

Avaliação da adequação do uso de aspectos na implementação de variabilidades de linha de produto de software

Rodrigo de C. Brito¹, Thelma E. Colanzi¹

¹Departamento de Informática – Universidade Estadual de Maringá (UEM)
Maringá – PR – Brasil

rodrigocb@sanepar.com.br, thelma@din.uem.br

Abstract. *Recently, the development of software product line (SPL) is one of the most used to enable the software reuse. In this regard, several studies have been conducted in order to enhance this activity. Some studies have shown that the implementation of variability of SPL using aspect-oriented programming (AOP) can be beneficial in various situations. Therefore, this work aimed at implementing some variabilities of a SPL for managing electronic ticketing transport (LPS-BET) using AOP. Besides the development of variabilities, we conducted a comparative analysis based on metrics, which showed that in the context of LPS-BET, the use of AOP does not exceed the benefits offered by component-based development.*

Resumo. *O desenvolvimento de Linhas de Produtos de Software (LPS) é uma das técnicas mais utilizadas atualmente para viabilizar o reuso de software. Neste sentido, vários trabalhos vêm sendo realizados com o intuito de aprimorar essa atividade. Algumas pesquisas têm mostrado que a implementação de variabilidades de LPS usando programação orientada a aspectos (POA) pode ser benéfica em várias situações. Sendo assim, este projeto teve por objetivo implementar algumas variabilidades de uma LPS para gestão de bilhetes eletrônicos de transporte (LPS-BET) usando POA. Além do desenvolvimento das variabilidades, foi realizada uma análise comparativa baseada em métricas, o que permitiu concluir que, no contexto da LPS-BET, a utilização de POA não supera as vantagens oferecidas pelo desenvolvimento baseado em componentes.*

1. Introdução

Existem atualmente diversas técnicas de reuso de software, utilizando *frameworks*, componentes, geradores de aplicações, padrões de projeto, Linha de Produto de Software (LPS), dentre outros. Essas técnicas podem ser combinadas para alcançar benefícios como ganho de produtividade, melhoria da qualidade do produto, redução do *time-to-market*. Assim, o reuso sistemático de software consiste de uma mudança da abordagem de construção de sistemas únicos para o desenvolvimento de famílias de produtos.

O objetivo da abordagem de LPS é a construção sistemática de software baseada em uma família de produtos. Nesta abordagem, um conjunto de características similares, entre os vários produtos de um dado domínio, permite a definição de uma estrutura comum de itens que será compartilhada entre os produtos da LPS. Além disso, as

características variáveis, denominadas de variabilidades, são resolvidas possibilitando a diferenciação entre os produtos a serem gerados [van der Linden et al 2005].

A programação orientada a objetos (POO), não fornece abstrações necessárias para a modularização de interesses transversais - interesses cuja implementação encontra-se espalhada por vários módulos de um sistema - causando assim um espalhamento de código pelo sistema, repetindo a lógica desses interesses por diversos módulos. Por isso, os interesses transversais são um tanto quanto difíceis de implementar e manter.

Visando solucionar os problemas relacionados ao espalhamento e entrelaçamento de código, Kiczales et al [1997] criaram, nos laboratórios de pesquisas da Xerox, o conceito de Programação Orientada a Aspectos (POA). A POA permite encapsular em um único módulo um interesse transversal, melhorando a modularidade do software.

Segundo Donegan [2008, p.2], diversos trabalhos enfatizam a dificuldade de elicitar, representar e implementar variabilidades no contexto de uma LPS. Ela afirma ainda que diversos pesquisadores têm investigado essa questão e têm proposto outras soluções baseadas POA e em programação orientada a características [Mezini e Ostermann 2004; Apel e Batory 2006; Heo e Choi 2006; Lee et al. 2006; Anastasopoulos e Muthig 2004].

Desta forma, o objetivo deste trabalho é implementar algumas variabilidades de uma LPS para gestão de bilhetes eletrônicos de transporte (LPS-BET) [Donegan 2008] utilizando aspectos; e comparar os resultados obtidos em relação à implementação anterior da LPS-BET, a qual foi realizada utilizando componentes do tipo caixa-preta.

