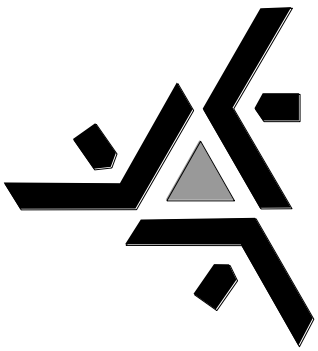

Universidade Estadual de Maringá
Centro de Tecnologia
Departamento de Informática

Uma investigação quanto ao uso de aspectos para implementação de variabilidades de Linha de Produto de Software

Guilherme Antonialli Meilsmith

TG-15-08

Maringá - Paraná

Brasil

Universidade Estadual de Maringá
Centro de Tecnologia
Departamento de Informática

Uma investigação quanto ao uso de aspectos para implementação de variabilidades de Linha de Produto de Software

Guilherme Antonialli Meilsmith

TG-15-08

Trabalho de Graduação apresentado ao Curso de Ciência da Computação, do Centro de Tecnologia, da Universidade Estadual de Maringá.
Orientador: *Profa. MSc.Thelma Elita Colanzi Lopes*

Maringá - Paraná

2008

Guilherme Antonialli Meilsmith

Uma investigação quanto ao uso de aspectos para implementação de variabilidades de Linha de Produto de Software

Este exemplar corresponde à redação final da monografia aprovada como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação da Universidade Estadual de Maringá, pela comissão formada pelos professores:

_____

Orientador: Profa. MSc.Thelma Elita Colanzi Lopes
Departamento de Informática, CTC, DIN

_____

Profa. Dra. Itana Maria de Souza Gimenes
Departamento de Informática, CTC, DIN

_____

Profa. Dra. Sandra Ferrari
Departamento de Informática, CTC, DIN

Maringá, novembro de 2008

Universidade Estadual de Maringá
Departamento de Informática
Av. Colombo 5790, Maringá-PR
CEP 87020-900
Tel: (44) 3261-4324 Fax: (44) 3261-4074

Agradecimentos

Ao longo destes anos de estudos tenho muito a agradecer e a muitas pessoas que me ajudaram direta e indiretamente.

Primeiramente agradeço a Deus, por me dar oportunidade, capacidade e vontade para realizar este trabalho.

Agradeço em especial à minha família maravilhosa que sempre esteve ao meu lado, me apoiando para que eu sempre alcance meus objetivos. Obrigado pela motivação e por acreditarem no meu potencial.

Agradeço à professora Thelma Elita Colanzi Lopes que me ensinou muito nesse período que passamos juntos. Obrigado pelas diversas sugestões desde os assuntos a ser pesquisados, artigos e livros a serem lidos, até as vírgulas e palavras a serem corrigidas.

Agradeço muito a Paula Marques Donegan por me auxiliar no desenvolvimento deste trabalho. Obrigado pela paciência e atenção sempre que precisei.

Agradeço à professora Sarajane Marques Peres pela dedicação, disposição e pela oportunidade que você proporcionou de eu apresentar nosso artigo no Segundo Congresso Latinoamericano de Objetos de Aprendizagem - Santiago do Chile.

Agradeço a todos integrantes do PET-Informática da Universidade Estadual de Maringá. Ser "petiano" ajudou a complementar minha formação acadêmica e pessoal.

Agradeço aos professores do Departamento de Informática da Universidade Estadual de Maringá que contribuíram para minha formação.

Agradeço a todos os meus amigos que sempre estiveram ao meu lado nos bons e a maus momentos. E aos que mesmo à distância não deixaram de me apoiar.

Finalmente, agradeço a todas as pessoas que contribuíram de alguma forma para a realização deste trabalho.

Resumo

Este trabalho tem como finalidade investigar o uso de aspectos para a implementação de variabilidades de Linha de Produto de Software. Tomou-se como base uma LPS existente que teve o objetivo de investigar como pode ser elaborada a estrutura de uma LPS com arquitetura baseada em componentes, usando aspectos para representar a composição de variabilidades de uma LPS e um gerador de aplicação para fazer a composição dos aspectos e dos componentes. O objetivo geral deste trabalho é aumentar as evidências sobre em quais situações é melhor usar aspectos na evolução de LPS. Para obter essas evidências, foi desenvolvido um estudo de caso baseado no projeto da LPS-BET para analisar uma solução de implementação de variabilidades usando componentes junto com aspectos para as variabilidades relacionadas ao uso de cartões.

Abstract

This project aims to investigate the use of aspects for the implementation of variability of Software Product Line. An existing LPS was takes as a basis, which had as objective to investigate how to elaborate the structure of a LPS with components based architecture, using aspects to represent the variability composition of a LPS and an application generator to perform the composition of the aspects and of the components. The general objective of this project is to increase the evidences about the situations in which it is better to use aspects in the evolution of LPS. To obtain these evidences, a study of case has been developed based in the project of LPS-BET, in order to review a solution to variability implementation with the use of components along with aspects for the variability related to the use of cards.

Lista de Figuras

Lista de Tabelas

Lista de Siglas

ADL	*Architecture Description Language*
BET	Bilhetes Eletrônicos de Transporte municipal
Captor	*Configurable Application Generator*
DSBC	Desenvolvimento de Software Baseado em Componentes
ESPLEP	*Evolutionary Software Product Line Engineering Process*
IDE	*Integrated Development Environment*
LPS	Linha de Produtos de Software
MVC	*Model View Controller*
OA	Orientação a Aspectos
OO	Orientação a Objetos
POA	Programação Orientada a Aspectos
SGBD	Sistema Gerenciador de Bancos de Dados
UML	*Unified Modeling Language*
XML	*eXtensible Markup Language*
XSLT	*eXtensible Stylesheet Language Transformations*

Sumário

Capítulo 1
Introdução

Atualmente existem diversas técnicas para o reuso de software, como componentes de software, *frameworks* orientados a objetos, geradores de aplicação, padrões de projeto e linguagens orientadas a aspectos, dentre outras. Para aumentar os benefícios do reuso, essas técnicas podem ser combinadas e usadas em uma linha de produto de software (LPS). Clements e Northrop (2002) definem LPS como um conjunto de sistemas de software que compartilham um conjunto de características comuns e gerenciadas, satisfazendo as necessidades específicas de um segmento de mercado, ou objetivo particular, e que são desenvolvidas de maneira pré-definida a partir de um conjunto comum de recursos-base.

Estudos têm indicado problemas a serem investigados, como aspectos, arquiteturas, métodos ágeis, componentes de software e LPS (FINKELSTEIN e KRAMER, 2000; OSTERWEIL, 2007; TAYLOR E VAN DER HOEK, 2007) *apud* (DONEGAN, 2008).

Donegan (2008) afirma que diversos artigos enfatizam a dificuldade de elicitar, representar e implementar variabilidades no contexto de uma LPS (BACHMANN *et al.*, 2003; BECKER e KAISERSLAUTERN, 2003; BOSCH *et al.*, 2002; JUNIOR *et al.*, 2005; ANASTASOPOULOS e GACEK, 2001). Características variantes são difíceis de implementar, pois elas podem se espalhar por diversas unidades de decomposição como classes e componentes, dependendo da tecnologia utilizada e são geralmente implementadas usando padrões de projeto, classes parametrizadas e outras técnicas de baixo nível. Diversos pesquisadores têm investigado essa questão e têm proposto outras soluções baseadas em linguagens de programação como programação orientada a aspectos e programação orientada a características (MEZINI e OSTERMANN, 2004; APEL e BATORY, 2006; HEO e CHOI, 2006; LEE *et al.*, 2006; ANASTASOPOULOS E MUTHIG, 2004) *apud* Donegan (2008). A combinação de aspectos, componentes e *frameworks* é proposta por Griss (2000) *apud* Donegan (2008) para implementar LPS, mas sem apresentar uma proposta particular em relação a como fazer a combinação.

O objetivo geral deste trabalho de graduação é analisar o uso de aspectos para a implementação de variabilidades de LPS a partir de um projeto de mestrado que teve como

finalidade investigar como pode ser elaborada a estrutura de uma LPS com arquitetura baseada em componentes, usando aspectos para representar a composição de variabilidades de uma LPS e um gerador de aplicação para fazer a composição dos aspectos (interesses transversais) e dos componentes (DONEGAN e MASIERO, 2007). A LPS em desenvolvimento controla Bilhetes Eletrônicos para Transporte (BET) municipal e foi denominada LPS-BET. Ela facilita o transporte urbano com o uso de um cartão eletrônico para pagar passagens e oferece várias outras funcionalidades para passageiros e companhias de ônibus. O seu domínio foi projetado e desenvolvido visando a geração de três produtos (ou aplicações), a partir de uma análise feita em três sistemas reais de municípios brasileiros: os sistemas BET de São Carlos (SP), Fortaleza (CE) e Campo Grande (MS).

Assim sendo, o objetivo específico deste trabalho é analisar o uso de aspectos para representar variabilidades relacionadas ao uso de cartões na arquitetura da LPS-BET. Esta análise envolve estudos para avaliar os prós e contras do uso de aspectos em uma arquitetura baseada em componentes caixa-preta.

No Capítulo 2 são apresentadas uma revisão bibliográfica sobre orientação a aspectos, uma explanação a respeito de Linha de Produto de Software, descrevendo o processo de desenvolvimento da LPS-BET e, por fim, conceitos sobre o funcionamento do gerador de aplicações configurável: Captor. No Capítulo 3 abordam-se questões relacionadas ao projeto da LPS-BET e é considerado o uso de aspectos como alternativa para implementação de variabilidades da LPS-BET em uma arquitetura baseada em componentes caixa-preta.

No Capítulo 4 é analisado o uso de aspectos para a implementação de variabilidades relacionadas ao uso de cartões existentes na LPS-BET. São mostrados os passos para a preparação do Ambiente BET, bem como as dificuldades encontradas para tal atividade. É apresentado também um tutorial para a preparação do Ambiente BET desenvolvido neste trabalho.

Finalmente, no Capítulo 5 apresentam-se as considerações finais, propostas de trabalhos futuros e dificuldades encontradas no desenvolvimento do trabalho.

Capítulo 2
Revisão Bibliográfica

Este capítulo apresenta uma revisão bibliográfica sobre orientação a aspectos, uma explanação a respeito de Linha de Produto de Software, descrevendo o processo de desenvolvimento da LPS-BET e, por fim, uma breve descrição sobre o funcionamento do gerador de aplicações configurável Captor.

2.1 Orientação a Aspectos

Apesar de a Orientação a Objetos (OO) facilitar o reuso, as classes não conseguem implementar apenas um elemento específico do domínio porque há um entrelaçamento de código entre as classes a fim de realizar requisitos não funcionais. Neste sentido, algumas soluções foram propostas na tentativa de resolver estes e outros problemas, entre elas, a Orientação a Aspectos (OA). A OA é um paradigma que estende a OO introduzindo novas abstrações.

O conceito de Interesse é fundamental em OA, que são as características relevantes de uma aplicação. Um interesse pode ser dividido em uma série de aspectos que representa os requisitos da aplicação. Os aspectos podem ser agrupados no domínio da aplicação, compondo os interesses funcionais, que formam a lógica de negócio. Podem também ser agrupados em elementos que apóiam os interesses funcionais nomeados por interesses sistêmicos, e também chamados de ortogonais ou transversais.

A tentativa de implementar um interesse sistêmico com a aplicação da OO tem como resultado códigos que se espalham por todo o programa. Para isso dá-se o nome de código espalhado (*Scattering Code*).

A implementação de múltiplos interesses sistêmicos e funcionais em um mesmo módulo com a OO resulta no código chamado de entrelaçado (*Tangled Code*). O código espalhado e emaranhado gera alguns problemas resultantes da descentralização do código, como: replicação de código, dificuldade de manutenção, redução da capacidade de reutilização de código e aumento da dificuldade de compreensão.

A Programação Orientada a Aspectos (POA) foi criada no fim da década de 1990, em Palo Alto, nos laboratórios da Xerox. O objetivo era construir uma abordagem que fosse um conjunto não necessariamente homogêneo que permitisse à linguagem de programação, composta por linguagem central e várias linguagens específicas de domínio, expressar de forma ideal as características sistêmicas do comportamento do programa. A essa abordagem dá-se o nome de meta-programação, Goetten e Winck (2006).

Uma forma de caracterizar uma linguagem de meta-programação é pelo seu compilador. A princípio, os compiladores destinados à orientação a aspectos não geram um produto final (um programa compilado e executável ou interpretável), mas um novo código. Nessa primeira compilação são acrescidos elementos ao código, para apoiar às novas abstrações. O código resultante necessita ser novamente compilado para que seja, então, gerado o produto final. Outra característica da meta-programação presente na POA é a reflexão computacional, em que parte do código gerado é destinada a alterar características do próprio programa, Goetten e Winck (2006).

O principal objetivo da POA consiste em separar o código referente ao negócio do sistema dos interesses transversais, de uma forma bem definida e centralizada.

A separação e a centralização proporcionada pela POA são indicadas pela Figura 2.1. Na Figura 2.1, cada nuvem representa um interesse sistêmico implementado no sistema, como auditoria (*log*), tratamento de exceções, persistência, distribuição, dentre outros. Por estarem bem separados e em locais bem definidos, os componentes podem ser melhor reutilizados e a sua manutenção e legibilidade torna-se mais agradável.

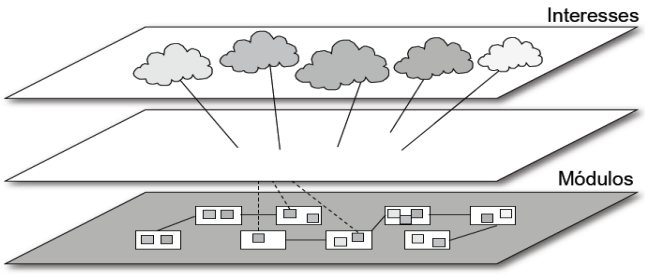

Figura 2.1: Separação de interesses com POA (GOETTEN e WINCK, 2006)

Ao disponibilizar mecanismos para decomposição não apenas dos elementos e das relações pertinentes aos problemas, mas também para separar interesses sistêmicos, a POA promete

simplificar a engenharia de software. Por exemplo, uma classe para implementação do controle de produtos, atende apenas a um interesse, não infringindo nenhum conceito da orientação a objetos. Posteriormente, surge a necessidade de implementação do controle de auditoria (*log*). Sem utilizar POA, o interesse seria implementado na própria classe Produto. Caso fosse necessário implementar o controle de exceções, esse também seria feito juntamente na classe. A conseqüência disso é um acréscimo gradativo da complexidade no código.

Na POA, por sua vez, os interesses são programados em módulos separados (classes e aspectos, por exemplo). Após a programação, ocorre a combinação entre as classes e os aspectos. Isso leva à simplificação do problema, pois o programador pode focar cada interesse de forma independente.

Em síntese, aspectos são módulos que permitem separar e encapsular interesses transversais eliminando o espalhamento e o entrelaçamento de código. O aspecto adiciona uma funcionalidade ao código-base interceptando o fluxo de execução. No código-base não é preciso haver menção explícita ao aspecto. Portanto, ao utilizar a POA, os sistemas ficam mais legíveis, fáceis de entender, implementar, integrar, reusar, personalizar, evoluir e manter (KICZALES, 1997).

2.1.1 Composição de um sistema orientado a aspectos

A composição de um programa OA consiste em: uma linguagem para programar os componentes (por exemplo, Java), uma linguagem para programar os aspectos (por exemplo, o AspectJ) e um combinador aspectual (*weaver*) para combinar as duas linguagens (ferramenta que também faz parte do AspectJ). O *weaver* é uma espécie de montador que tem como entrada um programa de componente e o(s) programa(s) de aspectos e como saída um programa em uma linguagem específica (por exemplo, Java). Essa composição é ilustrada na Figura 2.2.

A combinação aspectual é o processo que antecede a compilação, gerando um código intermediário na linguagem específica capaz de produzir a operação desejada, ou de permitir a sua realização durante a execução do programa.

As classes referentes ao código do negócio nos sistemas não sofrem qualquer alteração para apoiar a POA. Isso é feito no momento da combinação entre os componentes e os aspectos (Goetten e Winck, 2006).

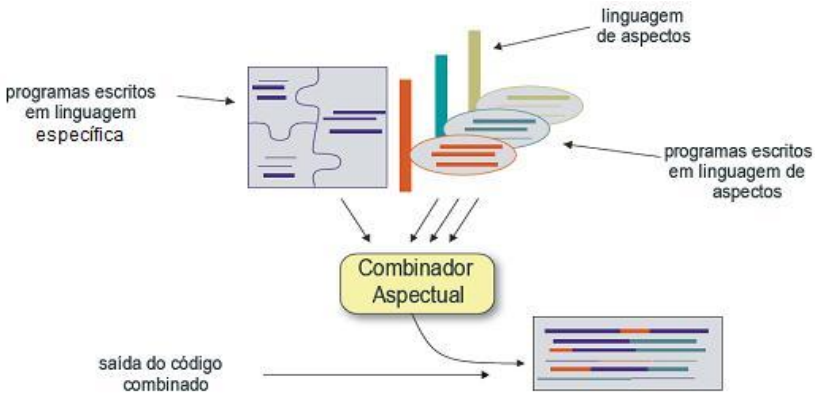

Figura 2.2: Composição de um sistema orientado a aspectos (adaptado de GOETTEN e WINCK, 2006)

2.1.2 Conceitos fundamentais da orientação a aspectos

A OA possui quatro conceitos fundamentais, que são descritos nas subseções a seguir:

Pontos de junção (*join points*)

Os pontos de junção são locais bem definidos da execução de um programa, como, por exemplo, uma chamada a um método ou a ocorrência de uma exceção, dentre muitos outros, onde o aspecto pode ser aplicado (Kiczales,1997).

A partir dos pontos de junção, são criadas as regras que darão origem aos pontos de atuação. Todos os pontos de junção possuem um contexto associado. Por exemplo, a chamada para um método possui um objeto chamador, o objeto alvo e os argumentos do método disponível como contexto.

Pontos de atuação (*pointcuts*)

Pontos de atuação são elementos do programa usados para definir um ponto de junção, como uma espécie de regra criada pelo programador para especificar eventos que serão atribuídos aos pontos de junção. Os pontos de atuação têm como objetivo criar regras genéricas para definir os eventos que serão considerados pontos de junção, sem precisar defini-los individualmente. Outra função dos pontos de atuação é apresentar dados do

contexto de execução de cada ponto de junção, que serão utilizados pela rotina disparada pela ocorrência do ponto de junção mapeado no ponto de atuação.

