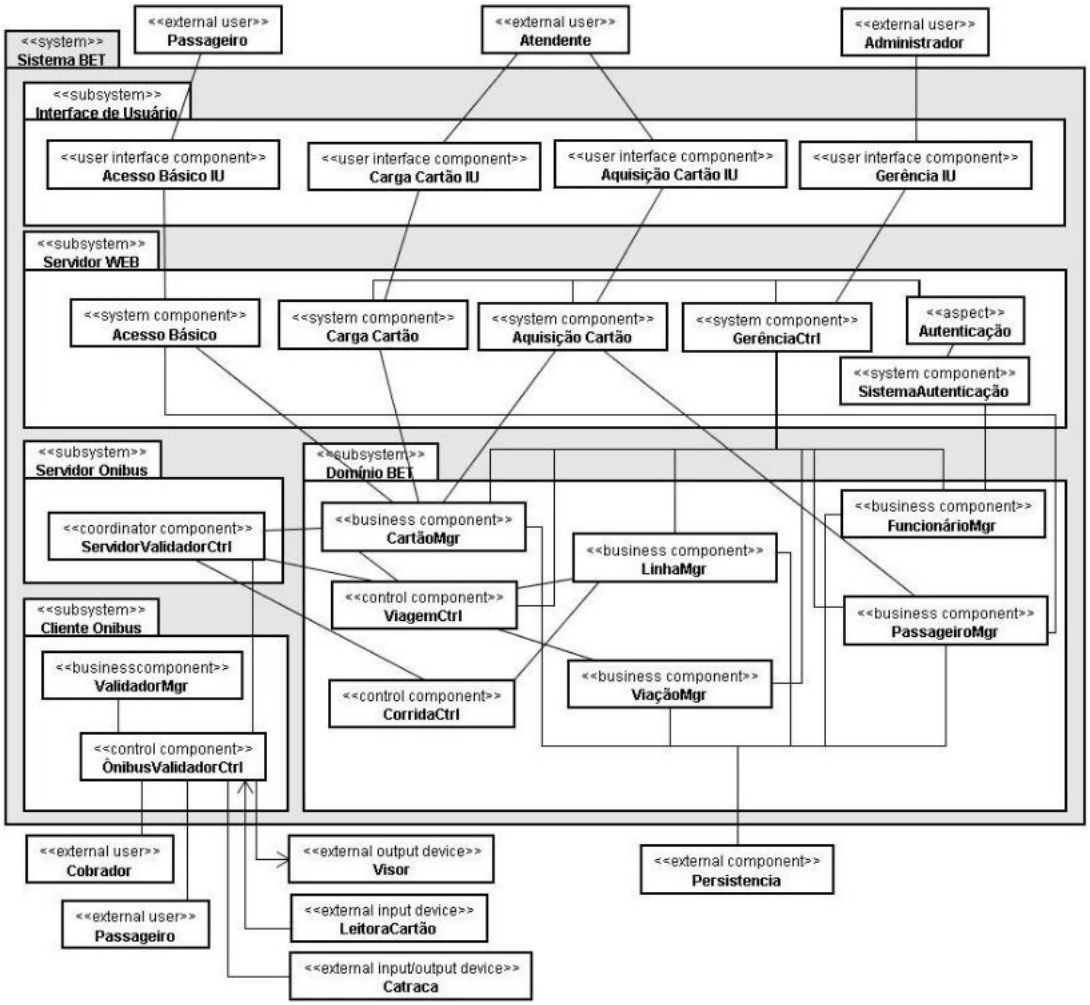

Figura 15: Arquitetura de componentes do núcleo da LPS-BET (Donegan,2008;p.63)

3.2 Decisões de projeto da LPS-BET

Para o desenvolvimento da LPS-BET foram utilizados componentes caixa-preta visando analisar seu impacto na agilidade do desenvolvimento que era uma das metas de Donegan quando seu trabalho foi desenvolvido. Também, decidiu utilizar aspectos para implementar algumas variabilidades mesmo não sendo esse o objetivo principal do projeto. Os aspectos foram utilizados para uma breve investigação das vantagens e desvantagens de se implementar tais variabilidades utilizando componentes e aspectos.

Donegan utilizou um projeto com componentes caixa-preta visando uma maior separação de interesses. Tal separação fica evidente quando novas funcionalidades surgem para atender novas especificações de requisitos e são encapsuladas em novos componentes que por sua vez, são específicos para a nova variabilidade. Isso é exemplificado pela Figura 16. Supondo que um componente A está funcionando perfeitamente e é parte integrante de um produto que está em produção. No momento em que for detectada a necessidade de novas funcionalidades, estas seriam encapsuladas em um componente B que poderia, juntamente com o componente A, formar um componente C que tiraria proveito de todas as funcionalidades de A e B. Dessa forma, todos os componentes poderiam fazer parte de qualquer produto da LPS mantendo suas funcionalidades intactas.

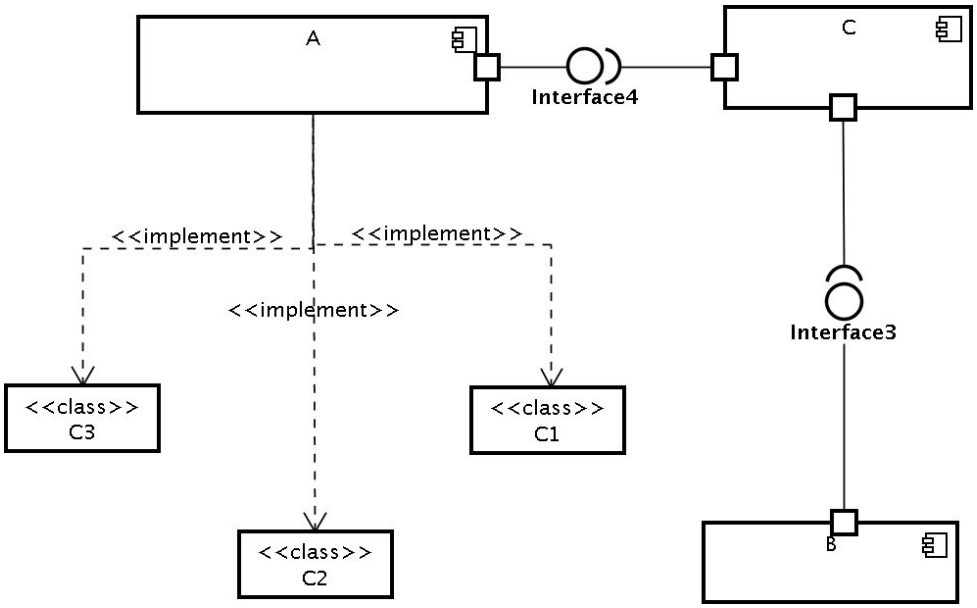

Figura 16: Exemplo de separação de interesses utilizando componentes caixa-preta

Uma solução utilizando componentes caixa-branca permitiria a injeção de código em classes pré-existentes e desta forma permitiria a duplicação de código e consequentemente dificultaria manutenções futuras. A Tabela 2 mostra algumas vantagens e desvantagens de cada abordagem.