As variabilidades cujas implementações foram desenvolvidas usando POA são *Acesso Adicional e Integração Terminal*. Após o desenvolvimento, foi realizada uma análise pautada em métricas com o intuito de comparar os resultados da solução baseada em POO e da solução baseada em POA. Assim, os resultados obtidos do desenvolvimento de variabilidades da LPS-BET com aspectos podem fornecer maiores indícios sobre quando é melhor utilizar aspectos na evolução de LPS, contribuindo para a melhoria do processo de desenvolvimento de LPS.

Este artigo está organizado da seguinte forma: na Seção 2 apresenta-se uma revisão bibliográfica sobre LPS, POA e uma visão geral de como a LPS-BET foi desenvolvida; na Seção 3 é abordada a solução elaborada para desenvolver as variabilidades *Acesso Adicional e Integração Terminal* utilizando POA; na Seção 4 são analisados os resultados alcançados e na Seção 5 apresentam-se as considerações finais do trabalho.

2. Revisão Bibliográfica

Nas próximas subseções serão apresentados conceitos sobre Linhas de Produtos de Software, Engenharia de Domínio e de Aplicação, Programação Orientada a Aspectos e a LPS-BET desenvolvida por Donegan [2008].

2.1 Linha de Produto de Software

Segundo Parnas [1979 apud Donegan 2008], uma coleção de sistemas que compartilham características comuns é chamada de “família de sistemas”. Hoje elas também são chamadas de Linha de Produto de Software. Uma LPS é constituída por um conjunto de aplicações similares de um domínio em particular, que podem ser

desenvolvidas a partir de uma arquitetura genérica comum, a arquitetura da LPS, e um conjunto de componentes que povoam a arquitetura [van der Linden et al 2005].

A organização de uma LPS consiste em três atividades: desenvolvimento do núcleo de artefatos, desenvolvimento do produto e gerenciamento da LPS [SEI 2009]. 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. Nas próximas subseções são apresentadas as atividades de engenharia de domínio e de aplicação.

2.1.1 Engenharia de Domínio

Nesta atividade é realizada a análise de domínio, que corresponde ao desenvolvimento de todas as características comuns aos produtos da LPS e constitui o núcleo de artefatos. O núcleo de artefatos pode incluir requisitos, modelo de domínio, arquitetura de software, engenharia de performance, documentação, código, planos de teste, casos de teste, cronogramas, planos de trabalho, entre outros itens [SEI 2009].

Na análise do domínio da LPS são consideradas as funcionalidades que: são comuns a todas as aplicações do domínio (núcleo da LPS); aquelas que são opcionais, existentes apenas para alguns produtos da LPS; e as alternativas, escolhidas a partir de um conjunto de possibilidades. Em uma LPS, as características constituem uma unidade lógica de comportamento que corresponde a um conjunto de requisitos funcionais e de qualidade [van Gorp e Bosch 2001]. Eles afirmam que existe uma relação n:m entre características e requisitos. Isto significa que um determinado requisito pode ser aplicado a várias características e uma característica em particular pode abranger mais de um requisito. As características podem ser de um dos seguintes tipos definidos pelo método PLUS (*Product Line UML-Based Software Engineering*) [Gomaa 2004]:

- **Obrigatórias:** características presentes em todos os produtos da LPS.
- **Opcionais:** características presentes em alguns produtos da LPS.
- **Alternativas:** conjunto de características em que se deve fazer uma única escolha dentre as possíveis. Pode-se dizer que as opções são mutuamente exclusivas.

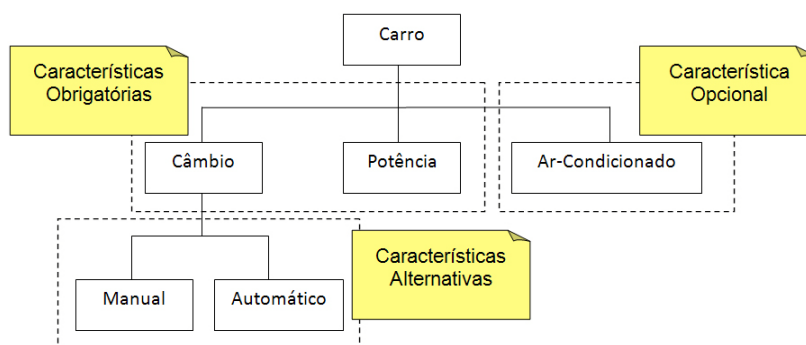


Figura 1. Exemplo de Modelo de Características [Adaptado de Kang et al 1990]

A Figura 1 exemplifica, para o domínio de carros, os tipos de características descritas anteriormente, onde as características obrigatórias que farão parte do núcleo são “câmbio” e “potência”, as características “manual” e “automático” são alternativas e

“Ar-condicionado” é opcional. Em relação às características alternativas pode-se imaginar como especializações, em que apenas uma delas pode ser implementada.