Após a identificação dos pontos de junção para um determinado interesse, é necessário agrupá-los em um ponto de atuação. A combinação do ponto de junção e atuação torna evidente a ocorrência de interesses sistêmicos, já que esse interesse muito provavelmente estará propagado em diversas classes no sistema, e essas classes, estarão, portanto, implementando mais de um interesse.

O ponto de atuação é definido dentro de um aspecto no AspectJ, o tipo de acesso pode ser tanto público como privado. Todos os pontos de atuação devem possuir um único nome, isto significa que não há sobrecarga de pontos de atuação no AspectJ. Os designadores fornecem uma definição no ponto de junção utilizado no ponto de atuação. A Tabela 2.1 apresenta uma lista dos principais designadores disponíveis no AspectJ.

Tabela 2.1: Listagem dos designadores em AspectJ

Designador	Características
Call (Signature)	Invocação do método / construtor identificado por assinatura
Execution (Signature)	Execução do método / construtor identificado por assinatura
Get (Signature)	Acesso a atributo identificado por assinatura
Set (Signature)	Atribuição do atributo identificado por assinatura
This (Type pattern)	Objeto em execução é instância do padrão tipo
Target (Type pattern)	Objeto de destino é instância do padrão tipo
Args (Type pattern)	Os argumentos são instância do padrão tipo
Within (Type pattern)	O código em execução está definido em padrão tipo

Adendo (*advice*)

Adendos são pedaços da implementação de um aspecto executados nos pontos de junção. Os adendos são compostos de duas partes: a primeira delas é o ponto de atuação, que define as regras de captura dos pontos de junção; a segunda é o código que será executado quando ocorrer o ponto de junção definido pela primeira parte. O adendo é um mecanismo bastante similar a um método (quando comparado com a programação OO), cuja função é declarar o código que deve ser executado a cada ponto de junção em um ponto de atuação, ou seja, um meio para alterar o comportamento dos pontos de junção.

A diferença básica entre os tipos de adendo disponíveis no AspectJ refere-se ao momento em que estes são executados assim que o ponto de junção for alcançado. A Tabela 2.2 apresenta os diferentes tipos de adendo do AspectJ.

Tabela 2.2: Tipos de adendo disponíveis do AspectJ (GOETTEN e WINCK, 2006)

Tipo de adendo	Descrição
before	Executa quando o ponto de atuação é alcançado, mas imediatamente antes da sua computação
after returning	Executa após a computação com sucesso do ponto de atuação
after throwing	Executa após a computação sem sucesso do ponto de atuação
after	Executa após a computação do ponto de atuação, em qualquer situação
around	Executa quando o ponto de junção é alcançado e tem total controle sobre a sua computação

Aspectos (*aspects*)

Os Aspectos encapsulam *pointcuts*, *advices* e declarações *inter-types* em uma unidade modular de implementação. São definidos de maneira semelhante às classes, enquanto essas encapsulam o código que encaixa dentro de classes hierárquicas, *aspects* encapsulam o código que é ortogonal, transversal ou sobreposto a essas classes hierárquicas**.**

2.1.3 AspectJ – Orientação a Aspectos com Java

O AspectJ é uma linguagem de aspectos disponibilizada como uma extensão para a linguagem de programação Java. Assim, existe uma preocupação com a compatibilidade de quatro itens extremamente importantes e essenciais:

- todo programa Java válido é também um programa AspectJ válido;
- todo programa AspectJ pode ser executado em uma máquina virtual java (JVM);
- deve ser possível estender ferramentas existentes para suportar o AspectJ de uma forma natural; isto inclui IDE's (*Integrated Development Environments*), ferramentas de documentação e ferramentas de projeto;
- ao programar com AspectJ, o programador deve se sentir como se estivesse utilizando uma extensão da linguagem Java.

Existe uma divisão do AspectJ em duas partes: a linguagem de especificação e a linguagem de implementação. A parte da linguagem de especificação define a linguagem na qual o código é escrito; com AspectJ, os interesses funcionais são implementados em Java. Para

implementação da combinação de interesses sistêmicos, são utilizadas as extensões disponibilizadas pelo próprio AspectJ. A parte da linguagem de implementação fornece ferramentas para compilação, *debug* e integração com IDEs.

2.1.4 Exemplo de código utilizando aspectos

A Figura 2.3 apresenta um exemplo didático (SANT'ANNA, 2004) para mostrar a diferença entre a implementação em Java e AspectJ de um mesmo programa. Esse exemplo mostra o código de um programa simples de elementos gráficos. A implementação em Java (lado esquerdo da figura) é formada pelas classes *Point*, *Line* e *Display* (essa última não é apresentada na figura). A implementação em AspectJ (lado direito da figura) é constituída das mesmas classes, mais o aspecto *DisplayUpdating*. Esse exemplo mostra que, após a chamada dos métodos *setX* e *setY* da classe *Point* e dos métodos *setP1* e *setP2* da classe *Line*, o método *update* da classe *Display* é chamado. Na implementação em Java, a chamada ao método *update* da classe *Display* fica espalhada pelos quatro métodos, pois é feita explicitamente no final de cada um deles (linhas 9, 13, 25 e 29). Na solução em AspectJ, essa chamada fica localizada apenas no aspecto *DisplayUpdating* (linha 37), e é introduzida nas classes pelo *pointcut move* (linhas 30-34).

```
01 class Point {
02  private int x = 0, y = 0;
03
04  int getX() { return x; }
05  int getY() { return y; }
06
07  void setX(int x) {
08    this.x = x;
09    Display.update();
10  }
11  void setY(int y) {
12    this.y = y;
13    Display.update();
14  }
15 }
16
17 class Line {
18  private Point p1, p2;
19
20  Point getP1() { return p1; }
21  Point getP2() { return p2; }
22
23  void setP1(Point p1) {
24    this.p1 = p1;
25    Display.update();
26  }
27  void setP2(Point p2) {
28    this.p2 = p2;
29    Display.update();
30  }
31 }
```

```
01 class Point {
02  private int x = 0, y = 0;
03
04  int getX() { return x; }
05  int getY() { return y; }
06
07  void setX(int x) {
08    this.x = x;
09  }
10  void setY(int y) {
11    this.y = y;
12  }
13 }
14
15 class Line {
16  private Point p1, p2;
17
18  Point getP1() { return p1; }
19  Point getP2() { return p2; }
20
21  void setP1(Point p1) {
22    this.p1 = p1;
23  }
24  void setP2(Point p2) {
25    this.p2 = p2;
26  }
27 }
28
29 aspect DisplayUpdating {
30  pointcut move():
31    call(void Line.setP1(Point))||
32    call(void Line.setP2(Point))||
33    call(void Point.setX(int))  ||
34    call(void Point.setY(int));
35
36  after() returning: move() {
37    Display.update();
38  }
39 }
```

Figura 2.3: O mesmo programa em Java (lado esquerdo) e AspectJ (lado direito) (SANT'ANNA, 2004)

2.2 Linha de Produto de Software

O reuso sistemático de software consiste de uma mudança da abordagem de construção de sistemas únicos para o desenvolvimento de famílias de sistemas. Dessa forma, o enfoque de linha de produto de software (LPS) consiste da proposta de construção sistemática de software baseada em uma família de produtos.

Gimenes e Travassos (2002) definem LPS como um conjunto de produtos de software com características suficientemente similares para permitir a definição de uma infra-estrutura comum de estruturação dos itens que compõem os produtos e a parametrização das diferenças entre os produtos. Em outras palavras, pode-se dizer que entre os produtos de uma LPS há um conjunto comum de requisitos e há também variabilidades significativas entre outros requisitos. Por exemplo, uma família de sistemas para gestão de bilhetes eletrônicos de transporte possui um núcleo de funcionalidades comum, tal como, o uso de um cartão eletrônico para pagamento de corridas. Ela possui ainda características variáveis, como o tipo de integração entre corridas que funciona de forma diferente entre os diversos produtos que podem ser gerados para essa LPS.

Essas características variáveis entre os produtos de uma LPS são denominadas de variabilidades. Elas são descritas em termos de pontos de variação e variantes. Um ponto de variação é o lugar específico no artefato da LP ao qual a decisão de projeto está associada. Cada ponto de variação está associado a um conjunto de variantes que correspondem às alternativas de projeto para uma variabilidade (HEUMANS e TRIGAUX, 2003 *apud* OLIVEIRA JUNIOR *et al*., 2005).

Clements *et al.* (2006) afirma que a organização de LPS engloba três atividades principais: desenvolvimento do núcleo de artefatos, desenvolvimento do produto e gerenciamento da LPS. A atividade de desenvolvimento do núcleo de artefatos pode ser chamada de engenharia de domínio, assim como a atividade de desenvolvimento do produto pode ser chamada de engenharia de aplicação. O núcleo de artefatos (do inglês *core assets)* é constituído dos recursos comuns a toda a LPS e pode incluir planos, requisitos, projetos, documentação, testes e código. O desenvolvimento dos produtos normalmente é apoiado por geradores de aplicação, como o Captor, descrito no final deste capítulo.

2.2.1 LPS-BET

Vários trabalhos enfatizam diferentes alternativas para a implementação de variabilidades no contexto de LPS. Donegan e Masiero (2007) discutem questões sobre projeto de famílias de produtos. Para ilustrar essas questões foi desenvolvida a LPS-BET, uma LPS para gestão de bilhetes eletrônicos de transporte. Donegan (2008) propôs uma adaptação do processo ESPLEP (do inglês *Evolutionary Software Product Line Engineering Process*) (Gomaa, 2004) e, para cada um dos quatro incrementos foram construídos, respectivamente, o núcleo da LPS e cada uma das três aplicações-referência da linha (os produtos para as cidades de Fortaleza/CE, Campo Grande/MS e São Carlos/SP). Os requisitos especificados para cada um dos sistemas foram analisados e comparados em detalhe, realizando um mapeamento entre os requisitos dos três sistemas. Dessa forma, a comparação permitiu a extração das características e a modelagem da LPS em termos de similaridades e variabilidades.

A arquitetura da LPS-BET é baseada em componentes caixa-preta, pelas vantagens de maior separação de interesses e facilidade de manutenção. No entanto, Donegan (2008) discute a possibilidade de uso de aspectos (POA) para a representação de requisitos não-funcionais e de variabilidades dessa família de produtos.

Foi considerado importante ter uma aplicação completa logo no começo do processo de desenvolvimento, optando assim, por usar os ciclos de incrementos horizontais, que permitem a geração de uma dessas aplicações a cada incremento. Desse modo, foram planejados quatro incrementos, um produzindo o núcleo da LPS-BET e os outros uma aplicação da linha, os quais são descritos a seguir, até o final da subseção, baseando-se no trabalho de Donegan (2008):

- ***Incremento 1:*** Desenvolvimento de casos de uso do núcleo da LPS.
- ***Incremento 2:*** Reuso do núcleo (incremento 1) e desenvolvimento de casos de uso específicos da aplicação-referência de Fortaleza.
- ***Incremento 3:*** Reuso do núcleo (incremento 1) e de alguns casos de uso desenvolvidos (incremento 2) e desenvolvimento de casos de uso específicos da aplicação-referência de Campo Grande.

- ***Incremento 4:*** Reuso do núcleo (incremento 1) e de alguns casos de uso desenvolvidos (incrementos 2 e 3) e desenvolvimento de casos de uso específicos da aplicação-referência de São Carlos.

A Figura 2.4 ilustra o processo de desenvolvimento da LPS-BET, evidenciando as disciplinas de requisitos, análise, projeto e implementação, assim como, a elaboração da receita de composição que deve ser usada posteriormente na engenharia de aplicação. Na concepção do primeiro ciclo foi planejado de forma geral o desenvolvimento da LPS e em detalhes, o desenvolvimento do núcleo e foram extraídos os requisitos das três aplicações-referência. Com base nos requisitos foi modelado o diagrama de casos de uso da LPS; verificaram-se quais casos de uso deveriam existir em cada aplicação para que seus requisitos fossem alcançados e então se iniciou a elaboração pela especificação detalhada dos casos de uso do núcleo. A partir da especificação dos casos de uso, a análise e o projeto

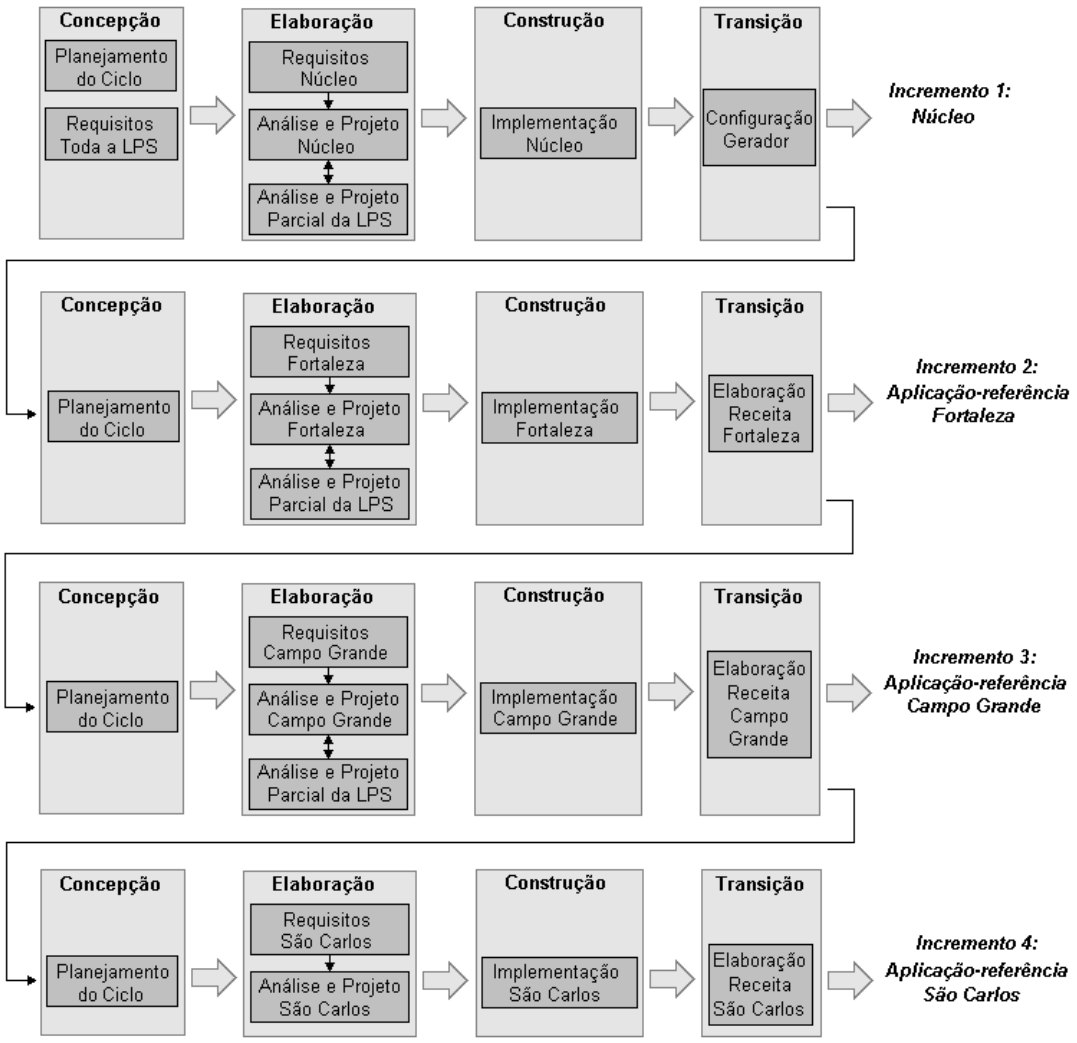

Figura 2.4: Visão geral do processo de desenvolvimento da LPS-BET (DONEGAN, 2008)

do núcleo foram feitos. No projeto parcial do primeiro incremento foram analisados alguns casos de uso que existem para Fortaleza, Campo Grande e São Carlos. Esse projeto parcial ofereceu retroalimentação para o projeto do núcleo, para que pudesse ser refinado. Com base nos requisitos e no projeto, foi implementado o núcleo da LPS, o qual foi reusado para as implementações das aplicações-referência nos outros ciclos. Na transição do núcleo foi feita a configuração inicial do gerador a ser usado na engenharia de aplicações da LPS-BET. A associação entre os requisitos e a análise e o projeto de ciclos diferentes não está sendo mostrada na Figura 2.4, que deve ficar subentendida.

Primeiramente foi desenvolvido um documento de requisitos para cada um dos sistemas. A partir dos documentos de requisitos dos três sistemas, o diagrama de características foi modelado. A Tabela 2.3 mostra as características (opcionais e alternativas) que foram incorporadas ao núcleo da LPS (mostrado na Figura 2.5) para obter cada uma das três aplicações-referência.

Para a disciplina de requisitos, além dos documentos de requisitos, foram feitos o diagrama de casos de uso e as especificações dos casos de uso. Foi feito um mapeamento entre requisitos, características e casos de uso para poder rastreá-los posteriormente. Os casos de uso básicos são especificados no incremento 1 e os casos de uso opcionais são especificados no incremento em que forem usados primeiro.

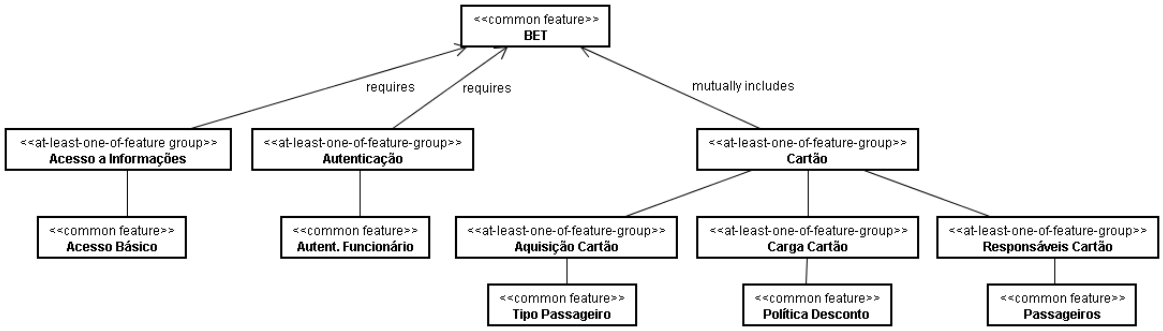

Figura 2.5: Diagrama de características do núcleo da LPS-BET (DONEGAN, 2008)

A arquitetura da LPS-BET, projetada na fase de Elaboração, é mostrada na Figura 2.6. Ela combina duas instâncias do padrão arquitetural cliente/servidor em multicamadas (GOMAA, 2004 *apud* DONEGAN, 2008): uma arquitetura em três camadas para o cliente do ônibus e seu servidor; e uma arquitetura em quatro camadas para o sistema web de informação, ambos compartilhando a mesma camada de domínio de negócio.