Tabela 2: Vantagens e desvantanges no uso de componentes do tipo caixa-branca ou caixa-preta (Donegan, 2008;p.82)

Tipo de componente	**Vantagens**	**Desvantagens**
Caixa-branca	Facilidade de implementação Facilidade de composição	Duplicação de código Dificuldade de manutenção
Caixa-preta	Maior separação de interesses Facilidade de manutenção	Maior interação entre componentes Pior desempenho

Em seu trabalho, Donegan(2008) abordou o uso de aspectos para a implementação de requisitos não-funcionais e também algumas variabilidades da LPS-BET. Os requisitos não-funcionais escolhidos foram o de autenticação (*Autenticacao*) e o de autorização (*Autorizacao*) como pode ser visto na Figura 17. Tal uso se justifica pois se não fosse utilizado aspectos para este fim, haveria código espalhado e entrelaçado dentro da LPS. Isto ocorreria, por exemplo, quando operações realizadas por componentes distintos necessitassem da autenticação do usuário. Sem aspectos, cada componente deveria tratar esta situação.

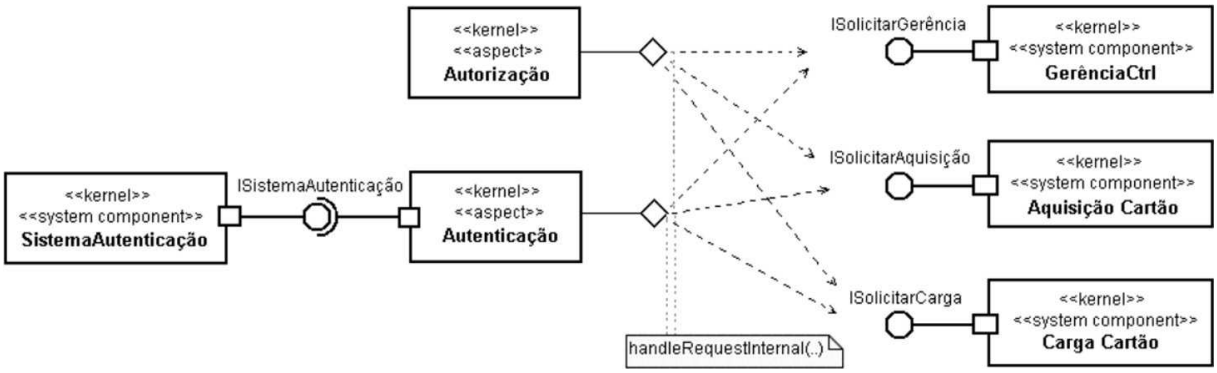

Figura 17: Arquitetura dos aspectos *Autenticacao* e *Autorizacao* e dos componentes entrecortados (Donegan, 2008;p.94)

O aspecto *Autenticacao* intercepta no momento da execução de todos os métodos *handleRequestInternal* com qualquer tipo de parâmetro, desde que não pertençam as classes GerenciarLogin e AcessoBasico independente do tipo de retorno. Isto porque, segundo Donegan(2008;p.93) "no núcleo não deve haver autenticação de passageiro". Esta interceptação é feita por meio do conjunto de ponto de junção *acessarAutenticacao()* como pode ser visto nas linhas 4, 5 e 6 da Figura 18.

O exemplo utilizado por Donegan(2008) para demonstrar o uso de aspectos para implementar variabilidades, contempla as características do grupo de características *Integração* (*Tempo, Linha Integrada e Número de Viagens*). Uma similaridade encontrada foi a necessidade de se verificar a existência de uma integração. Esta necessidade representada por meio de um método chamado *verificarIntegracao()* é realizada por cada componente de formas diferentes, apesar de terem a mesma assinatura.

A solução proposta por Donegan(2008) foi criar um aspecto abstrato *IntegracaoCtrl* e um aspecto para cada componente (*TempoViagemCtrl, NumeroViagemCtrl e LinhaIntegradaViagemCtrl*) como apresentado na Figura 19 em que *IntegracaoCtrl* é uma espécie de "super-aspecto[1]".

[1]Uma analogia a super-classe em se tratando de Java

```
1   public aspect Autenticacao {
2    ISistemaAutenticacao interfaceAutenticacao;
3
4    pointcut acessarAutenticacao(): execution(* lps.bet..*.handleRequestInternal(..)) &&
5       !execution(* lps.bet.basico.web.autenticacao.GerenciaLogin.handleRequestInternal(..)) &&
6       !execution(* lps.bet.basico.web.acessoBasico.AcessoBasico.handleRequestInternal(..));
7
8    before() : acessarAutenticacao (){
9     Object[] args = thisJoinPoint.getArgs();
10    if (args.length > 0){
11      HttpServletRequest request = (HttpServletRequest) args[0];
12      HttpServletResponse response = (HttpServletResponse) args[1];
13      HttpSession sessao = request.getSession(false);
14      if (sessao == null)
15        response.sendRedirect("login.html");
16      else {
17        String login = (String) sessao.getAttribute("login");
18        if(!interfaceAutenticacao.estaAutenticado(login) || interfaceAutenticacao.estaExpirada(request))
19          response.sendRedirect("login.html");
20        else
21          interfaceAutenticacao.atualizarSessao(request);
22       }
23     }
24   }
25  }
```

Figura 18: Implementação do aspecto *Autenticacao* (Donegan, 2008;p.94)