É no núcleo que todas as características do tipo obrigatórias estão implementadas, ou seja, todas aquelas que farão parte de todos os produtos da LPS. Ainda, para formar uma LPS é necessário representar as variabilidades que podem ocorrer em cada artefato que faz parte desta. Variabilidades são diferenças encontradas entre os produtos podendo ser reveladas e distribuídas entre os seus artefatos [van der Linden et al 2005]. Elas são descritas em termos de pontos de variação e variantes. Um ponto de variação é um lugar específico em um artefato da LPS ao qual uma decisão de projeto é conectada. Cada ponto de variação está associado a um conjunto de variantes que corresponde às alternativas de projeto para uma variabilidade [Heumans e Trigaux 2003 apud Oliveira Junior et al 2005].

A atividade de engenharia de domínio pode ser realizada seguindo as etapas de Concepção, Elaboração, Construção e Transição correspondentes ao ciclo iterativo incremental Donegan [2008] tomou como base o método PLUS [Gomaa 2004] para o processo de desenvolvimento da LPS-BET. Além disso, a engenharia de domínio pode ser incrementada utilizando-se de incrementos horizontais ou verticais. Utilizando os incrementos horizontais, o subgrupo de características de um produto é implementado, produzindo no final um produto completo. Utilizando incrementos verticais, todas as variabilidades de um subgrupo de características são implementadas, de forma completa, porém não produzem necessariamente um produto completo. É na etapa de transição que são elaborados os gabaritos a serem aplicados na instanciação dos produtos.

2.1.2 Engenharia de Aplicação

É a atividade em que cada produto é construído. É feita uma análise da aplicação-referência a ser produzida, e por meio dos resultados, são selecionadas as características elaboradas na engenharia de domínio.

A montagem de uma aplicação-referência pode ser realizada de duas formas: (i) **manual**: na qual o gabarito criado é seguido ao longo da engenharia de domínio para se gerar a aplicação; e (ii) **automatizada**, na qual o mesmo gabarito utilizado na montagem do tipo manual pode ser processado por geradores automáticos a fim de gerar a aplicação ao final do processo.

A montagem da aplicação-referência, utilizando o método automatizado, pode ser feito por meio de geradores de aplicação. Segundo Donegan [2008], os geradores de aplicação são ferramentas de software que transformam informação de alto nível em implementação de baixo nível. O problema ou a tarefa a ser realizada por um programa consiste na informação de alto nível e é descrita por meio de uma especificação, que é usada pelo gerador de aplicação para automaticamente produzir um programa.

Geradores de aplicação configuráveis (GAC) são aqueles que podem ser adaptados para gerar aplicações de diversos domínios, como, por exemplo, o Captor [Shimabukuro 2006]. A Figura 2 demonstra a visão de um GAC, que consegue gerar os artefatos da aplicação, tais como: código, documentação e teste, por meio dos arquivos de configuração no formato XML (*Extensible Markup Language*) e dos gabaritos.

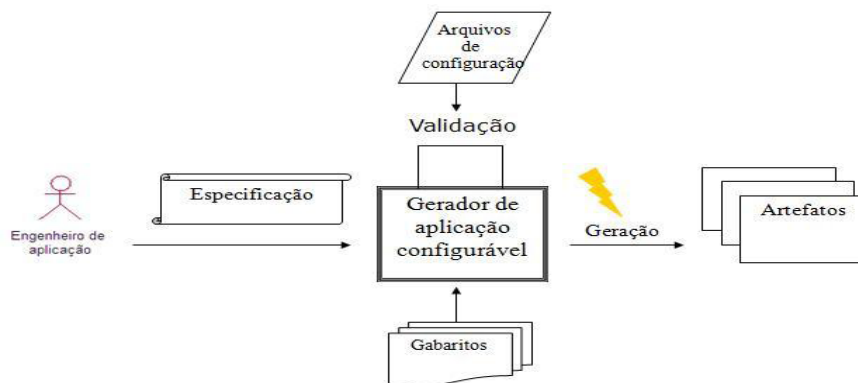


Figura 2. Gerador de Aplicação Configurável [SHIMABUKURO 2006]