Tabela 2.3: Características opcionais ou alternativas para aplicações da LPS-BET (DONEGAN, 2008)

Característica	Fortaleza	Campo Grande	São Carlos
Acesso Adicional		X	X
Autenticação Passageiro			X
Forma de Integração			
- Terminal	X	X	
- Integração			
* Tempo		X	X
* Linha Integração		X	X
* Número de Viagens de Integração		X	
Pagamento de Cartão	X		
Restrição de Cartões			
- Número de Cartões		X	
- Combinação de Cartões			X
Empresas Usuárias	X	X	
Limite de Passagens			X

Em um nível mais baixo, cada camada possui uma arquitetura interna de componentes, como é mostrado para o núcleo da LPS-BET na Figura 2.7. Há uma instância de cada módulo servidor (Servidor WEB, Servidor Ônibus, Domínio BET e Persistência) e várias ocorrências de dois módulos clientes (Cliente Ônibus e Interface de Usuário). Externo a esses módulos, há um ambiente com usuários e dispositivos externos que interagem com a LPS. Internos a esses módulos estão os componentes projetados para tratar as características comuns da LPS-BET. Componentes de sistema e de negócio (representados respectivamente com os estereótipos «*system component*» e «*business component*» na Figura 2.7) foram derivados usando diretrizes sugeridas por Cheesman e Daniels (2001) *apud* Donegan (2008), enquanto componentes controladores foram derivados usando sugestões de Gomaa (2004) *apud* Donegan (2008).

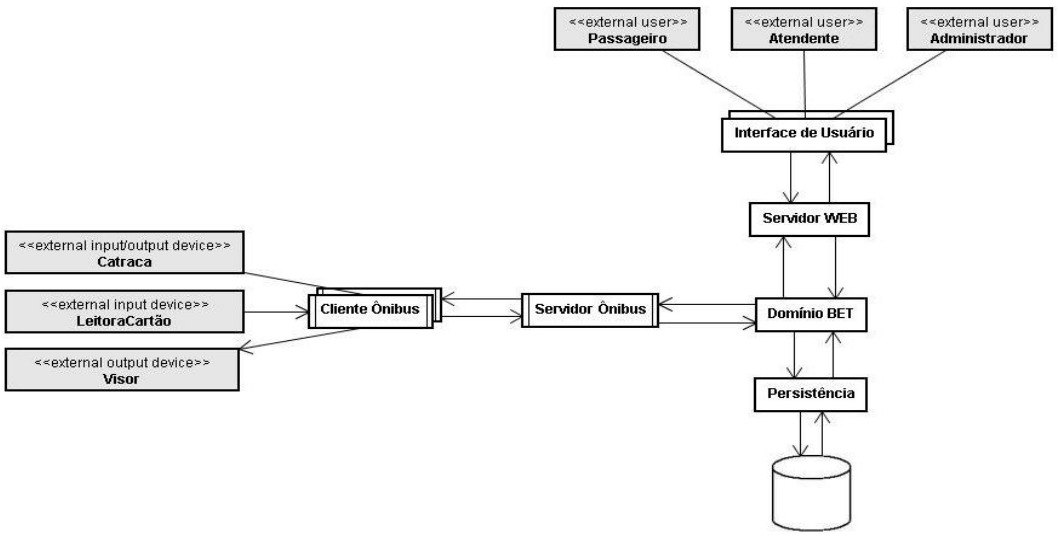

Figura 2.6: Arquitetura da LPS-BET (DONEGAN, 2008)

Componentes de sistema são identificados a partir das operações que emergem do diagrama de casos de uso e componentes de negócio são obtidos do refinamento do modelo conceitual. As operações dos componentes de sistema usam operações das interfaces de negócio. Um componente controlador executa conceitualmente um *statechart* e controla uma parte do sistema e um componente coordenador gerencia diversos componentes controladores (GOMAA, 2004) *apud* (DONEGAN, 2008).

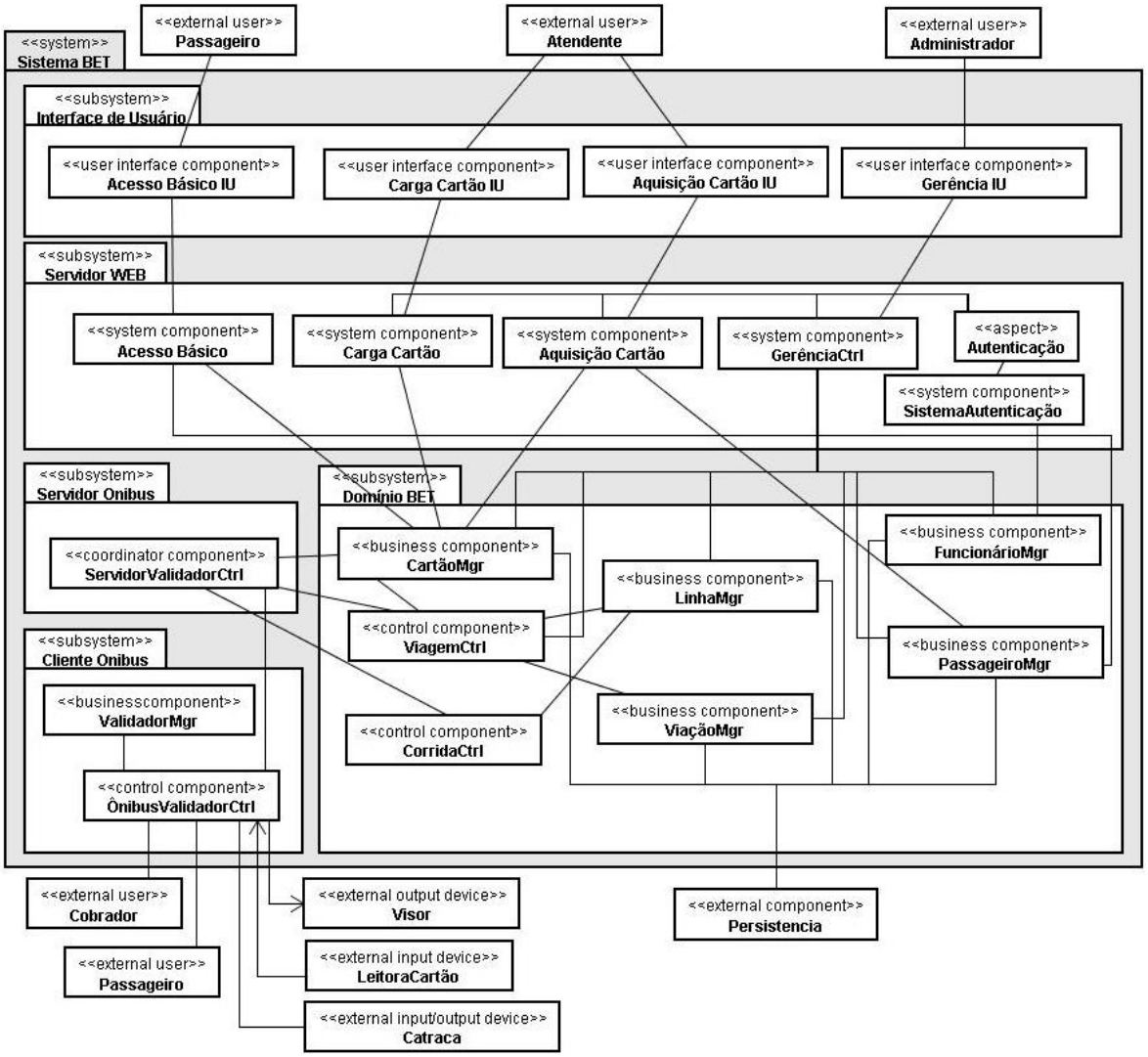

Figura 2.7: Arquitetura de componentes do núcleo da LPS-BET (DONEGAN, 2008)

A Figura 2.8 mostra em mais detalhes parte da arquitetura de componentes do núcleo, que inclui interfaces requeridas e fornecidas, assim como, componentes que as requerem e as fornecem. Os dispositivos externos e o subsistema do Cliente Ônibus, presentes em um ônibus, são mostrados do lado esquerdo da figura e o subsistema do Servidor Ônibus e seus

componentes relacionados (partes do subsistema Domínio BET) para processar uma viagem de um passageiro ou uma corrida de um ônibus são mostrados do lado direito. Nessa figura, os dispositivos externos LeitoraCartão, Catraca e Visor são colocados como sendo parte da LPS, pois é necessário que eles sejam projetados e implementados para o funcionamento da LPS, passando a ser representados também por componentes da LPS-BET. Os usuários cobrador e passageiro (representados na Figura 2.7) interagem com a LPS por meio das interfaces fornecidas ILeitora e IGirarCatraca pelos componentes LeitoraCartão e Catraca, respectivamente.

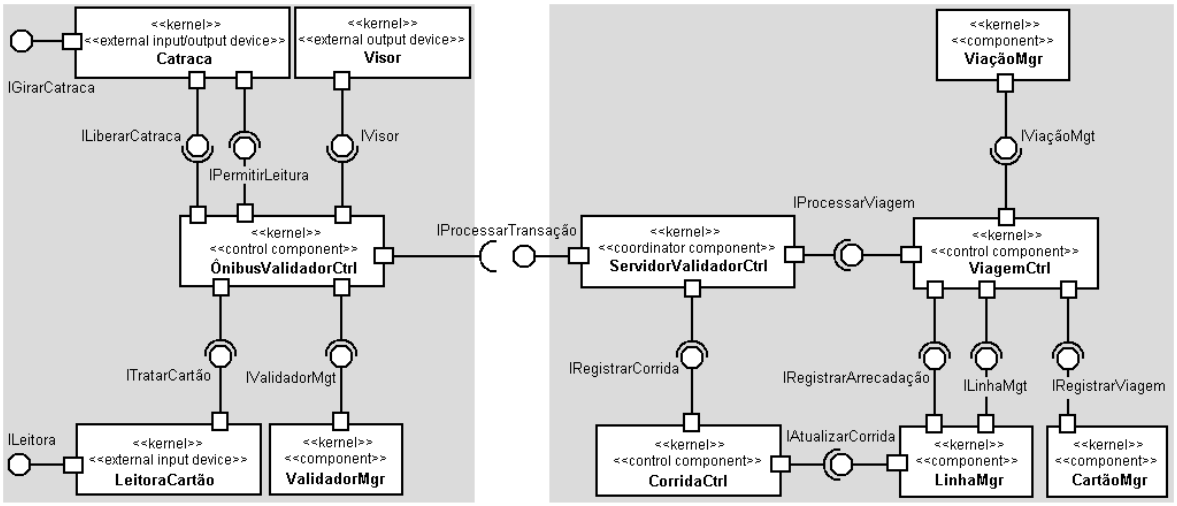

Figura 2.8: Parte dos componentes e interfaces do núcleo da LPS-BET (DONEGAN, 2008)

No projeto parcial foram analisadas as características relacionadas às *Formas de Integração* e a característica de *Pagamento de Cartão*. Essas características foram escolhidas por tratarem dois tipos diferentes de projeto de variabilidades: projeto com novas classes, que possuem menor acoplamento e projeto com novas subclasses, que possuem maior acoplamento. Com esse projeto, foi possível refinar a arquitetura de componentes do núcleo, para que menos retrabalho fosse necessário posteriormente nos outros incrementos, ao inserir variabilidades do grupo de características de *Forma de Integração* e da característica *Pagamento de Cartão*. A partir da arquitetura de componentes do núcleo, pôde-se fazer a implementação dos seus componentes durante a fase de construção. A organização da implementação da LPS-BET pode ser vista na Figura 2.9, distinguindo a parte básica (à esquerda) e as variabilidades (à direita).

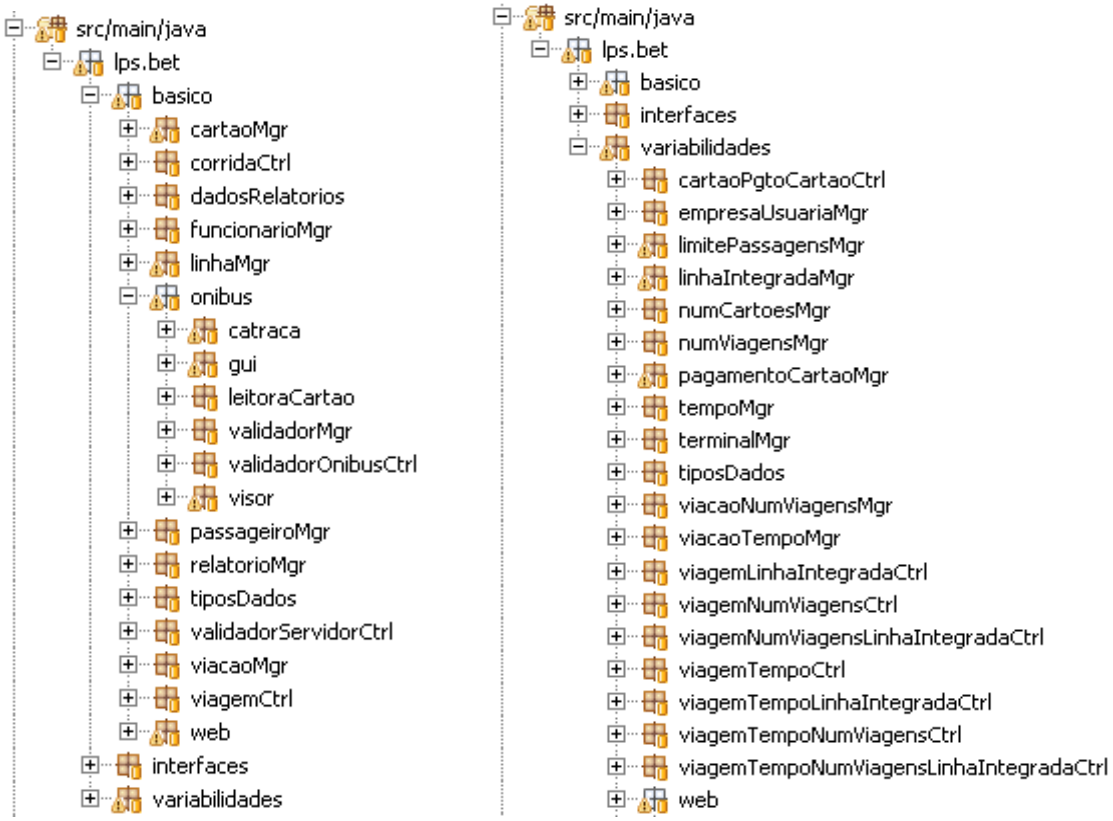

Figura 2.9: Organização da implementação da LPS-BET (DONEGAN, 2008)

Do ponto de vista técnico, o *framework* Spring (Spring, 2008) foi usado para construir a arquitetura de componentes. Segundo Donegan (2008), o Spring oferece a possibilidade de declarar quais componentes serão usados e como eles se ligam uns aos outros, por meio de interfaces. Há várias maneiras de definir essa configuração dos componentes usando Spring, a mais comum é por meio de arquivos XML, conhecidos como "contextos de aplicação". Na linguagem definida por esse padrão XML, os componentes são denominados *beans*. Cada *bean* pode ter um conjunto de propriedades (*property*, no XML), por meio das quais esse *bean* pode referenciar outros *beans*. A linguagem de configuração de *beans* do Spring corresponde a uma ADL[1] (*Architecture Definition Language*) não-hierárquica com conectores implícitos (MCVEIGH, 2008) *apud* (DONEGAN, 2008). É uma forma de definir a linguagem de descrição de domínio pelo modo como os componentes da LPS são interligados para associar características da LPS e obter uma arquitetura baseada em componentes.

[1] ADLs são linguagens formais que podem ser usadas para representar a arquitetura de um sistema de software

Na Figura 2.10 pode ser vista a configuração de um dos componentes mostrados na arquitetura de componentes do núcleo. Neste exemplo o componente ViagemCtrl requer interfaces de outros quatro componentes: interfaceRegistrarViagem, interfaceRegistrarArrecadacao, interfaceLinhaMgt, e interfaceViacaoMgt. Na nomenclatura do Spring, essa requisição de interfaces é conhecida pelo nome de "dependência". O uso de interfaces é extremamente importante, já que assim o desenvolvedor pode indicar declarativamente quais implementações dessas interfaces serão usadas, sem interferir na relação de dependência entre os componentes. Essa flexibilidade é explorada na LPS-BET para trocar as implementações de modo a gerar novos produtos da linha.

Depois de finalizada a fase de Construção do núcleo, a fase de Transição foi iniciada e pôde-se partir para a configuração inicial do gerador de aplicação configurável Captor (descrito na próxima subseção) para o novo domínio da LPS-BET. Para isso foi necessário considerar todas as características da linha e a combinação válida dessas características. Portanto, a configuração inicial foi feita sem possuir ainda as variabilidades implementadas e baseando-se no diagrama de características da LPS-BET.

```
1  <bean id="ViagemCtrl" class="lps.bet.basico.viagemCtrl.ViagemCtrl">
2      <property name="interfaceRegistrarViagem">
3          <ref bean="CartaoMgr"/>
4      </property>
5      <property name="interfaceRegistrarArrecadacao">
6          <ref bean="LinhaMgr"/>
7      </property>
8      <property name="interfaceLinhaMgt">
9          <ref bean="LinhaMgr"/>
10     </property>
11     <property name="interfaceViacaoMgt">
12         <ref bean="ViacaoMgr"/>
13     </property>
14 </bean>
```

Figura 2.10: *Bean* para injeção de dependências do componente ViagemCtrl (DONEGAN, 2008)

2.3 Captor

Geradores de aplicação são ferramentas que recebem a especificação de uma aplicação de software, validam essa especificação e geram artefatos automaticamente, por isso eles costumam ser usados para gerar os vários produtos de uma LPS. Geradores de aplicação configuráveis (GAC) são geradores que podem ser adaptados para fornecer apoio em domínios diferentes. Um GAC configurado deve apresentar as mesmas características de um gerador de aplicação específico, ou seja, receber uma especificação armazená-la em

meio persistente, permitir a edição e reedição dessa especificação, validá-la e transformá-la em artefatos de software. A principal vantagem dessa abordagem em relação a um gerador de aplicações específicas é que as atividades de configuração de um GAC podem ser menos custosas do que a construção completa de um gerador de aplicação específico, diminuindo o custo de uso do gerador e permitindo a utilização das técnicas de geração para domínios que normalmente não apresentam viabilidade econômica para investir nessas ferramentas Shimabukuro *et al* (2006). Na Figura 2.10 é apresentada uma visão geral de um gerador de aplicação configurável.