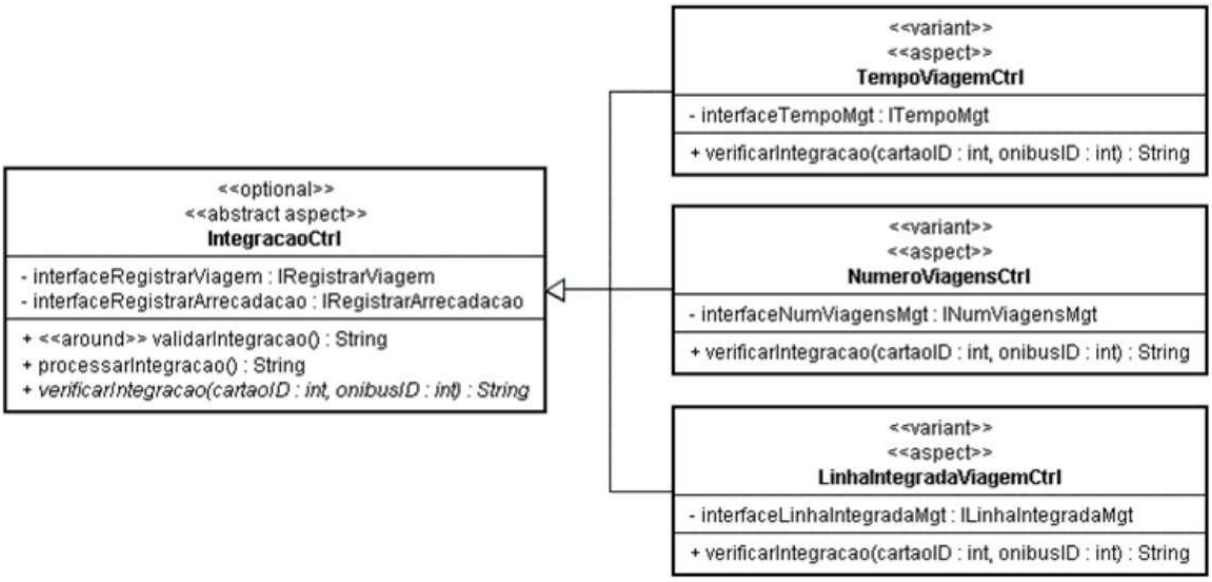

Figura 19: Aspecto abstrato e aspectos contratos para representar a característica *Integracao* (Donegan, 2008;p.96)

Dessa maneira, cada funcionalidade é implementada em seu respectivo aspecto. O aspecto *IntegracaoCtrl* é abstrato pois contém um método abstrato (*verificarIntegracao()*) que será implementado por seus aspectos filho. A interceptação ocorre no momento da chamada a quaisquer métodos da interface IProcessarViagem como pode ser visto na linha 5 da Figura 20.

```
1   public abstract aspect IntegracaoCtrl {
2    ICartaoMgt interfaceCartaoMgt;
3    IRegistrarArrecadacao interfaceRegistrarArrecadacao;
4
5   pointcut validarIntegracao():call(String lps.bet.interfaces.IProcessarViagem.*(..));
6
7    String around(): validarIntegracao(){
8      String estado="INT-NOK";
9      Object[] args = thisJoinPoint.getArgs();
10     if (args.length > 0){
11       int cartaoID = (Integer) args[0];
12       int onibusID = (Integer) args[1];
13       estado = verificarIntegracao(cartaoID onibusID);
14     }
15     return !estado.equals("INT-OK") ? proceed(): estado;
16   }
17
18   public String processarIntegracao(int onibusID Viagem viagem){
19     interfaceRegistrarArrecadacao.registrarArrecadacao(onibusID 0);
20     viagem.setNumViagens(viagem.getNumViagens()+1);
21     interfaceCartaoMgt.alterarViagem(viagem);
22     return "INT-OK";
23   }
24
25   public abstract String verificarIntegracao(int cartaoID int onibusID);
26  }
```

Figura 20: Implementação do aspecto abstrato IntegraçãoCtrl (Donegan, 2008;p.97)

3.3 Tecnologias de Implementação da LPS-BET

Para a construção da LPS-BET, Donegan (2008) fez uso de alguns recursos os quais a ajudaram durante o processo como mostra a Tabela 3.

Tabela 3: Lista de recursos utilizados durante o processo de construção

Finalidade	**Recurso**
Modelagem	Notação UML na ferramenta JUDE Community
Controle de versão	TortoiseSVN
Codificação (ferramenta)	Eclipse
Codificação (Linguagem)	Java e AspectJ
Persistência	PostgreSQL
Framework (persistência)	Hibernate
Framework (codificação)	Spring
Gerenciador de dependências	Maven

A seguir descreve-se brevemente cada um dos recursos mencionados na Tabela 3.

- **JUDE/Community**(Jude, 2009) é uma ferramenta de modelagem UML gratuita com funcionalidades básicas. Ela oferece características como edição e impressão de diagramas UML2.0, importação/exportação de código fonte Java, saída de gráficos;

- **Eclipse** (Eclipse, 2009) é uma comunidade *open source* que tem por objetivo construir uma extensa plataforma de desenvolvimento para realizar a construção, instalação e gerenciamento do software durante todo o seu ciclo de vida. Eclipse é conhecida por muitos como uma *Integrated Development Environment* (IDE) para a linguagem Java;

- **Java** (Java, 2009) é uma tecnologia que engloba a linguagem de programação e uma plataforma. No que diz respeito a linguagem de programação, é de alto nível e pode ser caracterizada como uma linguagem orientada a objetos, simples, portável, robusta, de alta performance, dinâmica e segura. A plataforma Java por sua vez, consiste do ambiente de software ou hardware no qual os programas são executados e possui dois componentes: A máquina virtual Java e a API Java. A máquina virtual Java é a base para a plataforma Java e pode ser instalada em diversas plataformas baseadas em hardware ou software caracterizando sua portabilidade. Logo a API Java é uma ampla coleção de componentes de software que provêem funcionalidades aos adeptos da linguagem;

- **AspectJ™** (Kiczales et al.,2001) é uma extensão orientada a aspectos para a linguagem Java. Com apenas alguns poucos construtores, AspectJ provê suporte para uma implementação modular à uma série de interesses transversais.

- **PostgreSQL**(PostgreSQL, 2009) é um sistema de banco de dados relacional de código aberto. Possui mais de 15 anos de desenvolvimento ativo e uma arquitetura confiável. Pode ser executado em sistemas operacionais como o Linux, Unix e Windows. Entre suas caracteristicas encontram-se total suporte a chaves extrangeiras , joins, views, triggers e stored procedures. Este sistema também conta com interfaces nativas de programação para C/C++, Java, .Net, Perl, Python, Tcl, Ruby, ODBC entre outras além de uma vasta documentação.