2.2 Programação Orientada a Aspectos

As linguagens de POO possuem limitações, no projeto e na implementação de interesses que normalmente afetam diversas classes e/ou módulos, não conseguindo manter a separação de interesses. Um interesse representa uma característica relevante de uma aplicação, que pode ser definido como uma parte isolada de um domínio com o intuito de entender melhor cada parte isoladamente. Um interesse pode ser um requisito funcional ou não-funcional [Meilsmith 2008]. Quando um mesmo interesse está presente em vários módulos do sistema, ele é chamado de interesse transversal.

Se o desenvolvimento de um software for realizado sem se preocupar com a separação de interesses, isso pode resultar em código entrelaçado e espalhado. O código entrelaçado (*code tangling*) acontece quando a implementação de um módulo em um software interage simultaneamente com vários interesses, tais como *logging*, autenticação, *multi-threaded safety*, validações, entre outras. O código espalhado (*code scattering*) acontece quando a implementação de um interesse encontra-se em múltiplos módulos [Kiczales et al 1997]. Uma vez que interesses transversais, por definição, são espalhados por vários módulos, logo sua implementação também se espalha por esses módulos [Donegan 2008].

Diante desse problema Kiczales et al. [1997] propuseram a abordagem de POA para separar e encapsular esses interesses transversais e, assim, eliminar o espalhamento e entrelaçamento de código em módulos chamados de aspectos.

Aspectos são compostos pelos elementos descritos a seguir:

- 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 et. al 1997].

- 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 definí-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.

- **Adendos:** são trechos 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 POO), 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. Há três tipos de adendos: (i) *before* que é executado antes do ponto de junção; (ii) *after*: que é executado depois do ponto de junção; e, (iii) *around* que é executado durante a execução do ponto de junção.

- **Declarações *inter-types*:** são declarações que adicionam novas funcionalidades à aplicação, por exemplo, adicionando um método ou atributo na classe.

- **Aspectos:** encapsulam pontos de junção, pontos de atuação, adendos e declarações *inter-types* em uma unidade modular de implementação. São definidos de maneira semelhante às classes, enquanto essas encapsulam o código que se encaixa dentro delas, *aspectos* encapsulam o código que é ortogonal, transversal a essas classes.

A Figura 3 exemplifica cada um desses elementos. O aspecto *FaultHandler* (linha 01) consiste de uma declaração *inter-type* que introduz um atributo na classe *Server* (linha 03).

Na linha 13 é apresentado o ponto de atuação de nome *services* que irá capturar todas as chamadas de todos os métodos de todas as classes, é apresentado um adendo do tipo *before* (linhas 15-17), ou seja antes de executar os comandos contidos no ponto de atuação *services(s)*, serão executados os comandos contidos dentro do adendo (linha 16). Nas linhas de 19 a 22 é apresentado um adendo do tipo *after*, ou seja, os comandos contidos dentro do ponto de atuação *services(s)* serão executados depois do contido no adendo (linhas 20 e 21).



Figura 3. Elementos de um Aspecto [Adaptado de AspectJ Team 2009]

O aspecto adiciona uma funcionalidade ao código-base interceptando o fluxo de execução. No código-base não é preciso haver menção explícita ao aspecto. Um processo de combinação (*weaving*) é executado pelo compilador do aspecto (*aspect weaver*), para que o código-base e os aspectos sejam combinados, gerando ao final um produto completo, com todos os interesses implementados. Portanto, ao utilizar a POA, os sistemas ficam mais legíveis, fáceis de entender, implementar, integrar, reusar, personalizar, evoluir e manter [Kiczales et al 1997].

2.3 Linha de Produto de Software para Bilhetes Eletrônicos de Transporte (LPS-BET)

LPS-BET [Donegan 2008] trata-se de uma LPS de gestão de bilhetes eletrônicos de transporte, que facilita o transporte urbano com o uso de um cartão eletrônico para pagar passagens e oferecer várias outras funcionalidades para passageiros e companhias de ônibus.