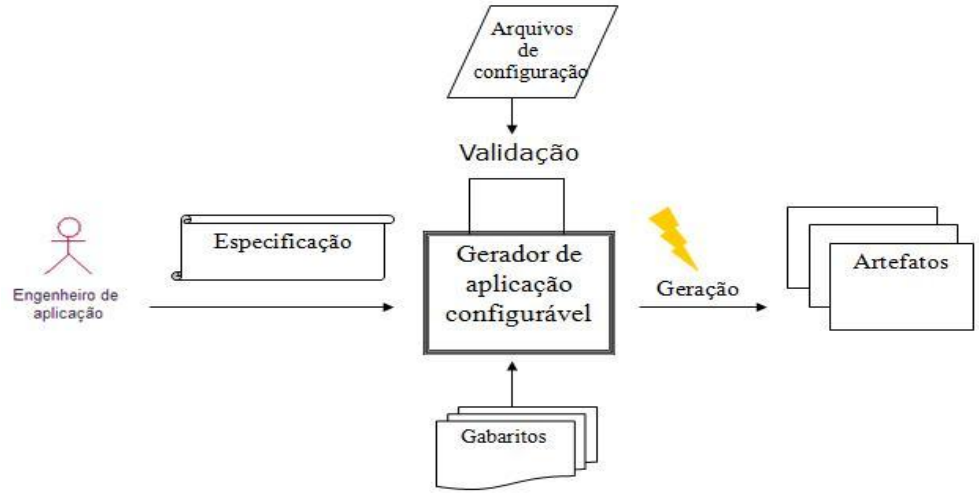

Figura 2.10: Gerador de Aplicação Configurável (SHIMABUKURO, 2006)

Assim, surgiu o Captor proposto por Shimabukuro (2006), que tem por objetivo facilitar a geração de artefatos de software em diferentes domínios. Vários tipos de artefatos de software podem ser gerados pelo Captor, entre eles código, documentação e testes.

Capítulo 3
Desenvolvimento da LPS-BET

3.1 Decisões de projeto da LPS-BET

Nesta seção são abordadas questões relacionadas ao projeto da LPS-BET desenvolvida com arquitetura baseada em componentes caixa-preta para a implementação de variabilidades. Além disso, é considerado o uso de aspectos como alternativa para implementação de variabilidades da LPS-BET em uma arquitetura baseada em componentes. As questões descritas neste capítulo são baseadas no trabalho de Donegan (2008).

3.1.1 Projeto baseado em componentes caixa-preta

O projeto baseado em componentes é uma das formas mais utilizadas por vários autores para projetar LPS (GOMAA, 2004; ATKINSON *et al*., 2001; POHL *et al*., 2005 *apud* DONEGAN, 2008). O Desenvolvimento de Software Baseado em Componentes (DSBC) surgiu como uma perspectiva de desenvolvimento de software caracterizada pela composição de partes já existentes (DONEGAN, 2008). Os componentes podem ser implementados como caixa-preta. Deste modo, são feitas apenas combinações de componentes por meio das interfaces e criação de novos componentes sem alterar o código interno dos componentes.

Algumas características variantes da LPS-BET mostradas na Tabela 2.3 são discutidas nesta seção para ilustrar como as decisões de projeto foram influenciadas pelas decisões tomadas em relação ao processo adotado para o desenvolvimento da LPS e ao tipo de componente.

Além da característica *Terminal*, o grupo de características *Forma de Integração* também contém o grupo de características *Integração* (Tabela 2.3). Integração consiste no uso de linhas integradas pré-definidas e/ou no uso de ônibus dentro de um intervalo de tempo. Pode também haver um número máximo de viagens de integração permitidas dentro de um

intervalo de tempo. Essas características (*Linha de Integração, Tempo e Número de Viagens de Integração*) podem ser vistas na Figura 3.1.

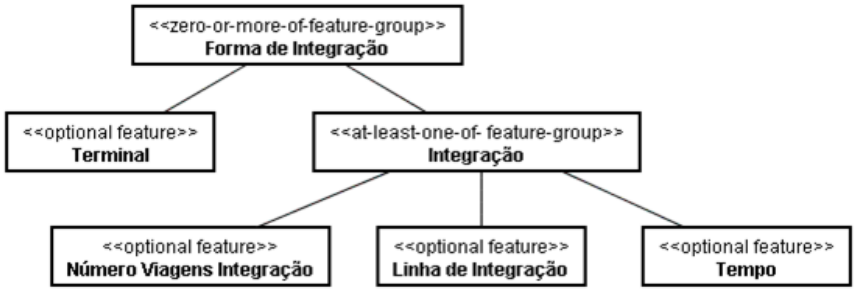

Figura 3.1: Parte do diagrama de características relacionadas à característica *Forma de Integração* (DONEGAN, 2008)

A característica *Linha de Integração* foi analisada e projetada parcialmente na fase de Elaboração do ciclo de desenvolvimento de Fortaleza e depois essa característica teve seu projeto detalhado na fase de Elaboração do ciclo de Campo Grande. Por fim, no ciclo de São Carlos, o projeto e a sua implementação puderam ser reusados.

No modelo de classes a característica *Linha de Integração* é representada por uma classe opcional chamada LinhaIntegrada e é associada com a classe Linha, que podem ser vistas na Figura 3.2. A classe opcional LinhaIntegrada é encapsulada em um novo componente chamado LinhaIntegradaMgr. Então um componente controlador é criado. Ele é mostrado na Figura 3.3.

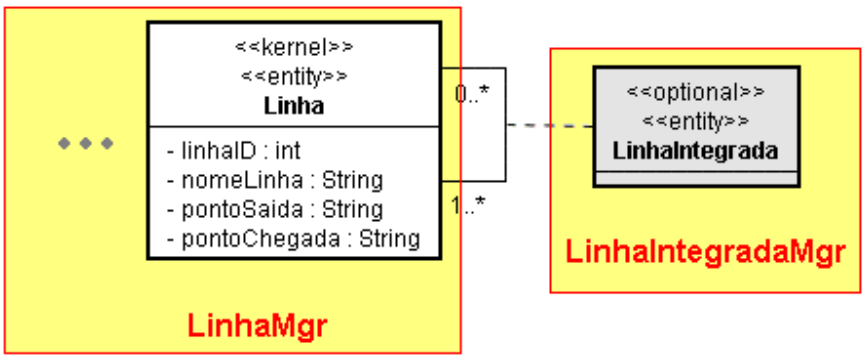

Figura 3.2: A característica *Linha de Integração* no modelo de classes (DONEGAN, 2008)

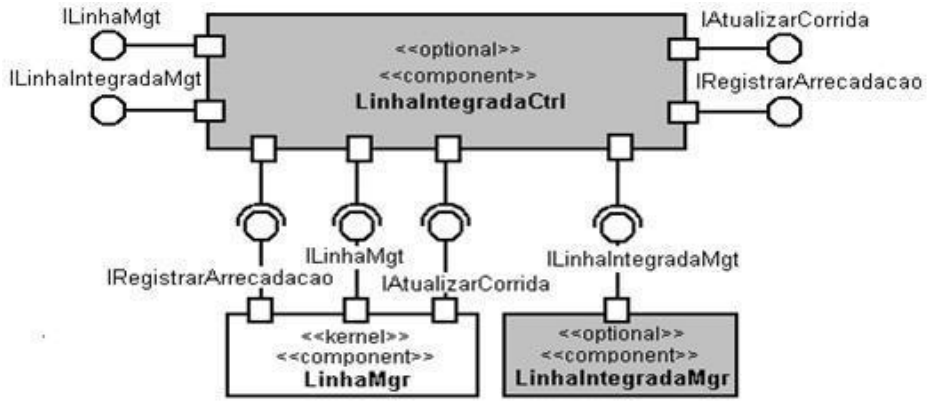

Figura 3.3: Componentes para implementar a característica *Linha de Integração* (adaptado de DONEGAN, 2008)

Esse componente controlador (LinhaIntegradaCtrl) tem uma interface nova ILinhaIntegradaMgt além daquelas de componentes correspondentes de versões anteriores. Essa interface é requerida por um controlador de viagem (representado pelo componente ViagemCtrl) para validar a integração de uma viagem de acordo com as características selecionadas do grupo de características de *Integração*. No caso da característica *LinhaIntegrada*, o controlador de viagem precisa validar se a linha corresponde a uma linha integrada. Logo, o componente ViagemCtrl também precisa ser alterado para validar a informação fornecida de acordo com as regras de negócio da característica *Linha Integrada*. Donegan (2008) apresenta as seguintes possibilidades de soluções: substituir o componente ViagemCtrl por outro; usar composição, projetando um novo componente; ou separar o interesse adicional em um aspecto (KICZALES *et al*., 1997; SUVÉE *et al*., 2006 *apud* DONEGAN, 2008).

As outras duas características do grupo de características de *Integração* são *Tempo* (existente em Campo Grande e São Carlos) e *Número de Viagens de Integração* (existente apenas em Campo Grande). A Figura 3.4 apresenta as classes EmpresaViaria, SistemaViarioUrbano e Tarifa que fazem parte da LPS-BET e são encapsuladas no componente ViacaoMgr. As características *Tempo* e *Número de Viagens* de *Integração* implicam em pontos de variação na classe SistemaViarioUrbano, alterando atributos e operações dessa classe que podem ser colocados em subclasses (representadas pelo estereótipo «*variant*» na Figura 3.4), contrastando com os exemplos anteriores, em que era necessário inserir uma nova classe no modelo.

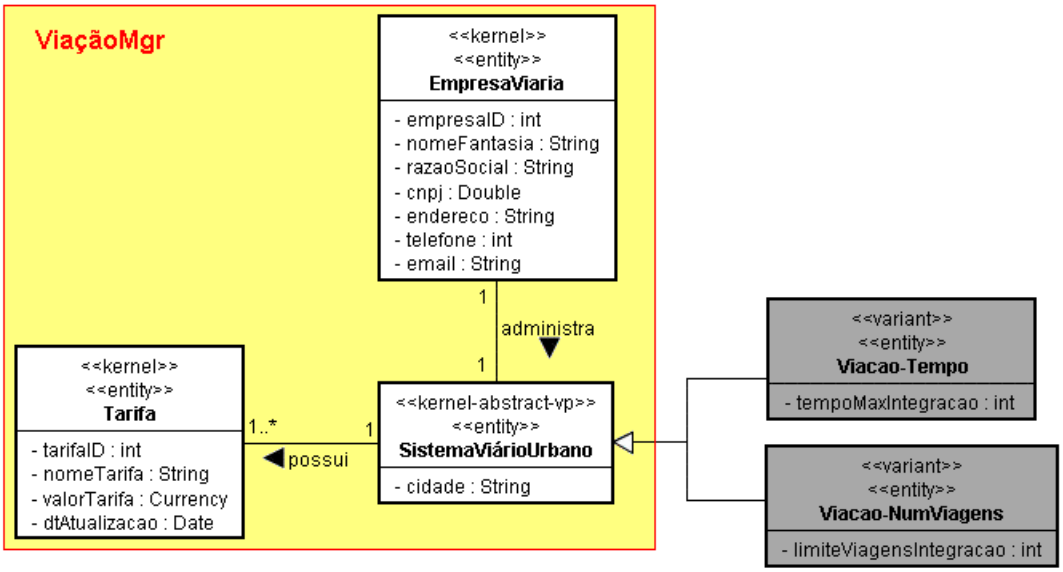

Figura 3.4: As características *Tempo* e *Número de Viagens de Integração* no modelo de classes (DONEGAN, 2008)

Diferentes opções foram analisadas para o projeto da LPS-BET, como podem ser vistas no trabalho de Donegan (2008). A opção escolhida foi a de usar classes independentes da classe básica SistemaViarioUrbano e separar essas características em novos componentes chamados TempoMgr e NumViagensMgr, com as interfaces ITempoMgt e INumViagensMgt respectivamente como interfaces fornecidas. Esses componentes caixa-preta são mostrados na Figura 3.5. As operações da interface ITempoMgt são mostradas na Figura 3.6.

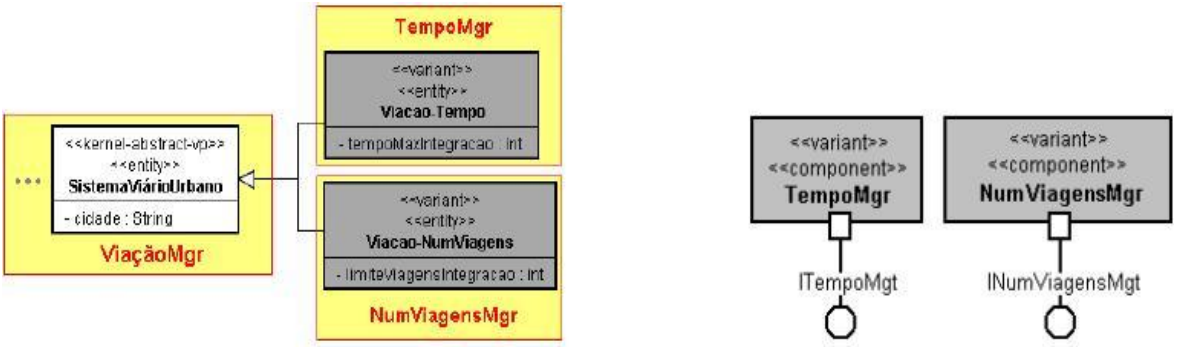

Figura 3.5: Componentes caixa-preta de negócio para o grupo de características de *Integração* (DONEGAN, 2008)

Os componentes TempoMgr e NumViagensMgr são projetados similarmente, pois ambos são variantes do mesmo componente e possuem o mesmo tipo de associação com o componente básico e operações requeridas. Embora a seguir seja apresentado somente o

projeto do componente TempoMgr, pode-se assumir um projeto análogo para o componente NumViagensMgr.

```
1  public interface ITempoMgt {
2
3    public int buscarTempo();
4    public void criarTempo(int tempoMaxIntegracao);
5    public void atualizarTempo(int tempoMaxIntegracao);
6
7  }
```

Figura 3.6: Operações da Interface ITempoMgt (DONEGAN, 2008)

É preciso um componente controlador para obter e tratar informações dos componentes TempoMgr e ViaçãoMgr. Uma solução possível é aquela mostrada para LinhaIntegradaMgr na Figura 3.3. Outra semelhança ao exemplo da *Linha Integrada* é a necessidade de adicionar uma interface no componente controlador, nesse caso porque o controlador de viagem (ViagemCtrl) precisa verificar uma possível integração, dependendo do intervalo de tempo desde a última viagem. Portanto, o componente ViagemCtrl também precisa ser alterado para implementar essa nova variabilidade e requerer a nova interface do componente TempoMgr fornecida pelo seu controlador.

Após uma série de análises de soluções, Donegan (2008) verificou que a melhor solução usando componentes caixa-preta seria a de já projetar inicialmente o componente ViagemCtrl de tal modo que ele pudesse ser reusado por todas as aplicações e um componente controlador separado de ViagemCtrl pudesse ser adicionado para uma variabilidade da característica de *Integração*, que no exemplo deste trabalho corresponde a *Tempo*. A solução é mostrada na Figura 3.7. O componente TempoViagemCtrl é criado e não demanda mudanças nos componentes básicos. Esse componente fornece a mesma interface IProcessarViagem para o componente ServidorValidadorCtrl, que assim não requer alteração. O novo componente requer a interface ITempoMgt para verificar uma possível integração por tempo e a interface IProcessarViagem, implementada pelo componente ViagemCtrl, para poder encaminhar o controle de uma viagem que não corresponda a uma integração.

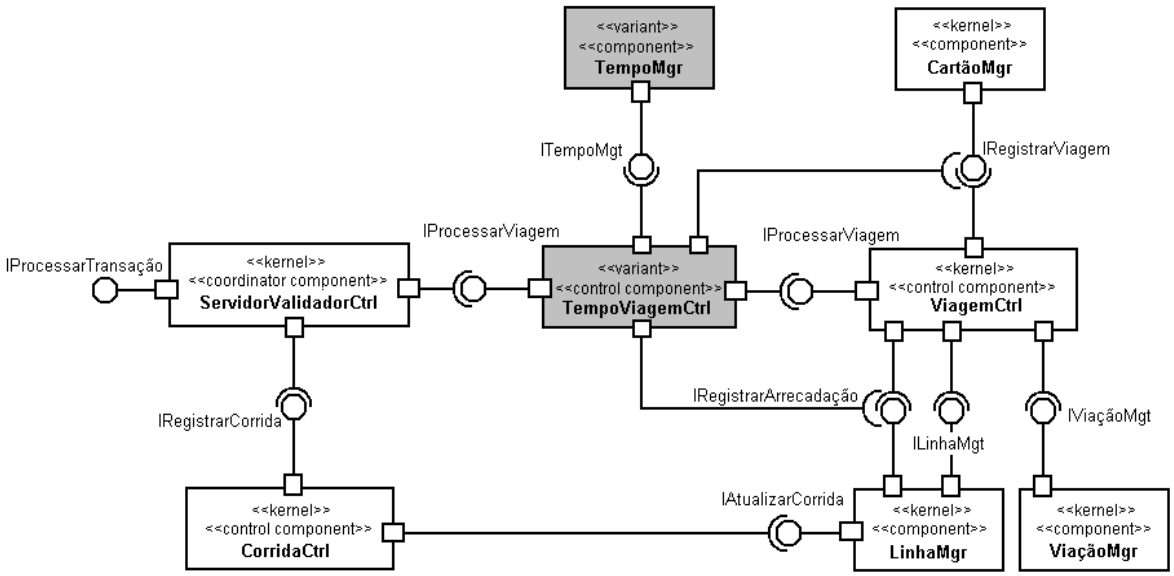

Figura 3.7: Solução para integrar a característica *Tempo* na arquitetura da LPS-BET (DONEGAN, 2008)

O componente TempoViagemCtrl encapsula uma classe que implementa o método processarViagem da interface IProcessarViagem (interface mostrada na Figura 3.8), que será requerida pelo componente ServidorValidadorCtrl. A implementação do componente TempoViagemCtrl é apresentada na Figura 3.9. A classe TempoViagemCtrl requer quatro interfaces (linhas 03-06): ITempoMgt (fornecida pelo componente variante TempoMgr), IRegistrarViagem (fornecida por CartãoMgr), IRegistrarArrecadacao (fornecido por LinhaMgr) e IProcessarViagem (fornecido pelo ViagemCtrl). O método processarViagem (linhas 08-15) é responsável por chamar o método processarViagem do componente ViagemCtrl se não estiver dentro do intervalo de tempo de uma viagem de integração. Para verificar isso, o método verificarIntegracao (linhas 17-29) é chamado com os parâmetros cartaoID e onibusID (linha 10). Esse método compara o tempo máximo de integração com o tempo que passou desde a última viagem e tem como resposta se a viagem corresponde ou não a uma integração. Caso haja integração por tempo, o método processarIntegração (linhas 31-36) é chamado para atualizar os dados da viagem do passageiro e o número de passageiros na corrida do ônibus, sem fazer a arrecadação do valor da passagem.

```
1  public interface IProcessarViagem {
2
3          public String processarViagem(int cartaoID, int onibusID);
4
5  }
```

Figura 3.8: Interface IProcessarViagem (DONEGAN, 2008)

```
1  public class TempoViagemCtrl implements IProcessarViagem{
2
3    ITempoMgt interfaceTempoMgt;
4    IRegistrarViagem interfaceRegistrarViagem;
5    IRegistrarArrecadacao interfaceRegistrarArrecadacao;
6    IProcessarViagem interfaceProcessarViagem;
7
8    public String processarViagem(int cartaoID, int onibusID) {
9      String estado="INT-NOK";
10     estado = verificarIntegracao(cartaoID, onibusID);
11     if (!estado.equals("INT-OK"))
12       return interfaceProcessarViagem.processarViagem(cartaoID, onibusID);
13     else
14       return estado;
15   }
16
17   private String verificarIntegracao(int cartaoID, int onibusID){
18     String estado = "INT-NOK";
19     long tempoDecorrido = Long.MAX_VALUE;
20     int tempoMaxIntegracao = interfaceTempoMgt.buscarTempo();
21     Viagem viagem = interfaceRegistrarViagem.buscarUltimaViagem(cartaoID);
22     if (viagem != null){
23       Calendar horaUltimaViagem = viagem.getHora();
24       tempoDecorrido = Calendar.getInstance().getTimeInMillis() - horaUltimaViagem.getTimeInMillis();
25       if (tempoDecorrido <= tempoMaxIntegracao)
26         estado = processarIntegracao(onibusID, viagem);
27     }
28     return estado;
29   }
30
31   private String processarIntegracao(int onibusID, Viagem viagem){
32     interfaceRegistrarArrecadacao.registrarArrecadacao(onibusID, 0);
33     viagem.setNumViagens(viagem.getNumViagens()+1);
34     interfaceRegistrarViagem.alterarViagem(viagem);
35     return "INT-OK";
36   }
37 }
```

Figura 3.9: Classe TempoViagemCtrl do componente de mesmo nome com implementação da interface IProcessarViagem (DONEGAN, 2008)

Em termos do arquivo XML no contexto da aplicação, a configuração desses componentes e a ligação dos componentes relacionados ao TempoViagemCtrl é mostrada na Figura 3.10. O componente ServidorValidadorCtrl (linhas 01-11) requer três interfaces (linhas 02, 05 e 08), fornecidas pelos componentes CorridaCtrl, TempoViagemCtrl e CartãoMgr. Diferentemente da configuração do componente no núcleo, que requeria a interfaceProcessarViagem implementada pelo componente ViagemCtrl no lugar do TempoViagemCtrl (linha 06).