- **Subversion (SVN)** (Tigris, 2009) é um sistema de controle de versão desenvolvido pela CollabNet Inc desde 1999 sob a licença Apache Licence. É usado para manter as versões de arquivos como por exemplo códigos fonte. Ele se tornou o sucessor de outro sistema de controle de versão, o Concurrent Version System (CVS). O SVN é bem conhecido pela comunidade *open source* e é utilizado em muitos projetos incluindo o Apache Software Foundation, Free Pascal, FreeBSD entre outros. Pode ser utilizado em sistemas operacionais como o Linux, Unix e Windows;

- **Maven**(Apache, 2009) é uma ferramenta para gerenciamento e automação de projetos. É baseado no conceito de um *project object model* (POM).

- **Hibernate**(Hibernate, 2009) é um *framework* de código aberto que realiza o mapeamento objeto-relacional para persistência de dados utilizando arquivos de configuração no formato XML.

- **Spring**(Spring, 2009) é um *framework* de código aberto utilizado para realizar inversão de controle e injeção de dependência. O Spring se encarrega de fazer a instanciação das classes em uma aplicação Java, definir as dependências entre elas e consequentemente permitir o baixo acoplamento.

Para a geração das aplicações, Donegan (2008) usou um gerador de aplicações configurável, o Captor (Shimabukuro, 2006).

4 Desenvolvimento de Variabilidade da LPS-BET usando aspectos

4.1 Considerações Iniciais

Neste capítulo apresenta-se a implementação da variabilidade número de cartões utilizando aspectos. Também apresenta-se a forma como foram feitas as configurações do ambiente de desenvolvimento e dos recursos que permitiram a utilização do banco de dados de forma remota.

4.2 Configurações do Ambiente de Desenvolvimento

4.2.1 Configuração do SVN

As implementações foram realizadas utilizando o sistema operacional linux Ubuntu 9.04. O código fonte da LPS-BET foi disponibilizado para estudo para um grupo de acadêmicos que fariam alterações em partes da LPS para atender aos propósitos específicos de cada um.

Para tanto, foi feito uso da IDE Eclipse com um plugin do gerenciador de controle de versão Subversion para que fosse possível realizar a sincronização das alterações realizadas no sistema. Isso permite a atuação de uma equipe sobre o mesmo sistema submetendo atualizações quando necessárias. Um roteiro completo para a instalação e configuração deste plugin, pode ser encontrado em http://subeclipse.tigris.org[1].

4.2.2 Configuração do Maven

Em seguida deve-se instalar o Maven para que todas as dependências do projeto sejam satisfeitas sempre que necessário. Para tanto, abra o console e digite a seguinte linha de comando:

[1] http://subclipse.tigris.org/servlets/ProjectProcess?pageID=p4wYuA . Acessado em 15/08/2009

```
sudo aptitude install maven2
```

Na sequência deve ser obtida uma cópia de trabalho dos fontes da LPS-BET que estão armazenados em

```
http://code.google.com/p/bet
```

4.2.3 Obtendo os fontes da LPS-BET

Com o plugin do Subversion instalado na IDE Eclipse, acessar o menu Window, submenu Customize Perspective e selecionar a opção SVN. Este conjunto de ações permitirá visualizar o item *Checkout project from SVN* ao acessar o menu File e submenu New. Ao selecionar esta opção, selecione a opção "Create a new repository location" e avance para a próxima tela. Preencha o campo Url com a url http://bet.googlecode.com/svn/trunk/ e finalize o processo clicando em finish. Com o repositório criado, clique com o botão direito sobre ele e selecione a opção checkout.

Ao final do checkout, deve-se abrir o console e vá até a pasta bet da cópia de trabalho e digite os seguintes comandos:

```
mvn eclipse:clean
```

```
mvn eclipse:eclipse
```

4.2.4 Configurando a IDE Eclipse para fazer a depuração do código

Para permitir que o código seja inspecionado quando de sua execução, tanto no ambiente Desktop quanto no ambiente Web, algumas configurações devem ser realizadas como segue:

- Execute a IDE Eclipse;
- Selecione o menu Run e o submenu Debug Configurations;
- No campo Name, informe um nome qualquer para esta configuração;
- Selecione a aba Main;
    - * No campo MainClass informe "org.codehaus.classworlds.Launcher";
    - * No campo Project informe o nome do projeto que se deseja inspecionar;

- Selecione a aba Arguments;

  * No campo Program Arguments informe “jetty:run”;

  * No campo VM Arguments infome:“-Xmx512M -Dclassworlds.conf= /usr/share/maven2/bin/m2.conf -Dmaven.home=/usr/share/maven2/”;

- Selecione a aba Classpath;

  * Remova a aplicação de User Entries e informe usr/share/maven2/boot/classworlds.jar

- Selecione a aba Source;

  * Adicione o projeto que deseja inspecionar;

A partir deste momento a IDE Eclipse está pronta para depurar o código do projeto. Para isso basta criar breakpoints pelo código e executar a aplicação acessando o menu Run, submenu Debug History e selecione a configuração que acabou de fazer.

4.2.5 Configurando o acesso ao banco de dados

As alterações referentes ao esquema de banco de dados foram feitas por meio de acesso remoto utilizando SSH (Secure Shell)(SSH, 2009). Dessa forma, foi possível utilizar a ferramenta PgAdmin para realizar as alterações quando necessárias.

Devido às limitações quanto à documentação do projeto, houve a necessidade de se manter em um primeiro instante, a base de dados centralizada. Dessa maneira qualquer membro da equipe que fizesse alguma alteração na estrutura de tabelas devia comunicar imediatamente o que foi feito para os outros membros. Esta abordagem foi utilizada para evitar que houvessem bases de dados com estruturas diferentes rodando aplicações em comum.

O banco de dados foi instalado em um servidor que se encontrava em uma rede interna do Departamento de Informática da Universidade Estadual de Maringá. Este servidor foi acessado pela aplicação utilizando uma técnica chamada *Tunneling*(SSH, 2009) como mostra a Figura 21. Esta técnica foi utilizada devido a segurança no envio de informações e também porque foi o único meio encontrado para podermos acessar um serviço remoto no tempo disponível.

Figura 21: Representação da Técnica Tunel SSH (SSH, 2009)

4.3 Considerações pré-implementação

Antes de iniciar a implementação da variabilidade "Número de Cartões" utilizando aspectos, verificou-se que os métodos contidos na interface *INumCartoesMgt* não identificavam o passageiro para a realização de consultas. De qualquer modo, após estudos realizados no código, constatou-se a necessidade de se fazer pequenas modificações devido a algumas exceções que estavam ocorrendo. Além disso, a tabela "var_num_cartoes" ganhou mais uma coluna (idpassageiro) para poder identificar o passageiro ao qual o número de cartões pertence. A seguir, uma breve descrição do código extra que foi adicionado:

- Interface *InumCartoesComplemento*: possui dois novos métodos para poder criar e buscar o número de cartões de cada passageiro;
- *FormCartaoNumCartoes.vm*: este formulário é o meio pelo qual será possível criar o número de cartões (quantidade) que cada passageiro poderá possuir, ficando a critério do funcionário, averiguar qual tipo de cartão o passageiro pode adquirir. Não se faz distinção entre as operações de criação e de alteração da quantidade de cartões, uma vez que o sistema faz o tratamento internamente.