A LPS-BET foi projetada e desenvolvida visando a geração de três produtos, 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), chamados de aplicações-referência. A partir de uma análise detalhada das características semelhantes e das variabilidades A diferença entre do domínio, foram projetados e implementados componentes para a construção do núcleo e das variabilidades da LPS-BET. Componentes são unidades independentes, que encapsulam funcionalidades. Cada um deles oferece serviços, por meio de interfaces bem definidas para o meio externo [Brown 2000]. Existem dois tipos de componentes: componente caixa-branca e componente caixa-preta. A diferença entre eles é que no primeiro pode-se ter acesso ao seu código, enquanto no segundo somente tem-se acesso à interface. Para desenvolver a LPS-BET, Donegan [2008] utilizou componentes caixa-preta.

O desenvolvimento da LPS-BET ocorreu em quatro incrementos horizontais [Donegan 2008], sendo eles:

- 1º) Desenvolvimento dos casos de uso do núcleo da LPS-BET;
- 2º) Reuso do núcleo e desenvolvimento dos casos de uso de Fortaleza;
- 3º) Reuso do núcleo e de alguns casos de uso de Fortaleza e desenvolvimento dos casos de uso de Campo Grande;
- 4º) Reuso do núcleo, de alguns casos de uso de Fortaleza e de alguns casos de uso de Campo Grande, além do desenvolvimento dos casos de uso de São Carlos.

Por fim, o gerador de aplicações Captor [Shimabukuro 2006] foi utilizado na engenharia de aplicação para automatizar a geração dos produtos da LPS.

Na Tabela 1 são mostradas as características opcionais e alternativas, que devem ser incorporadas ao núcleo da LPS para obter cada um dos três produtos, além da características obrigatórias. As características apresentadas na coluna Tipo da Tabela 1 identificadas pela letra *N* representam características obrigatórias; aquelas identificadas pela letra *O* indicam características opcionais; e as características identificadas pela letra *A* são alternativas.

Tabela 1. Características dos produtos da LPS-BET [adaptada de Donegan 2008]

Característica	Tipo	Fortaleza	Campo Grande	São Carlos
Acesso Básico	N	X	X	X
Autenticação de Funcionario	N	X	X	X
Tipo Passageiro	N	X	X	X
Política de Desconto	N	X	X	X
Passageiros	N	X	X	X
Acesso Adicional	O		X	X
Autenticação Passageiro	O			X
Forma de Integração	O		X	
- Terminal			X	
- Integração		X	X	X
* Tempo	O		X	X
* Linha de Integração	O		X	
* Número de Viagens de Integração	O			
Pagamento de Cartão	O	X		
Restrição de Cartões				
- Número de Cartões	A		X	X
- Combinação de Cartões	A			
Empresas Usuárias	O	X	X	
Limite de Passagens	O			X

A implementação da LPS-BET foi feita inicialmente utilizando-se o conceito de arquitetura baseada em componentes. Decidiu-se utilizar POA para a implementação de um requisito não-funcional, como é demonstrado na Subseção 2.3.1. Donegan [2008] fez ainda a implementação de um requisito funcional usando POA, para isso, os aspectos foram implementados visando interceptar as interfaces dos componentes. Este último caso é apresentado na Subseção 2.3.2.

2.3.1 Implementação de Requisito não-Funcional da LPS-BET usando POA

O requisito não-funcional de autenticação foi projetado e implementado usando aspectos, caso contrário ele estaria espalhado em vários componentes (*Carga Cartão*, *Aquisição Cartão*, *GerênciaCtrl*), e cada um deles seria responsável por autenticar e autorizar os usuários no sistema, levando à uma dificuldade de manutenção, devido ao espalhamento do código. Pode-se observar na Figura 4 a presença do aspecto de autenticação, identificado com o estereótipo <<aspect>>, que está presente nos três produtos, e por isso faz parte do núcleo.

Para que essa decisão de projeto possa ser explicada, é preciso analisar a documentação fornecida no trabalho de Donegan [2008]. Primeiramente, é possível verificar que os componentes com terminação Mgr são considerados de negócio [Chessman e Daniels 2001 apud Donegan 2008, p. 61], e aqueles com terminação Ctrl, de controle [Gomma 2004 apud Donegan 2008, p. 61].

Sobre a relação entre aspectos e componentes é salientado que: “os componentes de negócio são extraídos do refinamento do modelo conceitual da mesma forma como

são extraídos no projeto sem o uso de aspectos. Os aspectos substituem 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 a variabilidades da LPS” [Donegan 2008; p. 98].