Após a implementação da classe do componente TempoViagemCtrl, notou-se que o mesmo requer quatro interfaces. Isso também pode ser visto no contexto da aplicação da Figura 3.10. Uma das interfaces requeridas é a interfaceTempoMgt (linha 14) fornecida pelo componente TempoMgr (linha 15), que por sua vez, é definido nas linhas 28-32 e não requer interface, pois é um componente de negócio e apenas possui dependência da

propriedade sessionFactory (linhas 29-31) que é relacionada à persistência da entidade Viação-Tempo.

```
1  <bean id="ServidorValidadorCtrl" class="lps.bet.basico.servidorValidadorCtrl.ServidorValidadorCtrl">
2          <property name="interfaceRegistrarCorrida">
3                  <ref bean="CorridaCtrl"/>
4          </property>
5          <property name="interfaceProcessarViagem">
6                  <ref bean="TempoViagemCtrl"/>
7          </property>
8          <property name="interfaceCartaoMgt">
9                  <ref bean="CartaoMgr"/>
10         </property>
11 </bean>
12
13 <bean id="TempoViagemCtrl" class="lps.bet.variabilidades.tempoViagemCtrl.TempoViagemCtrl">
14         <property name="interfaceTempoMgt">
15           <ref bean="TempoMgr"/>
16         </property>
17         <property name="interfaceRegistrarViagem">
18           <ref bean="CartaoMgr"/>
19         </property>
20         <property name="interfaceRegistrarArrecadacao">
21           <ref bean="LinhaMgr"/>
22         </property>
23         <property name="interfaceProcessarViagem">
24           <ref bean="ViagemCtrl"/>
25         </property>
26 </bean>
27
28 <bean id="TempoMgr" class="lps.bet.variabilidades.tempoMgr.TempoDAO">
29         <property name="sessionFactory">
30                 <ref bean="sessionFactory"/>
31         </property>
32 </bean>
```

Figura 3.10: *Beans* relacionados ao componente TempoViagemCtrl (DONEGAN, 2008)

3.1.2 Uso de aspectos para implementação de variabilidades da LPS-BET

Na concepção de Soares e Borba (2002), a POA propõe não apenas uma decomposição funcional, mas também ortogonal do problema. POA permite que a implementação de um sistema seja separada em requisitos funcionais e não–funcionais.

Cottenier e Elrad (2004) *apud* Donegan (2008) ressaltam que a integração entre aspectos e componentes não é uma tarefa simples, especialmente em relação aos interesses transversais do tipo intra-componente. Isto ocorre porque os aspectos podem alterar o funcionamento de componentes, infringindo alguns dos princípios desta técnica de desenvolvimento de software, como o encapsulamento e a certificação de qualidade. A fim de solucionar esse impasse, existem abordagens que propõem permitir que os aspectos atuem somente nas operações disponíveis nas interfaces dos componentes, por exemplo, o trabalho de Eler (2006) *apud* Donegan (2008).

O uso de aspectos para representar variabilidades de LPS também tem sido discutido por Figueiredo *et al*. (2008). A sua motivação vem da convicção de outros pesquisadores de que o uso de POA é efetivo para apoiar a implementação de variabilidades e prolongar a estabilidade do projeto de LPS. Figueiredo *et al*. (2008) e seus colaboradores realizaram diversos experimentos em duas LPS. Para cada uma delas, variabilidades foram implementadas com e sem o uso de aspectos. Em seus experimentos, foi estabelecida uma série de métricas, a fim de observar vantagens e desvantagens de se utilizar POA nesse contexto. Ao final das atividades, obtiveram evidências de que, em alguns casos, os aspectos de fato favorecem a evolução da LPS.

Inicialmente a solução de projeto de variabilidades da LPS-BET foi feita e implementada considerando uma arquitetura puramente baseada em componentes, usando aspectos apenas para representar requisitos não-funcionais. A partir dessa solução baseada em componentes, foi projetada outra solução baseada em componentes e em aspectos. Nessa solução os aspectos passam a ser utilizados para representar variabilidades. Como são usados componentes do tipo caixa-preta, os aspectos são usados de modo a interceptar apenas operações das interfaces. Donegan (2008) realizou um estudo de caso com o objetivo de comparar ambas as soluções de implementação de variabilidades: a solução de projeto e de implementação de variabilidades apenas com componentes e a solução usando componentes e aspectos.

O uso de aspectos é ilustrado considerando o mesmo exemplo da seção 3.1.1: as características do grupo de características *Integração* (*Tempo, Linha de Integração e Número de Viagens de Integração*). Como visto no projeto desses componentes, os controladores dos três variantes possuem um comportamento bastante semelhante, pois todos requerem alterações no mesmo componente (ViagemCtrl) e devem ser capazes de processar uma integração pela requisição de duas interfaces: IRegistrarViagem e IRegistrarArrecadação. Eles diferem em termos do componente de negócio que irão precisar (TempoMgr, NumViagensMgr e LinhaIntegradaMgr) e, portanto, diferem em relação à interface que irão requerer. Isso significa que eles serão distintos em relação às regras de negócio para verificação da existência de uma integração (Donegan, 2008).

Desse modo, com a escolha de usar aspectos para implementar essas variabilidades, um aspecto abstrato foi usado para generalizar as similaridades desses três tipos de integração. Cada variabilidade corresponde a um aspecto e implementa um aspecto abstrato (IntegraçãoCtrl), herdando o ponto de junção, o adendo e um método (processarIntegração), além de duas interfaces comuns que são requeridas. O aspecto IntegraçãoCtrl define um método abstrato chamado verificarIntegração que deve ser implementado por cada aspecto especializado. Esses aspectos são mostrados na Figura 3.12.

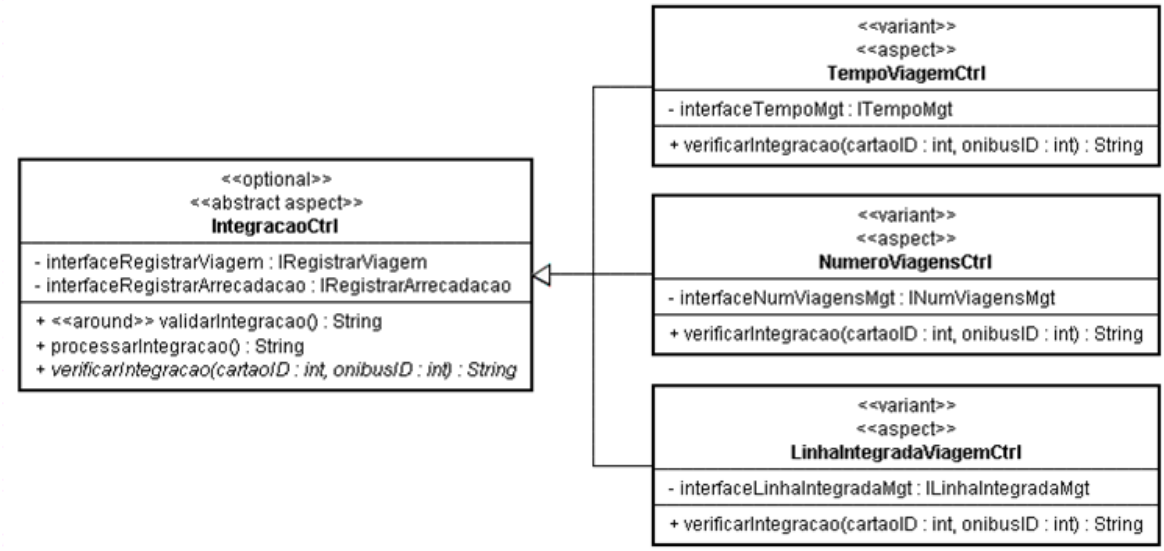

Figura 3.12: Aspecto abstrato e aspectos contratos para representar a característica *Integração* (DONEGAN, 2008)

A implementação do aspecto abstrato IntegraçãoCtrl usando AspectJ (KICZALES *et al.*, 2001 *apud* DONEGAN, 2008) é apresentada na Figura 3.13. As interfaces requeridas são definidas nas linhas 03 e 04. O ponto de junção entrecorta apenas a chamada dos métodos da interface IProcessarViagem (linha 06) e então o adendo do tipo *around* é executado (linhas 08-17). Se houver integração (estado recebe "INT-OK"), o adendo retorna o estado ao método da interface entrecortada, sem proceder a execução do método interceptado. Entretanto, caso não haja integração (estado é igual a "INT-NOK"), o método entrecortado procede com sua execução normal (linha 16).

```
1  public abstract aspect IntegracaoCtrl {
2
3    ICartaoMgt interfaceCartaoMgt;
4    IRegistrarArrecadacao interfaceRegistrarArrecadacao;
5
6    pointcut validarIntegracao() : call(String lps.bet.interfaces.IProcessarViagem.*(..));
7
8    String around() : validarIntegracao() {
9      String estado="INT-NOK";
10     Object[] args = thisJoinPoint.getArgs();
11     if (args.length > 0) {
12       int cartaoID = (Integer) args[0];
13       int onibusID = (Integer) args[1];
14       estado = verificarIntegracao(cartaoID, onibusID);
15     }
16     return !estado.equals("INT-OK") ? proceed() : estado;
17   }
18
19   public String processarIntegracao(int onibusID, Viagem viagem) {
20     interfaceRegistrarArrecadacao.registrarArrecadacao(onibusID, 0);
21     viagem.setNumViagens(viagem.getNumViagens()+1);
22     interfaceCartaoMgt.alterarViagem(viagem);
23     return "INT-OK";
24   }
25
26   public abstract String verificarIntegracao(int cartaoID, int onibusID);
27
28 }
```

Figura 3.13: Implementação do aspecto abstrato IntegraçãoCtrl (DONEGAN, 2008)

A Figura 3.14 mostra o projeto da característica *Tempo* usando aspectos. Esse projeto é correspondente à solução mostrada na Figura 3.7 que usa apenas componentes. A principal diferença é que o componente ServidorValidadorCtrl não precisa requerer a interface IProcessarViagem de um novo componente, mas continua requerendo a interface fornecida

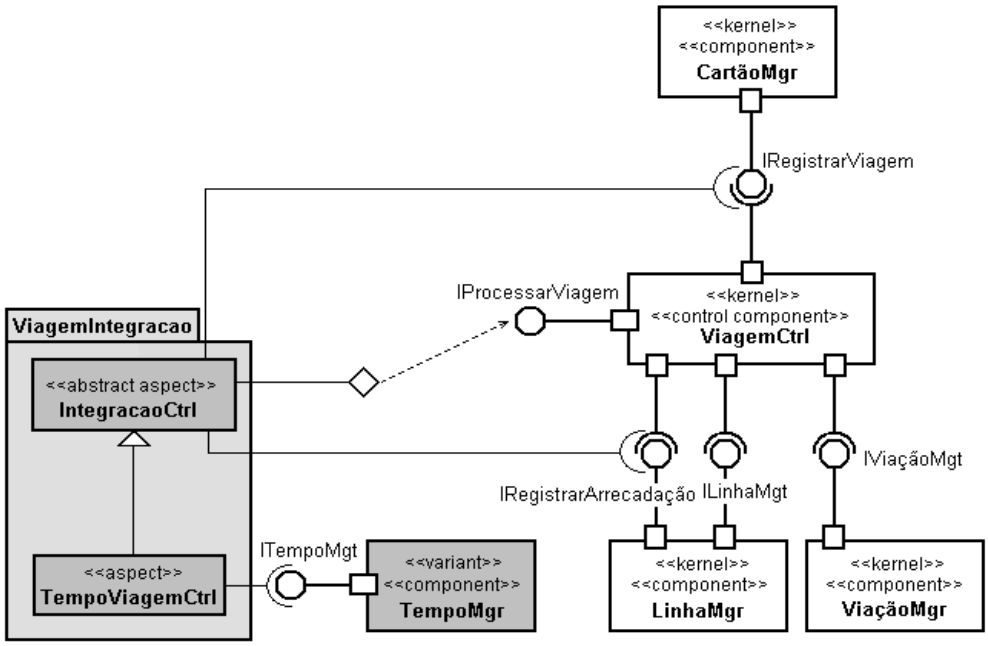

Figura 3.14: Uma solução usando aspectos para adicionar a característica *Tempo* na arquitetura (DONEGAN, 2008)

pelo componente básico ViagemCtrl. O aspecto IntegracaoCtrl entrecorta a interface e é estendido pelo aspecto TempoViagemCtrl que requer as operações da interface ITempoMgt do componente de negócio TempoMgr.

A implementação do aspecto TempoViagemCtrl é mostrada na Figura 3.15. Ele estende o aspecto abstrato IntegraçãoCtrl (linha 01), assim sendo, pode utilizar o método processarIntegração (linha 16). O aspecto verifica se a integração se aplica comparando o tempo decorrido desde a última viagem (linha 14) com o tempo máximo de integração, obtido a partir do método buscarTempo da interface ITempoMgt (linha 10). Como esse aspecto estende o IntegraçãoCtrl, ele também apenas entrecorta a interface IProcessarViagem.

```
1  public aspect TempoViagemCtrl extends IntegracaoCtrl{
2
3    ITempoMgt interfaceTempoMgt;
4
5    public String verificarIntegracao(int cartaoID, int onibusID) {
6
7      String estado = "INT-NOK";
8      long tempoDecorrido = Long.MAX_VALUE;
9      Viagem viagem = interfaceCartaoMgt.buscarUltimaViagem(cartaoID);
10     int tempoMaxIntegracao = interfaceTempoMgt.buscarTempo();
11
12     if (viagem != null) {
13       Calendar horaUltimaViagem = viagem.getHora();
14       tempoDecorrido = Calendar.getInstance().getTimeInMillis() - horaUltimaViagem.getTimeInMillis();
15       if (tempoDecorrido <= tempoMaxIntegracao)
16         estado = processarIntegracao(onibusID, viagem);
17     }
18     return estado;
19   }
20 }
```

Figura 3.15: Implementação do aspecto ViagemTempoCtrl (DONEGAN, 2008)

Donegan (2008) verificou que para a solução com componentes pode não ser necessário alterar componentes do núcleo para a adição de variabilidades, mas ao menos é preciso alterar a configuração no contexto de aplicação do componente que implementa a interface requerida pelo componente básico, como visto anteriormente na Figura 3.10. Portanto, os componentes básicos em si podem não possuir conhecimento da inserção de variabilidades, pois são usadas interfaces com nomes equivalentes. Contudo, como a interface é fornecida por um componente diferente, é necessário alterar a configuração dos componentes básicos requerendo essa interface pela referência (ref) ao componente que implementa a interface. Ao usar aspectos essa alteração no contexto da aplicação não é necessária, pois o aspecto entrecorta a interface sem que os componentes básicos tenham conhecimento do mesmo. No exemplo da característica de *Tempo*, para a solução com aspectos o componente básico

ServidorValidadorCtrl não seria modificado nem na implementação nem na sua configuração. A configuração para esse exemplo usando aspecto é mostrada na Figura 3.16.

```
<bean id="ServidorValidadorCtrl" class="lps.bet.basico.servidorValidadorCtrl.ServidorValidadorCtrl">
        <property name="interfaceRegistrarCorrida">
                <ref bean="CorridaCtrl"/>
        </property>
        <property name="interfaceProcessarViagem">
                <ref bean="ViagemCtrl"/>
        </property>
        <property name="interfaceCartaoMgt">
                <ref bean="CartaoMgr"/>
        </property>
</bean>

<bean id="ViagemTempoCtrlAspecto" class="lps.bet.variabilidades.ViagemTempoCtrl" factory-method="aspectOf">
        <property name="interfaceTempoMgt">
                        <ref bean="TempoMgr"/>
        </property>
        <property name="interfaceCartaoMgt">
                        <ref bean="CartaoMgr"/>
        </property>
        <property name="interfaceRegistrarArrecadacao">
                        <ref bean="LinhaMgr"/>
        </property>
</bean>

<bean id="TempoMgr" class="lps.bet.variabilidades.tempoMgr.TempoDAO">
        <property name="sessionFactory">
                <ref bean="sessionFactory"/>
        </property>
</bean>
```

Figura 3.16: *Beans* relacionados ao aspecto TempoViagemCtrl (DONEGAN, 2008)

A solução aqui apresentada para o uso de aspectos não implementa a variabilidade como um todo com o uso de aspectos, pois deseja-se continuar com as vantagens do uso de componentes caixa-preta. Os componentes de negócio são extraídos do refinamento do modelo conceitual, da mesma forma como são extraídos no projeto, sem o uso de aspectos.