Além do intuito de minimizar o impacto no código já existente, esta nova interface foi criada para substituir os métodos contidos na interface *InumCartoesMgt*, pois estes não faziam referência ao passageiro no momento de criar e de buscar o limite de cartão.

4.4 Implementação da variabilidade Número de Cartões utilizando aspectos

Para este trabalho, os aspectos foram utilizados para implementar código referente a requisitos funcionais da LPS-BET aplicação referência de Campo Grande, mais precisamente, a variabilidade número de cartões. Esta variabilidade trata das operações de controle relacionadas ao caso de uso adquirir cartão. A restrição número de cartões é responsável por criar ou alterar

a quantidade de cartões que um usuário pode ter e está representada no sistema pela classe *GerenciaCartaoNumCartoes* como mostra a Figura 22.

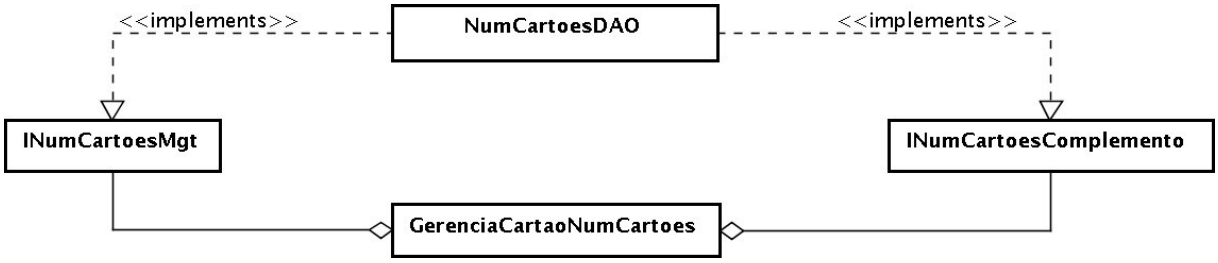

Figura 22: Diagrama de classe representando a classe *GerenciaCartaoNumCartoes*

A Figura 23 representa parcialmente a arquitetura da LPS original, onde a classe *NumCartoesDAO* implementa as operações contidas em *INumCartoesComplemento*.

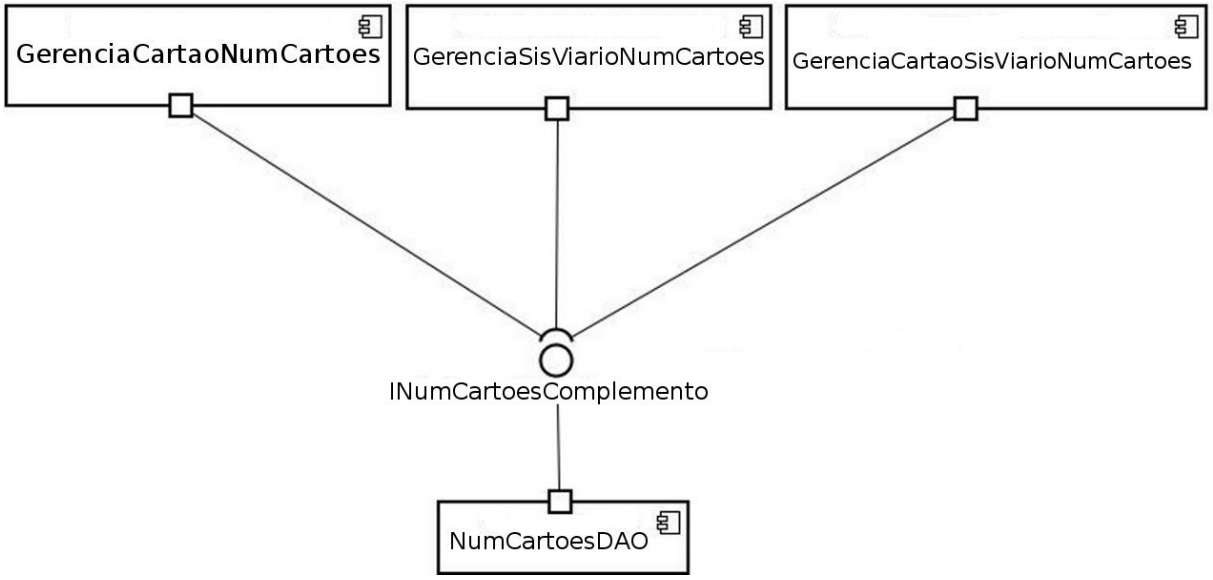

Figura 23: Arquitetura parcial da LPS-BET - sem aspecto

As Figuras 22 e 23 foram extraídas do trabalho de Donegan(2008) utilizando-se engenharia reversa do código da LPS-BET, uma vez que estas não foram representadas por meio de figuras em seu trabalho. O aspecto criado, *NumCartoesAspect*, intercepta as chamadas aos métodos contidos na interface *INumCartoesComplemento* utilizando o adendo *around* como mostra a Figura 24. A Figura 25 mostra a classe *NumCartoesDAO* que implementa a interface *INumCartoesComplemento*. Ao observar o código que está compreendido entre as linhas 42 a 52 e 55 a 68 da Figura 25 constata-se que o código é o mesmo que está contido entre as linhas 20 a 30 e 40 a 53 da Figura 26 respectivamente.