Em soluções para implementar uma variabilidade tendo um componente básico A que pode ser reusado sem problemas e é preciso adicionar um componente B, a solução usando aspectos é similar àquela mostrada nesta seção. O aspecto entrecorta a interface do componente A, realiza um processamento e, de acordo com o resultado, retorna o fluxo para o componente A. O processamento feito pelo aspecto pode consistir em uma chamada a operações do componente B, caso o componente seja de negócio, ou em uma implementação das operações que estariam no componente B, caso o componente seja de controle ou de sistema. Como explicado anteriormente, a vantagem dessa solução é que evita a necessidade de alterar a configuração de componentes que requerem o componente A do núcleo (DONEGAN, 2008).

Para soluções de projeto baseadas em componentes em que é necessário substituir um componente A' do núcleo por outro B' com a variabilidade a ser implementada, a solução usando aspectos requer que o aspecto entrecorte as operações das interfaces do componente A', não permitindo que as operações do componente A' sejam executadas. Pode ser feita uma solução em que o aspecto chama as operações do componente B' (componente de negócio) ou implementa as operações que estariam no componente B' (componente de controle ou de sistema). O aspecto executaria de modo a não permitir que a aplicação-referência utilize as operações do componente do núcleo (componente A'). Mesmo que o componente não seja utilizado, o componente se encontra conectado aos outros componentes e a arquitetura da aplicação-referência não corresponde à arquitetura de fato, podendo dificultar manutenções posteriores na aplicação. Nesse ponto, a solução usando componentes possui vantagem sobre a solução usando aspectos, pois a arquitetura dos componentes corresponde realmente à forma como a aplicação funciona (DONEGAN, 2008).

3.2 Tecnologias de Implementação da LPS-BET

A LPS-BET foi modelada seguindo a notação UML na ferramenta Jude (JUDE, 2008). Os artefatos desenvolvidos ao longo do processo de desenvolvimento foram mantidos sob o controle de versão do TortoiseSVN (TIGRIS, 2008) e permaneceram hospedados em um repositório do Google[2] (GOOGLE, 2008).

A implementação da LPS-BET realizada por Donegan (2008) foi feita usando a linguagem de programação orientada a objetos Java (seguindo as diretrizes do modelo JavaBeans) e a linguagem de programação orientada a aspectos AspectJ (ASPECTJ TEAM, 2008) no ambiente Eclipse (Eclipse, 2008). O banco de dados adotado foi o PostgreSQL (POSTGRESQL, 2008), porém, a aplicação pode executar sobre qualquer outra base de dados relacional. Para realizar o mapeamento objeto-relacional e poder persistir os dados na base foi usado o *framework* Hibernate (HIBERNATE, 2008). Além disso, o Spring (SPRING, 2008) foi utilizado para fazer inversão de controle (especificamente a injeção de dependência) (Fowler, 2004 *apud* Donegan, 2008), usar o MVC (*Model-View-Controller*) para a parte web e facilitar o uso do Hibernate utilizando-se de sua integração com o

[2] http://code.google.com/p/bet

framework. O Velocity (APACHE, 2008) foi aplicado como mecanismo para geração de páginas, pois permite um maior desacoplamento entre a apresentação e a lógica de negócios, podendo reforçar mais a separação entre ambos. Como ferramenta de *build* foi utilizado o Maven 2 (APACHE, 2008), por ser declarativo e permitir uma maior abstração. O Maven já possui um conjunto de convenções pré-estabelecidas que facilitam as atividades de *build*.

Durante a construção e a transição dos ciclos de desenvolvimento da LPS, Donegan (2008) realizou testes do domínio BET usando a estratégia de amostras de aplicações (*Sample Application Strategy*) (POHL *et al*., 2005 *apud* DONEGAN, 2008). Essa estratégia consiste da validação na engenharia de domínio de algumas aplicações que sejam sistemas representativos da LPS-BET para validar os ativos centrais do domínio. Com isso, algumas variabilidades da LPS já são testadas, porém, não elimina a necessidade de realizar testes na engenharia de aplicação, pois nem todas as combinações de aplicações possíveis são testadas na engenharia de domínio (DONEGAN, 2008).

Os testes consistiram basicamente de testes funcionais manuais, a verificação e a validação das funcionalidades do núcleo operacional e das aplicações-referência foram feitas, assim como de algumas outras aplicações representativas usando combinações de variabilidades. Apenas foram preparados testes unitários para as classes com maior dificuldade de entendimento no nível de implementação, nesse caso usando o JUnit (JUNIT, 2008) *apud* Donegan (2008). Os problemas encontrados foram registrados no *site* do repositório para serem corrigidos.

Com essa infra-estrutura fez-se a engenharia de domínio da LPS-BET. A partir disso, pôde-se fazer a engenharia das aplicações. Donegan (2008) usou um gerador de aplicação configurável (GAC), pois assim, o gerador pôde ser configurado para o domínio BET. O gerador usado foi desenvolvido pelo próprio grupo de pesquisa – o Captor (SHIMABUKURO, 2006) – que usa a linguagem de transformação XSLT (do inglês *eXtensible Stylesheet Language Transformations*) (W3C, 2008) *apud* (DONEGAN, 2008) para a implementação da linguagem de modelagem de aplicação.

Capítulo 4
Desenvolvimento de variabilidades da LPS-BET com aspectos

Este capítulo é dividido da seguinte forma: na Seção 4.1 é analisado o uso de aspectos para a implementação de variabilidades relacionadas ao uso de cartões existentes na LPS-BET; na Seção 4.2 são mostrados os passos para a preparação do Ambiente BET criado por Donegan (2008); na Seção 4.3 é apresentado um tutorial para a preparação do Ambiente BET desenvolvido neste trabalho e na Seção 4.4 são discutidas as dificuldades encontradas no decorrer deste trabalho.

4.1 Uso de aspectos para implementação de variabilidades relacionadas ao de uso de cartões

Neste trabalho, foi realizado um estudo de caso baseado no projeto da LPS-BET para analisar uma solução de implementação de variabilidades relacionadas ao uso de cartões usando aspectos. Para isso foi considerada a característica *Pagamento de Cartão.* Essa característica variável provém da aplicação-referência de Fortaleza que possui mais outras duas características (*Terminal e Empresas Usuárias*) além daquelas características já implementadas para o incremento do núcleo da LPS-BET, como pode ser visto na Tabela 2.3. O pagamento de cartão acontece quando a pessoa adquire um cartão.

O objetivo é analisar o uso de aspectos para representar variabilidades nessa arquitetura (aplicação-referência de Fortaleza, ilustrada na Figura 4.1). Assim sendo, são feitos estudos para avaliar os prós e contras do uso de aspectos em uma arquitetura baseada em componentes caixa-preta. A Figura 2.5 mostra parte do diagrama de características relacionadas ao uso de cartões.

Como visto na arquitetura de componentes para a aplicação-referência de Fortaleza na Figura 4.1, a característica *Pagamento de Cartão* requer alteração no componente CartãoPgtoCartãoCtrl o qual deve ser capaz de processar o pagamento pela requisição de duas interfaces: ICartaoMgt e IPagtoCartaoMgt. Os componentes de negócio necessários são CartãoMgr e PagamentoCartãoMgr. A Figura 4.2 apresenta as classes TipoPassageiro,

Cartao, Viagem e Pagamento que fazem parte da LPS-BET e são encapsuladas no componente CartaoMgr. As características TipoPassag-PgtoCartao e Pagamento-PgtoCartao são encapsuladas no componente PagamentoCartãoMgr e implicam em pontos

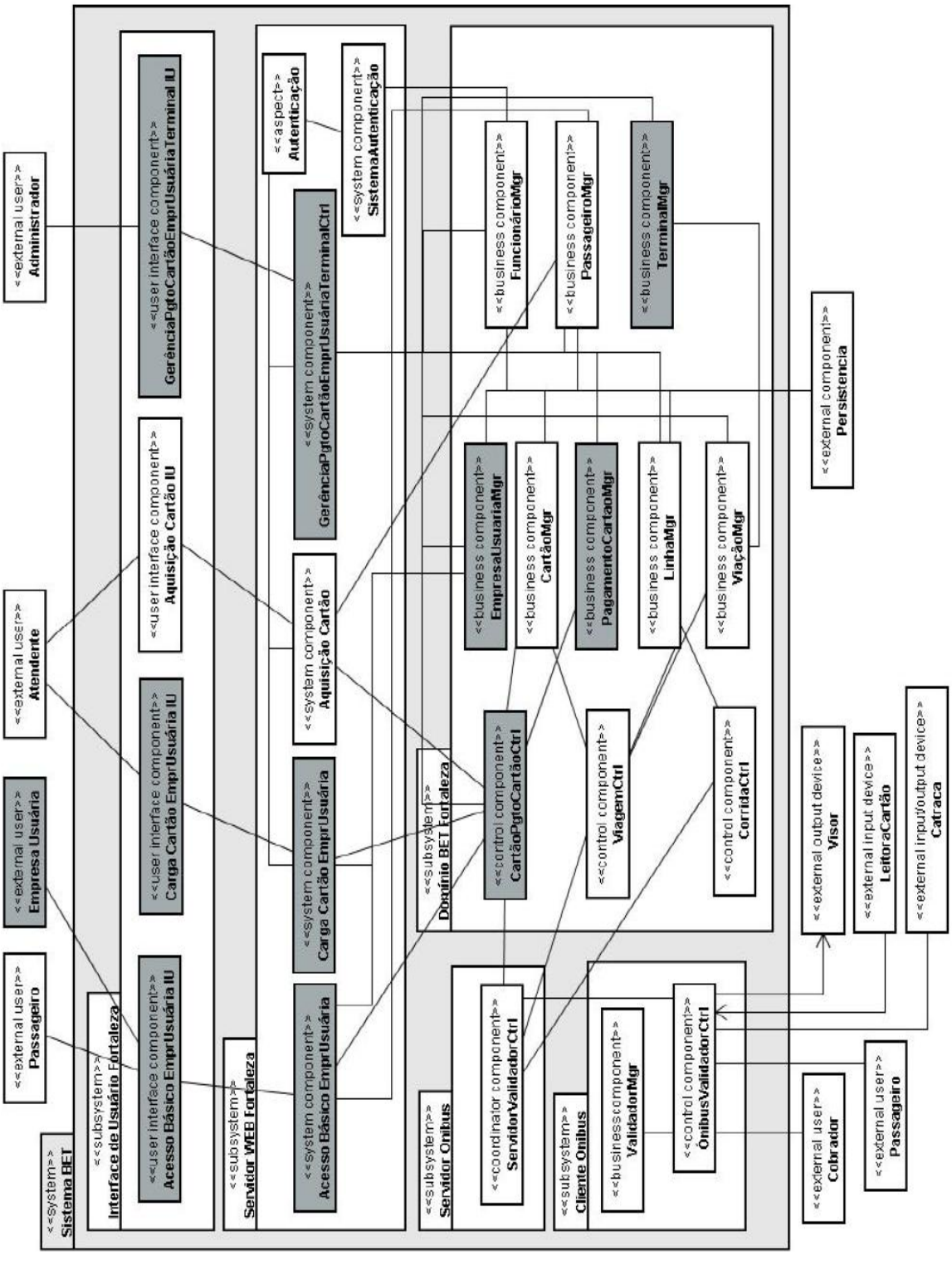

Figura 4.1: Arquitetura de Componentes para a aplicação-referência de Fortaleza (DONEGAN, 2008)

de variação na classe TipoPassageiro e Pagamento respectivamente, alterando atributos e operações dessas classes que podem ser colocados em subclasses (representadas pelo estereótipo «*variant*» na Figura 4.2). As setas são para indiciar que é uma especialização, se fosse implementado como caixa-branca dentro de um mesmo componente, seria feita uma especialização, usando *extends*. A diferença de componentes caixa-branca em relação aos componentes caixa-preta é que se pode ter acesso ao código das classes dos componentes e nesse caso seria possível fazer a especialização.

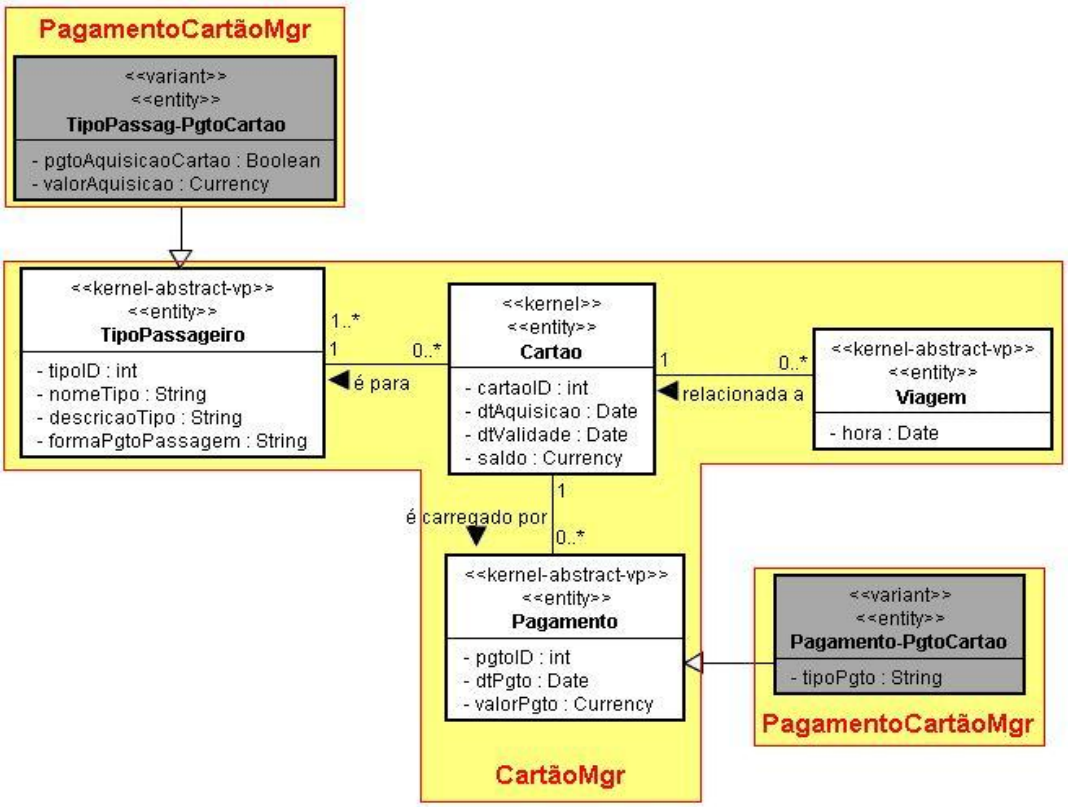

Figura 4.2: Componentes caixa-preta de negócio para a característica *Pagamento de Cartão*

A opção escolhida no trabalho de Donegan (2008) foi a de usar classes independentes das classes básicas TipoPassageiro e Pagamento e separar essa característica em um novo componente chamado PagamentoCartãoMgr, com a interface IPagtoCartaoMgt como interface fornecida. Esse componente caixa-preta é mostrado na Figura 4.2. As operações da interface IPagtoCartaoMgt são mostradas na Figura 4.3.

```
1  package lps.bet.variabilidades.pagamentoCartaoMgr;
2
3  import java.util.Collection;
8
9  public interface IPagtoCartaoMgt {
10
11     public void criarPagtoCartao(PagamentoPagtoCartao pgtoCartao);
12     public void alterarPagtoCartao(PagamentoPagtoCartao pgtoCartao);
13     public PagamentoPagtoCartao buscarPagamentoPagtoCartao(int pgtoID);
14
15     public int buscarUltimoPagamento();
16
17     public void registrarTipoPassagPagtoCartao(TipoPassagPagtoCartao tipoPagamento);
18
19     public TipoPassagPagtoCartao buscarTipoPassagPagtoCartao(int tipoID);
20     public TipoPassagPagtoCartao buscarTipoPagtoPorTipoPassageiro(TipoPassageiro tipoPassageiro);
21     public int buscarUltimoTipoPassageiro();
22
23     public Collection<TipoPassagPagtoCartao> buscarTodosTipos();
24     public Collection<PagamentoPagtoCartao> buscarPagtosCartao();
25     public Collection<TipoPassagPagtoCartao> buscarTiposPagtosPermitidos(Collection<TipoPassageiro> tiposPermitidos)
26 }
```

Figura 4.3: Operações da Interface IPagtoCartaoMgt

A solução usando componentes caixa-preta feita por Donegan (2008) foi a de projetar inicialmente o componente CartãoMgr de tal modo que ele pudesse ser reusado por todas as aplicações. Essa solução é mostrada na Figura 4.4. O componente CartãoPgtoCartãoCtrl fornece a interface ICartaoMgt para o componente ServidorValidadorCtrl e para o componente AquisiçãoCartão. O componente CartãoPgtoCartãoCtrl requer a interface IPagtoCartãoMgt e ICartãoMgt para realizar o pagamento de cartões. Ele é usado apenas quando existir o Pagamento de Cartão, o que somente ocorre na aplicação-referência de Fortaleza.

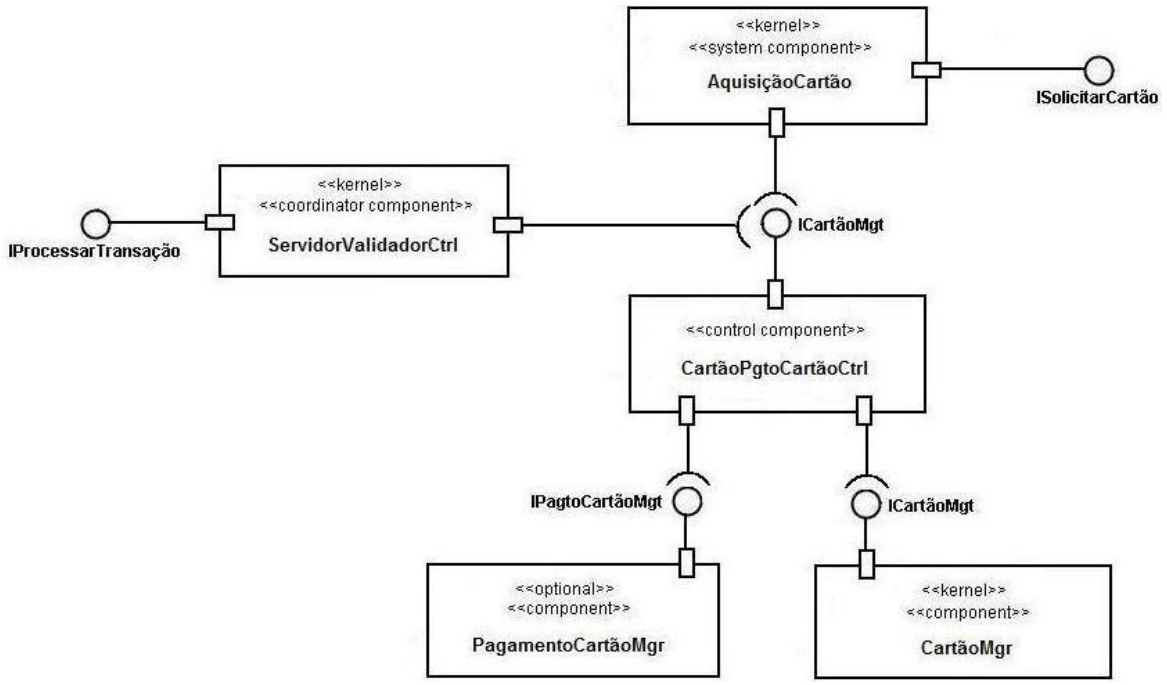

Figura 4.4: Solução para integrar a característica *Pagamento de Cartão* na arquitetura da LPS-BET

Em termos do arquivo XML de contexto da aplicação, a configuração do componente CartãoPgtoCartãoCtrl e a ligação dos componentes relacionados a ele são mostradas na Figura 4.5.

```
<bean id="ControlValidadorServidor"
    class="lps.bet.basico.controlValidadorServidor.ControlValidadorServidor" lazy-
    init="default" autowire="default" dependency-check="default">
    <property name="interfaceCartaoMgt">
          <ref bean="CartaoMgr" />
    </property>
  </bean>

<bean id="AquisicaoCartao"
    class="lps.bet.basico.web.aquisicaoCartao.AquisicaoCartao" lazy-init="default"
    autowire="default" dependency-check="default">
    <property name="interfaceCartaoMgt">
          <ref bean="CartaoMgr" />
    </property>
  </bean>

<bean id="CartaoPgtoCartaoCtrl"
    class="lps.bet.variabilidades.cartaoPgtoCartaoCtrl.CartaoPgtoCartaoCtrl" lazy-
    init="default" autowire="default" dependency-check="default">
    <property name="interfaceCartaoMgt">
          <ref bean="CartaoMgr" />
    </property>
    <property name="interfacePagtoCartaoMgt">
          <ref bean="PagamentoCartaoMgr" />
    </property>
  </bean>

<bean id="PagamentoCartaoMgr"
    class="lps.bet.variabilidades.pagamentoCartaoMgr.PagamentoCartaoMgr" lazy-
    init="default" autowire="default" dependency-check="default">
    <property name="pagtoCartaoDAO">
          <ref bean="PagtoCartaoDAO" />
    </property>
    <property name="tipoPassagPagtoCartaoDAO">
          <ref bean="TipoPassagPagtoCartaoDAO" />
    </property>
    </bean>
```

Figura 4.5: *Beans* relacionados ao componente CartaoPgtoCartaoCtrl

A Figura 4.6 mostra o projeto da característica *Pagamento de Cartão* usando aspectos. Esse projeto é correspondente à solução mostrada na Figura 4.4 que usa apenas componentes. A principal diferença é que os componentes básicos ServidorValidadorCtrl e AquisiçãoCartão não precisam requerer a interface ICartaoMgt do componente CartãoPgtoCartãoCtrl. Portanto, esse componente de controle foi excluído, pois o aspecto PagamentoCartãoCtrl implementa todas as operações que estão no componente CartãoPgtoCartãoCtrl. O aspecto PagamentoCartãoCtrl entrecorta todas as operações ICartaoMgt e requer as operações da interface IPagtoCartãoMgt do componente de negócio PagamentoCartãoMgr. Os outros componentes não devem chamar o CartãoPgtoCartãoCtrl e sim o CartaoMgr.

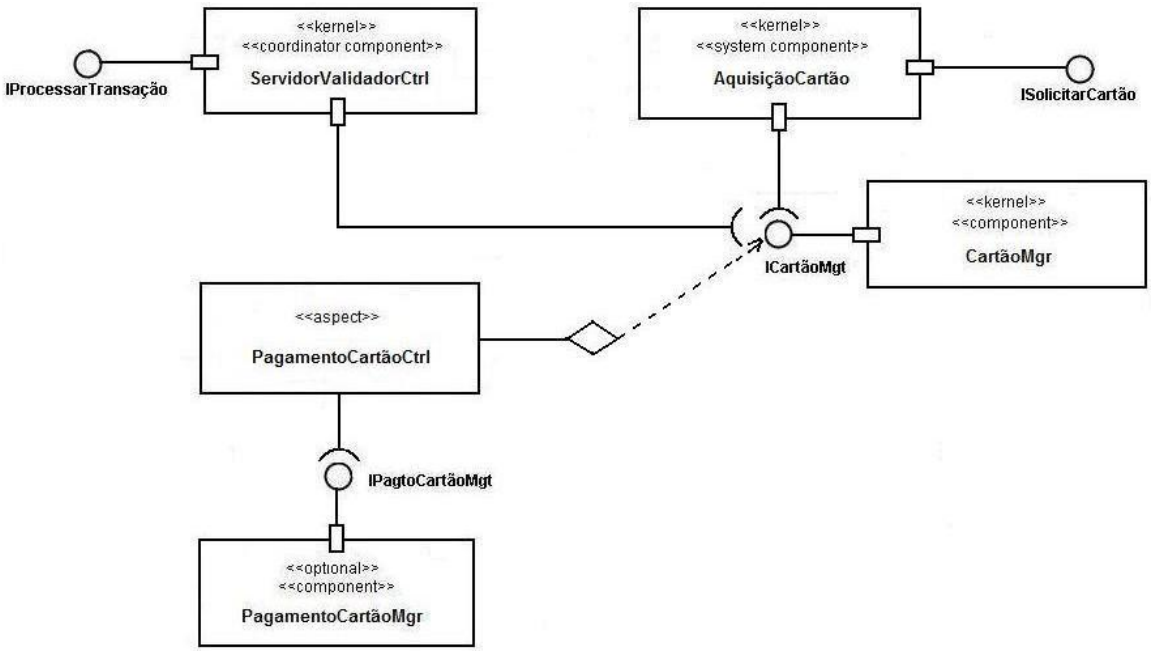

Figura 4.6: Uma solução usando aspectos para adicionar a característica *Pagamento de Cartão* na arquitetura

Como mencionado na seção 3.1.2, os aspectos são usados de modo a interceptar apenas operações das interfaces. Deste modo, um esboço da implementação do aspecto PagamentoCartãoCtrl usando AspectJ é apresentada na Figura 4.7. A interface requerida é a IPagtoCartaoMgt. O ponto de junção entrecorta as chamadas de todos os métodos da interface ICartaoMgt (as operações da interface ICartaoMgt são mostradas na Figura 4.8) e então o adendo do tipo *around* é executado. Se as operações ocorrerem com sucesso, ou seja, as condições para que as execuções dos métodos para ocorrer o pagamento de cartões forem satisfeitas, o adendo retorna o estado ao método da interface entrecortada, sem proceder a execução do método interceptado. Entretanto, caso as operações não ocorram com sucesso o método entrecortado procede com sua execução normal. Essa verificação pode ser feita adicionando variáveis *booleanas* no código (estado = OK ou estado = NOK, por exemplo). Nesse caso foi utilizado o adendo do tipo *around*, porém para outros cenários pode ser necessário utilizar outro tipo de adendo.

Donegan (2008) verificou que para a solução com componentes pode não ser necessário alterar componentes do núcleo para a adição de variabilidades, mas ao menos é preciso alterar a configuração no contexto de aplicação do componente que implementa a interface requerida pelo componente básico.

```
package lps.bet.variabilidades;

import java.util.Collection;
import java.util.List;

import lps.bet.basico.tiposDados.Cartao;
import lps.bet.basico.tiposDados.Pagamento;
import lps.bet.basico.tiposDados.Passageiro;
import lps.bet.basico.tiposDados.TipoPassageiro;
import lps.bet.basico.tiposDados.Viagem;
import lps.bet.interfaces.ICartaoMgt;
import lps.bet.variabilidades.pagamentoCartaoMgr.IPagtoCartaoMgt;
import lps.bet.variabilidades.tiposDados.PagamentoPagtoCartao;
import lps.bet.variabilidades.tiposDados.TipoPassagPagtoCartao;

public aspect PagamentoCartaoCtrl {

	IPagtoCartaoMgt interfacePagtoCartaoMgt;

	public IPagtoCartaoMgt getInterfacePagtoCartaoMgt() {
		return interfacePagtoCartaoMgt;
	}

	public void setInterfacePagtoCartaoMgt(IPagtoCartaoMgt interfacePagtoCartaoMgt) {
		this.interfacePagtoCartaoMgt = interfacePagtoCartaoMgt;
	}

/*Métodos que são alterados pelo controlador CartãoPgtoCartãoCtrl (parametros)*/
	pointcut: call(String lps.bet.interfaces.ICartaoMgt.criarCartao(..));

	String around(): criarPagamentoPgtoCartao(Cartao cartao){

	//String estado=”NOK”;
		...
		...

	//Buscar Tipo de Passageiro do Cartão para verificar depois se tem que pagar
		TipoPassagPagtoCartao tipoPassagPagtoCartao =
interfacePagtoCartaoMgt.buscarTipoPassagPagtoCartao(cartao.getTipoPassageiro().getTipoID());

		//Verificar Pagamento para Aquisicao de Cartao
		if (tipoPassagPagtoCartao.isPagtoAquisicaoCartao()){

			Pagamento pagamento = new Pagamento();
			pagamento.setCartao(cartao);
			pagamento.setDtPgto(cartao.getDtAquisicao());
			pagamento.setValorPgto(tipoPassagPagtoCartao.getValorAquisicao());

			//Criar pagamento normalmente (como já era com carga de cartão)
			int pagtoID = criarPagamento(pagamento);

			//Criar pagamento de aquisicao de cartao
			PagamentoPagtoCartao pgtoCartao = new PagamentoPagtoCartao();
			pgtoCartao.setPagtoID(pagtoID);
			pgtoCartao.setTipoPagto("Aquisição");
			pgtoCartao.setPagamento(pagamento);

			//Criar pagamento de aquisição de cartão com os dados adicionais
			interfacePagtoCartaoMgt.criarPagtoCartao(pgtoCartao);
		}

/*Demais métodos da interface ICartaoMgt que são entrecortados*/

	pointcut: call(String lps.bet.interfaces.ICartaoMgt.buscarCartoes(..));

	……

	pointcut: call(String lps.bet.interfaces.ICartaoMgt.buscarCartao(..));

	……

	pointcut: call(String lps.bet.interfaces.ICartaoMgt.alterarCartao(..));

	……

	pointcut: call(String lps.bet.interfaces.ICartaoMgt.removerCartao(..));

	……
}
```

Figura 4.7: Esboço da implementação do aspecto PagamentoCartãoCtrl

No exemplo da característica de *Pagamento de Cartões*, para a solução com aspectos não seria necessário modificar a implementação e as configurações dos componentes básicos

(ServidorValidadorCtrl e AquisiçãoCartão). A Figura 4.9 mostra como fica a configuração para esse exemplo, usando aspecto em termos do arquivo XML de contexto da aplicação. Ao usar aspectos, essa alteração no contexto da aplicação não é necessária, pois o aspecto entrecorta a interface sem que os componentes básicos tenham conhecimento do mesmo. Essa é uma vantagem que é verificada neste trabalho, confirmando assim, a vantagem do uso de aspectos em uma arquitetura baseada em componentes caixa-preta, para implementação da variabilidade relacionada à característica *Pagamento de Cartão*.

```
1  package lps.bet.interfaces;
2
3  import lps.bet.basico.tiposDados.*;
7
8  public interface ICartaoMgt {
9
10     public boolean validarCartao(int cartaoID);
11     public TipoPassageiro buscarTipoPassagPorCartao(int cartaoID);
12     public float buscarSaldo(int cartaoID);
13     public boolean podeDebitar(int cartaoID, float valor);
14     public Collection buscarTiposPermitidos(Passageiro passageiro, TipoPassageiro tipoSelecionado);
15     public Collection buscarTiposPermitidos(Passageiro passageiro);
16     public Collection buscarTiposPermitidos(Cartao cartao);
17     public void carregarCartao(int cartaoID, float valor);
18     public List buscarViagens();
19     public List buscarViagensPorCartao(int cartaoID);
20     public Viagem buscarUltimaViagem(int cartaoID);
21     public Viagem buscarViagem(int viagemID);
22     public void criarViagem(Viagem viagem);
23     public void alterarViagem(Viagem viagem);
24     public void removerViagem(int viagemID);
25     public List buscarCartoes();
26     public Cartao buscarCartao(int cartaoID);
27     public void criarCartao(Cartao cartao);
28     public void alterarCartao(Cartao cartao);
29     public void removerCartao(int cartaoID);
30     public Pagamento buscarPagamento(int pgtoID);
31     public int criarPagamento(Pagamento pagamento);
32     public void alterarPagamento(Pagamento pagamento);
33     public List buscarPagamentos();
34     public void removerPagamento(int pgtoID);
35     public List buscarTiposPassageiros();
36     public TipoPassageiro buscarTipoPassageiro(int tipoID);
37     public TipoPassageiro buscarTipoPassageiro(String nomeTipo);
38     public void criarTipoPassageiro(TipoPassageiro tipo);
39     public void alterarTipoPassageiro(TipoPassageiro tipo);
40     public void removerTipoPassageiro(int tipoID);
41
42 }
```

Figura 4.8: Operações da Interface ICartaoMgt

Os aspectos podem substituir componentes controladores e de sistema. Eles entrecortam componentes básicos, adicionam lógicas de sistema ou de controle e obtêm informações necessárias de componentes de negócio. Dessa forma, os aspectos atuam como uma espécie de código de ligação (*glue code*) entre o núcleo e as variabilidades da LPS (DONEGAN, 2008).

A solução aqui apresentada para o uso de aspectos não pôde ser efetivamente testada pois não foi possível concluir a preparação do Ambiente BET pelas dificuldades encontradas na configuração de algumas tecnologias/ferramentas necessárias, conforme discussão no final deste capítulo. Portanto, apenas um esboço da implementação do aspecto foi realizado para futuramente efetivar a implementação.

```
<bean id="ControlValidadorServidor"
     class="lps.bet.basico.controlValidadorServidor.ControlValidadorServidor" lazy-
     init="default" autowire="default" dependency-check="default">
     <property name="interfaceCartaoMgt">
           <ref bean="CartaoMgr" />
     </property>
  </bean>

<bean id="AquisicaoCartao"
     class="lps.bet.basico.web.aquisicaoCartao.AquisicaoCartao" lazy-init="default"
     autowire="default" dependency-check="default">
     <property name="interfaceCartaoMgt">
           <ref bean="CartaoMgr" />
     </property>
  </bean>

<bean id="CartaoPagamentoCtrl" class="lps.bet.variabilidades.cartaoPgtoCartaoCtrl"
     factory-method="aspectOf">
     <property name="interfacePagtoCartaoMgt">
           <ref bean="PagamentoCartaoMgr" />
     </property>
  </bean>

<bean id="PagamentoCartaoMgr"
     class="lps.bet.variabilidades.pagamentoCartaoMgr.PagamentoCartaoMgr" lazy-
     init="default" autowire="default" dependency-check="default">
     <property name="pagtoCartaoDAO">
           <ref bean="PagtoCartaoDAO" />
     </property>
     <property name="tipoPassagPagtoCartaoDAO">
           <ref bean="TipoPassagPagtoCartaoDAO" />
     </property>
     </bean>
```

Figura 4.9: *Beans* relacionados ao aspecto PagamentoCartaoCtrl

4.2 Preparação do Ambiente

Com as ferramentas necessárias para a preparação do Ambiente BET, já devidamente instaladas (PostgreSQL, Eclipse, Maven2, Plugin do Maven2 para o Eclipse Tortoise SVN, Plugin do AspectJ para o Eclipse), os passos para a preparação do Ambiente BET, que servem tanto para o Windows XP como para o Vista, indicados por Donegan (2008), são:

1. Acessar o repositório (http://code.google.com/p/bet/) com seu *login* google
   i. Clicar no seu nome em *Project Member*, verifique o seu "*username*"
   ii. Clicar na aba *Settings*, verifique a sua senha

2. Criar um diretório local no seu computador para o projeto, por exemplo -> "*C:\svn*"

3. Clicar no botão da direita e escolha a opção "*SVN Checkout*"
i. URL of repository: https://bet.googlecode.com/svn/

4. Preencher os dados do membro do projeto no repositório google na janela que aparece após o passo 3 (*username* e *password*)

5. A última versão do projeto será baixada do repositório
i. Acessar o diretório no Explorer do Windows e verifique se está com uma marca verde, que significa que o diretório está sincronizado com o repositório

6. Acessar o diretório pelo DOS
i. Entre em "*trunk*" e depois em "*bet*"
ii. Se estiver usando o exemplo, estará em "*C:\svn\trunk\bet*"

7. Colocar o comando maven: mvn eclipse:clean

8. Colocar o comando maven: mvn eclipse:eclipse
i. Todas as dependências existentes no pom.xml que não forem encontradas no repositório local do maven serão baixadas pela web

9. Importar o projeto no eclipse:
i. *File -> Import -> Existing Projects into Workspace -> Select root directory*
ii. Selecionar o diretório local em que o projeto se encontra, até a parte *bet*
iii. No exemplo seria "*C:\svn\trunk\bet*"

10. Habilitar o Maven no projeto:
i. Clicar com o botão da direita sobre o projeto "*bet*"
ii. Opção: *Maven2 -> Enable*
iii. As dependências declaradas no pom.xml serão reconhecidas pelo Eclipse

11. Criar o Banco de Dados BET no PostgreSQL
i. Usar o script de backup do banco de dados disponibilizado no repositório Google do projeto para criar o banco de dados bet
ii. O arquivo filter.properties encontra-se configurado com *username = root* e *password = root*, manter essas informações consistentes

12. Para rodar o servidor, deve-se ir ao diretório do projeto e usar o comando maven "mvn jetty:run"
i. O diretório no exemplo seria o "*C:\svn\trunk\bet*"
ii. "*Starting scanner at internal of 10 seconds*" (rodando com sucesso)
iii. Para rodar o cliente do ônibus, deve-se executar a classe "ValidadorGUI" do componente GUI.
iv. Para rodar a parte web, usar http://localhost:8080/bet/controlGerencia.html

13. A LPS-BET ainda se encontra em desenvolvimento, portanto, não deve ser realizado o *commit* sem autorização. Aconselha-se usar a versão atualizada e trabalhar com ela

localmente em um projeto diferente, para que não haja problemas no desenvolvimento da LPS-BET em si.