A implementação do método é toda feita dentro do aspecto, mesmo sem ter sido utilizado a palavra reservada *implements* (o que não é possível de ser feito por um aspecto). A

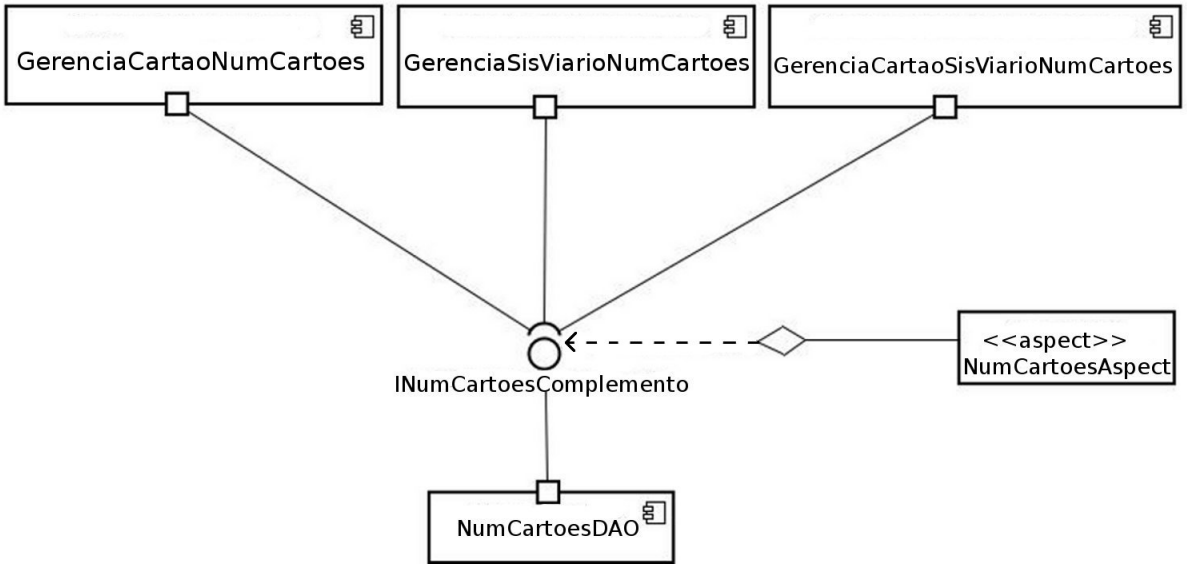

Figura 24: Arquitetura parcial da LPS-BET - com aspecto

```
10 public class NumCartoesDAO extends HibernateDaoSupport implements INumCartoesMgt, INumCartoesComplemento {
11
12⊕     public void salvarNumeroCartoes(SistViarioUrbanoNumCartoes numCartoes) {
15
16⊕     public void alterarLimiteCartoes(int limiteCartoes) {
21
22⊕     public int buscarLimiteCartoes() {
27
28⊕     public SistViarioUrbanoNumCartoes buscarSistViarioUrbanoNumCartoes() {
32
33⊕     public void criarLimiteCartoes(int limiteCartoes) {
40
41⊖     public int buscarLimiteCartoes(int idPassageiro) {
42          DetachedCriteria passageiro = DetachedCriteria.forClass(SistViarioUrbanoNumCartoes.class);
43          passageiro.add(Restrictions.eq("idPassageiro", idPassageiro));
44          List tmp = getHibernateTemplate().findByCriteria(passageiro);
45          SistViarioUrbanoNumCartoes object = null;
46          int retorno = 0;
47          if (tmp.size()>0){
48               object = (SistViarioUrbanoNumCartoes) tmp.get(0);
49               retorno = object.getLimiteCartoes();
50          }
51          return retorno;
52      }
53
54⊖     public void criarLimiteCartoes(int limite, int idPassageiro) {
55          SistViarioUrbanoNumCartoes obj = new SistViarioUrbanoNumCartoes();
56          obj.setIdPassageiro(idPassageiro);
57          obj.setLimiteCartoes(limite);
58          DetachedCriteria criteria = DetachedCriteria.forClass(SistViarioUrbanoNumCartoes.class);
59          criteria.add(Restrictions.eq("idPassageiro", idPassageiro));
60          List tmp = getHibernateTemplate().findByCriteria(criteria);
61          if (tmp.size() > 0){
62              SistViarioUrbanoNumCartoes s = (SistViarioUrbanoNumCartoes) tmp.get(0);
63              obj.setNumCartoesID(s.getNumCartoesID());
64              getHibernateTemplate().saveOrUpdate(obj);
65          }else{
66              getHibernateTemplate().saveOrUpdate(obj);
67          }
68      }
69 }
```

Figura 25: Classe *NumCartoesDAO* que implementa as interfaces alvo da interceptação

Figura 26 mostra a implementação do aspecto *NumCartoesAspect*. As linhas 15 e 34 da Figura 26 mostram a construção dos *pointcuts* , que são responsáveis pela interceptação. Pode-se ob-

servar que a construção foi feita utilizando uma referência completa ao método que deve ser interceptado, ou seja, especificando visibilidade (*public*), tipo (int), objeto (classe ou interface) e assinatura do método. A partir deste momento, a responsabilidade pelas operações contidas na interface *INumCartoesComplemento* fica a cargo do aspecto representado pela Figura 26.

```
1   package lps.bet.variabilidades;
2
3   import java.util.List;
4   import lps.bet.variabilidades.numCartoesMgr.INumCartoesComplemento;
5   import lps.bet.variabilidades.numCartoesMgr.NumCartoesDAO;
6   import lps.bet.variabilidades.numCartoesMgr.INumCartoesMgt;
7   import lps.bet.variabilidades.tiposDados.SistViarioUrbanoNumCartoes;
8   import org.hibernate.criterion.DetachedCriteria;
9
10  public aspect NumCartoesAspect {
11      HibernateDaoSupport hibernate;
12      NumCartoesDAO numCartoesDao;
13
14      pointcut buscarLimiteCartoes(HibernateDaoSupport h, int idPassageiro):
15          call(public int INumCartoesComplemento.buscarLimiteCartoes(int))
16          && args(idPassageiro) && target(h) && !within(NumCartoesAspect);
17
18          int around(HibernateDaoSupport h, int idPassageiro):
19              buscarLimiteCartoes(h, idPassageiro){
20                  DetachedCriteria passageiro = DetachedCriteria.forClass(SistViarioUrbanoNumCartoes.class);
21                  passageiro.add(Restrictions.eq("idPassageiro", idPassageiro));
22                  List tmp = h.getHibernateTemplate().findByCriteria(passageiro);
23                  SistViarioUrbanoNumCartoes object = null;
24                  int retorno = 0;
25                  if (tmp.size()>0){
26                      object = (SistViarioUrbanoNumCartoes) tmp.get(0);
27                      retorno = object.getLimiteCartoes();
28                  }
29              return retorno;
30              }
31
32
33      pointcut criarLimiteCartoes(HibernateDaoSupport h, int limite, int idPassageiro):
34          call(public void INumCartoesComplemento.criarLimiteCartoes(int, int))
35          && args(limite, idPassageiro) && target(h) && !within(NumCartoesAspect);
36
37
38      void around(HibernateDaoSupport h, int limite, int idPassageiro):
39          criarLimiteCartoes(h, limite,idPassageiro){
40              SistViarioUrbanoNumCartoes obj = new SistViarioUrbanoNumCartoes();
41              obj.setIdPassageiro(idPassageiro);
42              obj.setLimiteCartoes(limite);
43              DetachedCriteria criteria = DetachedCriteria.forClass(SistViarioUrbanoNumCartoes.class);
44              criteria.add(Restrictions.eq("idPassageiro", idPassageiro));
45              List tmp = h.getHibernateTemplate().findByCriteria(criteria);
46              if (tmp.size() > 0){
47                  SistViarioUrbanoNumCartoes s = (SistViarioUrbanoNumCartoes) tmp.get(0);
48                  obj.setNumCartoesID(s.getNumCartoesID());
49                  h.getHibernateTemplate().saveOrUpdate(obj);
50              }else{
51                  h.getHibernateTemplate().saveOrUpdate(obj);
52              }
53          }
54  }
```

Figura 26: Aspecto *NumCartoesAspect*

4.4.1 Limitações e dificuldades encontradas

No início deste projeto, foi proposto a implementação das variabilidades Número de Cartões e Combinação de Cartões com o uso de aspectos. No decorrer das pesquisas foi constatado que a implementação de uma das variabilidades havia sido feita parcialmente (Número de Cartões) utilizando componentes caixa-preta. Aliado a este contratempo, estava a falta da

definição clara de tal variabilidade. Desse modo, foi necessário completar a implementação utilizando as interfaces requeridas e fornecidas que contemplam tal variabilidade.

Quanto a variabilidade Combinação de Cartão, segundo Donegan(2008, p.71),

> A fase de construção desse ciclo tomou como base essa arquitetura de componentes. Das três variabilidades que necessitavam ser desenvolvidas, duas foram implementadas (Autenticação de Passageiro e Acesso Adicional) e uma será implementada futuramente pelo grupo de pesquisa (Combinação de Cartões). Portanto, faltou a implementação dos componentes CombinacaoMgr, GerênciaIntegracaoCombinacaoLimPassagensCtrl e AquisiçãoCartãoCombinação.

portanto, foi desconsiderada a implementação com aspectos para esta variabilidade.

Além disso, devido ao grande número de tecnologias utilizadas para realizar o trabalho, encontrou-se certa dificuldade na utilização do Hibernate. Além disso, a pouca documentação no código dificultou o entendimento do funcionamento do sistema.

4.5 Métricas da Variabilidade Número de Cartões

A Tabela 4 contém alguns dados quantitativos referentes a parte da LPS que foi afetada. Para efeito de comparação, pode ser observado que o número de adendos por aspecto é sempre igual ao número de operações interceptadas de cada interface. Isso ocorre porquê as operações referem-se a requisitos funcionais e tem funções distintas dentro do contexto da aplicação. Os números poderiam ser outros se o aspecto fosse utilizado para interceptar requisitos não-funcionais como por exemplo, registrar em *log* qual usuário realizou determinada operação. Neste caso, poderia ser usado apenas 1 adendo para N operações independente de onde o método fosse chamado.

As linhas de código do controlador substituído (30) é menor em relação a quantidade de linhas de código do aspecto (54) porquê no controlador substituído foi contado apenas as linhas referentes aos métodos da interface interceptada. Quanto aos aspectos, foram consideradas as linhas do aspecto como um todo, uma vez que ele representa apenas os métodos da interface interceptada acrescentados a este, seus construtores, *advices* e *pointcuts*.

Tabela 4: Algumas métricas da implementação da variabilidade número de cartões

Descrição	**Qtde**
Interfaces envolvidas com a variabilidade	1
Componentes	1
Linha de código do método `buscarLimiteCartoes(int)`	14
Linha de código do método `criarLimiteCartoes(int,int)`	16
Interfaces interceptadas por aspectos	1
Adendos por aspecto	2
Operações interceptadas na interface	2
Classes que referenciam o controlador substituído por aspecto	3
Componentes que referenciam o controlador substituído por aspecto	1
Linhas de código do controlador substituído	30
Linha de código do aspecto	54

4.6 Considerações Finais

No contexto desse trabalho, no caso de alguma classe implementar as funcionalidades de alguma interface que estão marcadas para serem interceptadas por algum aspecto sob a forma do adendo *around*, o código implementado na classe não será executado pois os adendos utilizados desviam o fluxo de execução, tomando para si esta responsabilidade.

Por outro lado, quando os métodos forem interceptados por um aspecto na forma dos adendos *after* ou *before* os métodos manterão sua função no contexto da aplicação. Nesta situação, funcionalidades extras são adicionadas caracterizando seu uso junto aos requisitos não-funcionais.

Se ao invés de implementar os métodos de uma interface, uma classe utilizar a interface como uma propriedade e houver a interceptação em sua chamada, o aspecto deve conter o código referente a funcionalidade. Nestas circunstâncias, apenas o adendo *around* pode ser utilizado, pois o único código existente está dentro do aspecto.

O uso de aspectos, tanto para implementar requisitos não-funcionais quanto para requisitos funcionais, podem causar alguma confusão se o código que vai ser alvo da interceptação não for devidamente documentado. Com ferramentas modernas de apoio ao desenvolvimento, as informações que elas emitem no ato da codificação nos ajudam a depurar e a documentar o código. No caso dos aspectos sendo utilizados na IDE Eclipse, estas informações são visuais e ficam à esquerda do código que será interceptado. Isto pode ser visto na Figura 27.

A não execução de um código ou o comportamento não esperado antes ou depois de seu processamento podem fazer com que haja desperdício de tempo para descobrir o que está acontecendo. Estes são alguns dos prejuízos decorrentes da má utilização de aspectos.

Portanto, não foi observado vantagens quanto ao uso de aspectos para a implementação

```
156
157     protected ModelAndView mostrarForm(String cartaoID, String passageiroID, String form) {
158
159         ModelAndView mav = new ModelAndView(form);
160         mav.addObject("cartaoID", cartaoID);
161         mav.addObject("idPassageiro", passageiroID);
162 advised by NumCartoesAspect.around(HibernateDaoSupport,int): buscarLimiteCartoes.. (runtime test) (passagei
163
164
165         // SimpleDateFormat sdf = new SimpleDateFormat();
166         mav.addObject("sdf", sdf);
167
168         Cartao cartao = null;
169         Calendar dtAquisicao;
170         Calendar dtValidade;
171         List passageiros = interfacePassageiroMgt.buscarPassageiros();
```

Figura 27: Trecho de código interceptado por aspectos

de variabilidades desta LPS. O uso da POA com a linguagem AspectJ para este fim, tende a descaracterizar a estrutura da linguagem Java, pois faz com que o conceito de herança do Java fique a mercê do desenvolvedor.