Obs.: Por questões de licença, certas bibliotecas da *Sun* não podem ser disponibilizadas em servidores de controle de versão, portanto, devem ser baixadas diretamente no site.

4.3 Tutorial para preparação do ambiente BET

Além de seguir os passos apresentados na seção 4.2, foi criado um tutorial de instalação de alguns recursos de software baseando-se no sistema operacional Windows Vista.

4.3.1 PostgreSQL

O PostgreSQL (PostgreSQL, 2008) é um dos SGBDs (Sistema Gerenciador de Bancos de Dados) de código aberto mais avançados, contando com recursos como: consultas complexas, chaves estrangeiras, integridade transacional, controle de concorrência multi-versão, suporte ao modelo híbrido objeto-relacional, gatilhos, visões, procedimentos armazenados em várias linguagens, dentre outros.

Sem algumas mudanças na segurança do Windows Vista a instalação do PostgreSQL 8.0 não funciona, pois é gerado um erro na criação do usuário quando o banco executa o serviço no sistema operacional. Para resolver esse problema e fazer a instalação com sucesso é preciso seguir os seguintes passos:

1. Clicar em *Start -> Control Panel -> User Accounts -> Turn User Account Control on or off*;
2. Desmarcar a opção de segurança do usuário do sistema operacional;
3. Reinicializar o Windows Vista para ser aplicada essa configuração;
4. Instalar o PostgreSQL normalmente (http://www.postgresql.org/download/);
5. Voltar à configuração do usuário para o padrão;
6. Reinicializar o sistema operacional para reaplicar as alterações;

4.3.2 Eclipse

O Eclipse (ECLIPSE, 2008) é um IDE (*Integrated Development Environment* - Ambiente integrado para desenvolvimento de software) de código aberto para a construção de programas de computador. Possui como características marcantes o uso da SWT (*Standard*

Widget Toolkit) e não do Swing como biblioteca gráfica, a forte orientação ao desenvolvimento baseado em *plug-ins* e o amplo suporte ao desenvolvedor com centenas de *plug-ins* que procuram atender as diferentes necessidades de diferentes programadores.

Para a instalação do Eclipse, basta acessar o site http://www.eclipse.org/downloads/, fazer o *download* do arquivo e instalar normalmente.

4.3.3 Maven 2

Segundo (WIKIPÉDIA, 2008), Apache Maven, ou simplesmente Maven, é uma ferramenta para gerenciamento e automação de projetos em Java. Ela é similar à ferramenta Ant, mas possui um modelo de configuração mais simples, baseado no formato XML. Maven é um projeto da *Apache Software Foundation*.

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

Uma característica chave do Maven é o fato de ser construído para trabalhar em rede. O núcleo da ferramenta pode baixar *plug-ins* de um repositório (o mesmo repositório utilizado pelos outros projetos Java do Apache e outras organizações). O Maven disponibiliza suporte nativo para a recuperação de arquivos desse repositório, e para a inclusão dos artefatos resultantes no final do processo. Um *cachê* de artefatos atua como ponto de sincronização dos artefatos de um projeto local.

Para a instalação do Maven 2 (Apache, 2008), basta acessar o site: http://maven.apache.org/download.html, fazer o *download* do arquivo e seguir os passos abaixo:

1. Descompactar a distribuição do arquivo, ou seja, apache-maven-2.0.9-bin.zip para o diretório no qual se deseja instalar maven 2.0.9. Estas instruções assumem que foi escolhido *C:\Program Files\Apache Software Foundation*. O subdiretório apache-maven-2.0.9 será criado a partir do arquivo.

2. Adicionar a variável de ambiente M2_HOME a partir de *Start -> Control Panel -> System -> Advanced system settings -> Environment Variable*; e, em seguida, acrescentar a variável M2_HOME, nas variáveis do usuário com o valor *C:\Program Files\Apache Software Foundation\apache-maven-2.0.9*. Certificar-se de omitir quaisquer aspas ao redor do caminho, mesmo que contenha espaços. Em versões do Maven anteriores à 2.0.9, deve-se também verificar se o M2_HOME não tem um '\' como último caractere, se tiver, remova-o.

3. Na mesma janela, adicionar a variável de ambiente M2 às variáveis do usuário com o valor %M2_HOME%\bin.

4. Opcional: Na mesma janela, adicionar a variável de ambiente MAVEN_OPTS às variáveis do usuário especificar as propriedades da JVM, por exemplo, o valor-Xms256m-Xmx512m. Essa variável de ambiente pode ser usada para fornecer opções extras para Maven.

5. Na mesma janela, atualizar/criar o *Path* variável de ambiente às variáveis do usuário e colocar o valor %M2% para adicionar Maven disponível na linha de comando.

6. Na mesma janela, certificar-se que JAVA_HOME existe em suas variáveis de usuário ou nas variáveis do sistema, e está definido para a localização do seu JDK, por exemplo, C:\Program Files\Java\jdk1.5.0_02 e que %JAVA_HOME%\bin está no seu *Path* das variáveis de ambiente.

7. Abrir um novo prompt (*WinKey* + R), em seguida digitar cmd e executar *mvn --version* para verificar se ele foi instalado corretamente.

4.3.4 Plug-in do Maven para o Eclipse

Para instalar o *plug-in* no Eclipse, basta seguir os seguintes passos:

1. Selecionar o menu "*Help*"
2. Selecionar a opção "*Software Updates*"
3. Selecionar a opção "*Find and Install*"

4. Uma nova tela aparecerá. Escolher a opção "*Search for New Features do Install*" e clique no botão "*Next*"
5. Na nova tela que aparecerá, clicar no botão "*New Remote Site*". Na seqüência, digitar o nome "*Maven 2 Plugin*" no campo "Name" e a URL http://m2eclipse.codehaus.org/update/ no campo "URL". Clique no botão "*OK*".
6. Veja se o quadro ao lado do nome "*Maven 2 Eclipse*" na lista está marcado, se não estiver, marque-o.
7. Clicar no botão "*Finish*".

Aparecerá uma tela pedindo confirmação sobre a instalação do *plug-in*, confirme a instalação e o Eclipse começará a baixar os arquivos necessários automaticamente. Quando isso for concluído, o computador deve ser reiniciado a fim de concluir a instalação do plug-in do Maven 2 no Eclipse.

O *plug-in* permite executar o Maven 2 de dentro do Eclipse e adiciona automaticamente qualquer dependência do POM no *classpath* do projeto do Eclipse. Para transformar um projeto comum em um projeto gerenciado pelo Maven, basta clicar com o botão direito do mouse na pasta do projeto, procurar a opção "Maven 2" e clicar na opção "*Enable*". Se o projeto já tiver um "*pom.xml*" o Maven simplesmente usa este POM como configuração para o projeto, se ainda não existir um POM, ele vai mostrar um formulário para a indicação das informações iniciais do projeto.

O *plug-in* também permite adicionar dependências sem ter que alterar diretamente o arquivo de configuração. Basta clicar com o botão direito do mouse na pasta do projeto, ir à opção "Maven" e selecionar a opção "*Add Dependency*". Aparecerá um formulário onde se pode digitar o nome da dependência e será possível buscar por ela nos repositórios configurados no seu Maven. A parte de busca do *plug-in* ainda tem alguns problemas, pois mesmo que a dependência não seja encontrada, ela pode estar disponível no *plug-in*, mas provavelmente não pode ser indexada por algum motivo. Então, em alguns momentos, ainda pode ser necessário alterar diretamente o POM para adicionar uma dependência ao seu projeto.

Adicionar o *plug-in* do Maven ao Eclipse vai facilitar ainda mais a gerência do *classpath*, pois não vai mais ser preciso ficar colocando todos os JAR dos quais ele depende dentro do próprio projeto, pois o *classpath* vai ser gerenciado pelo Maven e a configuração do POM do projeto.

4.3.5 Tortoise SVN

TortoiseSVN é um cliente *open-source* do sistema de controle de versões *Subversion*. Isto é, TortoiseSVN gera pastas e diretórios ao longo do tempo. As pastas são armazenadas num repositório central. Esse repositório funciona de modo muito parecido a um comum servidor de pastas, exceto que controla e registra todas as modificações feitas nas suas pastas ou diretórios. Isto permite recuperar versões antigas das pastas e permite examinar o histórico das modificações dessas mesmas pastas ou diretórios, sabendo quando e como foram modificados e, ainda, quem os modificou.

Para começar, basta acessar o site http://tortoisesvn.net/downloads, fazer o *download* do arquivo e instalar o TortoiseSVN. Sua instalação é bem simples e não tem nenhum segredo. Deve-se criar, no *Windows Explorer,* um diretório vazio para ser seu repositório. Clicar com o botão direito, no menu escolha TortoiseSVN e depois *Create Repository Here*. Na seqüência, clicar com o botão direito sobre seu repositório e ir a Tortoise e *Repo Browser*. Seu repositório será mostrado e então será possível adicionar seus arquivos. É recomendado seguir apenas um padrão para separar os projetos e diretórios.

4.3.6 Plug-in do AspectJ para o Eclipse

O AspectJ é uma extensão da linguagem Java que permite editar e combinar aspectos a programas Java. Para instalar os *plug-ins* basta acessar o site http://www.eclipse.org/ajdt/downloads/ e fazer o *download* do AspectJ para o Eclipse. O arquivo baixado estará compactado (formato “ZIP”). Ao fazer a extração dos arquivos, serão criadas as pastas: “features” e “plugins”. O conteúdo destas pastas deverá ser copiado para as pastas de respectivos nomes para dentro da pasta onde o Eclipse foi instalado.

4.4 Dificuldades Encontradas

Houve algumas dificuldades no decorrer do estudo sobre aspectos, pois não é uma abordagem de compreensão intuitiva. Também houve dificuldade para a instanciação de um domínio para geração de uma aplicação utilizando o Captor, pois não são oferecidos muitos materiais/tutoriais sobre esse gerador.

No decorrer do preparo do Ambiente BET, foram surgindo alguns problemas e com o auxílio da MSc. Paula Marques Donegan via MSN e *e-mails*, esses problemas foram sendo superados, mas ainda existem erros a serem corrigidos para a preparação do Ambiente BET ser efetivamente concluída. As dificuldades foram concentradas na incompatibilidade de versões e das diferentes configurações das ferramentas necessárias para a preparação do Ambiente BET propriamente dito utilizando-se o sistema operacional *Windows Vista*, dado que os passos para instalação e execução do projeto BET foram definidos utilizando-se o sistema operacional *Windows XP*.

Conseqüentemente, os testes do uso de aspectos para a implementação de variabilidades relacionadas às variabilidades de uso de cartões não foram efetivamente realizados, uma vez que não foi possível concluir a preparação do Ambiente BET pelas dificuldades encontradas na configuração de algumas tecnologias/ferramentas necessárias. Ficando essa atividade para trabalhos futuros.

Capítulo 5
Conclusão

Neste trabalho foram apresentados conceitos sobre a POA, Linha de Produto de Software, o processo de desenvolvimento da LPS-BET e uma breve descrição sobre o Captor.

Foi feita uma explanação a respeito da LPS-BET, uma LPS praticamente completa e não trivial, que poderá ser usada para apoiar outras pesquisas. Foram abordadas decisões sobre o projeto da LPS-BET com componentes caixa-preta e foi mostrado o uso de aspectos nas características de *Integração* da LPS-BET.

Foi projetado um aspecto para a variabilidade relacionada ao uso de cartões e foi criado um tutorial para instalação dos recursos necessários à preparação do Ambiente BET no sistema operacional Windows Vista.

As contribuições do trabalho à área de Ciência e Tecnologia são em relação ao uso de aspectos. Os aspectos de um sistema podem ser alterados, inseridos ou removidos em tempo de compilação, e comumente reusados. Como estão no mesmo bloco de código, a manutenção é muito mais fácil e a complexidade do sistema diminui, facilitando o entendimento do mesmo. Porém, por se tratar de um tema novo, muita pesquisa ainda pode ser realizada para encontrar novas vantagens e desvantagens em relação ao seu uso, sendo interessante investigar esses estudos a fundo.

Infelizmente não foi possível concluir a implementação do aspecto pela falta de tempo para superar as dificuldades encontradas no decorrer do trabalho, no entanto as atividades desenvolvidas até aqui ficam de apoio ao conhecimento para desenvolvimento de trabalhos futuros.

Como trabalhos futuros, espera-se que o aspecto projetado aqui seja efetivamente implementado, bem como seja desenvolvida uma versão da LPS-BET com todas as variabilidades implementadas por aspectos. Deste modo, poderá ser analisado em quais situações foi válido usar aspectos para implementar as variabilidades.

Referências Bibliográficas

APACHE, **The Apache Maven Project**. Acessado em agosto, 2008. Disponível em: http://maven.apache.org/

ASPECTJ TEAM, **The AspectJ Programming Guide**. Acessado em agosto, 2008. Disponível em: http://www.eclipse.org/aspectj/doc/released/progguide/

CLEMENTS, P.; NORTHROP, L. **Software Product Lines: Practices and Patterns**, Addison-Wesley, 2002.

CLEMENTS, P. C., JONES, L. G., MCGREGOR, J. D., NORTHROP, L. M., **Getting There From Here: A Roadmap for Software Product Line Adoption.** Communications of the ACM. Vol. 49, Nº 12. December, 2006.

DONEGAN, P. M., MASIERO, P. C., **Design Issues in a Component-based Software Product Line.** In: Simpósio Brasileiro de Componentes, Arquiteturas e Reuso de Software, 2007, Campinas. Anais do Simpósio Brasileiro de Componentes, Arquiteturas e Reuso de Software. Porto Alegre : Sociedade Brasileira de Computação, 2007. v. 1. p. 3-16.

DONEGAN, P. M. **Geração de Famílias de Produtos de Software com Arquitetura Baseada em Componentes.** 2008. Dissertação (Mestrado em Ciência da Computação e Matemática Computacional). ICMC-USP/São Carlos. São Carlos, SP.

ECLIPSE, **The Eclipse Foundation.** Acessado em agosto, 2008. Disponível em: http://www.eclipse.org/

FIGUEIREDO, E., CACHO, N., SANT'ANNA, C., MONTEIRO, M., KULESZA, U., GARCIA, A., SOARES, S., FERRARI, F., KHAN, S., FILHO, F., DANTAS, F. **Evolving Software Product Lines with Aspects: An Empirical Study on Design Stability.** International Conference on Software Engineering, a ser publicado, 2008.

GIMENES, I. M. S.; TRAVASSOS, G. H. **O Enfoque de Linha de Produto para Desenvolvimento de Software.** In: Ingrid Jansch Porto. (Org.). XXI Jornada de Atualização em Informática (JAI) - Livro Texto. 1 ed. Porto Alegre: Sociedade Brasileira de Computação, 2002, v. 2, p. 01-31.

GOETTEN J., WINCK D. **AspectJ – Programação Orientada a Aspectos com Java.** São Paulo: Novatec Editora, 2006.

GOMAA, H. **Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures.** Addison-Wesley Boston, 736 p., 2004.

GOOGLE, **Google Code.** Acessado em agosto, 2008. Disponível em: http://code.google.com/

HIBERNATE, **Hibernate - Relational Persistence for Java and .NET.** Acessado em setembro, 2008. Disponível em: http://www.hibernate.org/

JUDE, **Jude UML Modeling Tool.** Acessado em outubro, 2008. Disponível em: http://jude.change-vision.com/jude-web/index.html/

KICZALES, G.; LAMPING, J.; MENDHEKAR, A.; MAEDA, C.; LOPES, C. V.; LOINGTIER, J. M.; IRWIN, J.. **Aspect-Oriented Programming**. European Conference on Object-Oriented Programming (ECOOP), LNCS (1241), Springer-Verlag, Finland., June 1997.

OLIVEIRA JUNIOR, E. A.; GIMENES, I. M. S.; HUZITA, Elisa Hatsue Moriya ; MALDONADO, José Carlos . **A Variability Management Process for Software Product Lines.** In: The 15th Annual International Conference of Computer Science and Software Engineering (CASCON), 2005, Ontario - Toronto. Proceedings of CASCON 2005. Toronto, Ontario - Canada : IBM, 2005. v. 1. p. 30-44.

POSTGRESQL, **PostgreSQL Global Development Group**. Acessado em agosto, 2008. Disponível em: http://www.postgresql.org/

SANT'ANNA, Cláudio Nogueira. **Manutenibilidade e Reusabilidade de Software Orientado a Aspectos: Um Framework de Avaliação**. 2004. Dissertação de Mestrado – Programa de Pós – Graduação em Informática da PUC- Rio.

SHIMABUKURO JR, E. K. . **Um gerador de aplicações configurável.** Dissertação de Mestrado, ICMC-USP, 2006.

SHIMABUKURO JR, E. K. ; BRAGA, R. T. V. ; MASIERO, P. C. . **Captor: Um Gerador de Aplicações Configurável.** In: XIII Sessão de Ferramentas do SBES, 2006, Florianópolis. Caderno de Ferramentas do SBES. Porto Alegre: Sociedade Brasileira de Computação, 2006. p. 121-126.

SOARES, S. e BORBA, P. **AspectJ - Programação Orientada a Aspectos em Java**. Tutorial do VI Simpósio Brasileiro de Linguagens de Programação (SBLP'2002). PUC-Rio, Rio de Janeiro, Brasil, Junho 2002. Disponível em: http://www.cin.ufpe.br/~scbs/artigos/AspectJ_SBLP2002.pdf. Acessado em abril, 2008.

SPRING, **Spring Framework.** Acessado em setembro, 2008. Disponível em: http://www.springframework.org/

TIGRIS, **Tortoise SVN.** Acessado em agosto, 2008. Disponível em: http://tortoisesvn.tigris.org/

WIKIPÉDIA. **Desenvolvido pela Wikimedia Foundation**. Apresenta conteúdo enciclopédico. Acessado em agosto, 2008. Disponível em: http://pt.wikipedia.org/w/index.php?title=Apache_Maven&oldid=10734230/

Maringá, 27 de novembro de 2008.

À Coordenação de Trabalho de Graduação
Departamento de Informática
Centro de Tecnologia
Universidade Estadual de Maringá

Encaminhamos a monografia intitulada “Uma investigação quanto ao uso de aspectos para implementação de variabilidades de Linha de Produto de Software” para avaliação desta coordenação.

_____
Aluno: Guilherme Antonialli Meilsmith

_____
Orientador: Profa. MSc.Thelma Elita Colanzi Lopes