Isto se deve ao fato de que a construção dos *pointcuts* permitem que as interceptações ocorram indiscriminadamente sem observar os princípios de herança do Java. Um exemplo disso pode ser visto no uso de um padrão chamado *template* como ilustra a Figura 28.

```
1   public abstract class ClasseExemplo{
2
3       final void iniciarProcesso(){
4           metodo1();
5           emitirDocumento();
6           metodo2();
7           ...
8           ...
9           ...
10      }
11
12      public abstract void emitirDocumento();
13
14      public void metodo1(){
15          ...
16          ...
17          ...
18      }
19
20      public void metodo2(){
21          ...
22          ...
23          ...
24      }
25  }
```

Figura 28: Classe representando o padrão *template*

Este padrão tem por característica permitir que uma sequência de algoritmos sejam executados de forma a atender necessidades específicas utilizando pelo menos um método abstrato sendo chamado a partir de um método *final*. Todas as classes concretas que extenderem a classe onde este método abstrato está inserido deverá realizar sua implementação conforme

sua necessidade como mostra a Figura 29. Dessa maneira, o desenvolvedor que utilizar um aspecto para interceptar a execução deste método deverá ser bem específico em sua construção ou execuções inadvertidas poderão ocorrer.

```
1  public class IntegracaoCtr extends ClasseExemplo{
2
3
4      public void emitirDocumento(){
5          ...
6          ...
7          ...
8      }
9  }
```

Figura 29: Classe concreta que implementa o método herdado

Com base no exposto até o momento, pode-se realizar uma comparação entre a implementação utilizando componentes e a implementação utilizando aspectos. Esta comparação levou em conta a implementação dos métodos contidos na interface *INumCartoesComplemento* (interface adaptada ao projeto semelhante a interface *INumCartoesMgt*) que mostrou que o código produzido tanto para componentes quanto para aspectos foram iguais. Além disso, a implementação com aspectos fez com que o código implementado sem aspecto não fosse mais alcançado pelo fluxo de execução.

5 Conclusão

O objetivo principal deste trabalho era implementar as variabilidades “Número de Cartões” e “Combinação de Cartões” utilizando a linguagem orientada a aspectos AspectJ para se obter evidências da eficácia do uso de aspectos com requisitos funcionais da LPS-BET. Também foi previsto a geração de uma aplicação utilizando os novos requisitos implementados com aspectos. Portanto, a principal contribuição deste trabalho é mostrar evidências sobre o uso de aspectos na implementação de variabilidades da LPS.

Quanto às variabilidades alvo deste trabalho atingiu-se parcialmente o objetivo visto que a variabilidade “Combinação de Cartões” não foi implementada como evidenciado por Donegan (2008).

O uso de aspectos para implementar código referente a requisitos funcionais pode causar confusão. Os desenvolvedores devem estar cientes da necessidade da documentação do código produzido a fim de evitar desperdícios de recursos de qualquer natureza. Além disso, no caso da variabilidade implementada, o uso de aspectos se mostrou equivalente à implementação com componentes e portanto, não observou-se vantagens na utilização de aspectos para implementar variabilidades que representam requisitos funcionais.

Por outro lado, o emprego de aspectos para implementar código referente a requisitos não-funcionais contribui para uma maior modularização e facilidade de manutenção.

Quanto à etapa de geração de uma aplicação com as variabilidades implementadas com aspectos será contemplada em um trabalho futuro a ser realizado por uma equipe da USP com o auxílio de uma versão orientada a aspectos do gerador de aplicação configurável Captor.

Ainda como trabalhos futuros pode-se citar a realização de novos estudos de caso com o intuito de se implementar variabilidades de outra LPS utilizando aspectos para reforçar as evidências encontradas. Além de fazer a evolução da LPS-BET em duas versões: uma para adicionar novas funcionalidades usando aspectos e outra versão usando componentes a fim de comparar os resultados obtidos.

Referências Bibliográficas

APACHE The Apache Software Foundation. Disponível em http://www.apache.org/. Acessado em Julho, 2009.

CLEMENTS, P.; NORTHROP, L. **Software Product Lines: Practices and Patterns**. Boston: Addison-Wesley, 2001.

DONEGAN, P. M. **Geração de Famílias de Produtos de Software com Arquitetura Baseada em Componentes**. São Paulo, junho/2008. p.160 Dissertação(Mestrado em Ciência da Computação) - Instituto de Ciências Matemáticas e de Computação - ICMC, Universidade de São Paulo(USP).

ECLIPSE. The Eclipse Foundation. Disponível em http://www.eclipse.org. Acessado em maio, 2009.

HIBERNATE, Hibernate Red Hat Middleware, LLC. Disponível em http://www.hibernate.org. Acessado em agosto, 2009.

JAVA Sun Microsystems, Inc. Disponível em http://Java.sun.com/. Acessado em outubro, 2009.

KICZALES, G.;HILSDALE, E.; HUGUNIN, J.; KERSTEN, M.; PALM, J.; GRISWOLD, W. **An Overview of AspectJ**. In: *Proceedings of the 15th European Conference on Object-Oriented Programming, Lecture Notes in Computer Science*, Springer-Verlag, London, UK, 2001, p. 327-353 **An overview of AspectJ. Disponível em: http://www.cs.ubc.ca/ gregor/** Acessado em maio, 2009.

PARNAS, D. **Designing Software for Ease of Extension and Contraction**. IEEE Transactions on Software Engineering 5, 2, p. 128-138, 1979.

POSTGRESQL PostgreSQL Global Development Group. Disponível em http://www.postgresql.org/. Acessado em outubro, 2009.

SEI - SOFTWARE ENGINEERING INSTITUTE. http://www.sei.cmu.edu. Acessado em maio, 2009.

SHIMABUKURO, E. K. J. **Um Gerador de Aplicações Configurável**. São Paulo, abril/2006. p.155 Dissertação (Mestrado em Ciência da Computação) - Instituto de Ciências Matemáticas e de Computação - ICMC, Universidade de São Paulo(USP).

SPRING, Spring Framework. Disponível em http://www.springsource.org. Acessado em setembro, 2009.

SSH, SSH Communications Security Corp. Disponível em http://www.ssh.com. Acessado em agosto, 2009.

TIGRIS SUBVERSION. Disponível em http://subversion.tigris.org. Acessado em outubro, 2009.