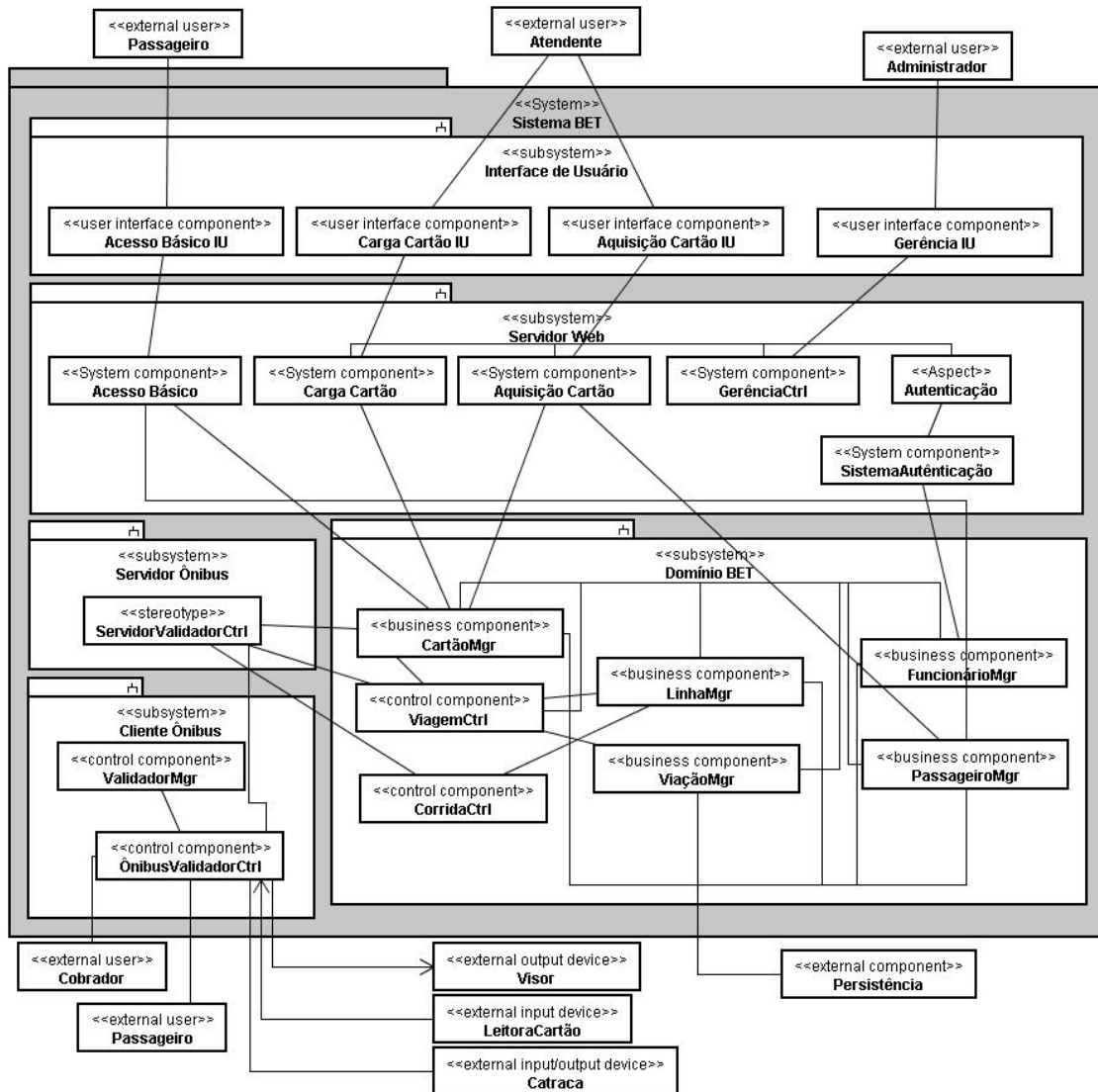


Figura 4. Arquitetura de Componentes do Núcleo da LPS-BET [Donegan 2008]

O aspecto de autenticação foi dividido em duas partes, uma para autenticar o usuário (*Autenticação*) e outra para autorizar as páginas da web que o usuário pode navegar (*Autorização*). Ambos possuem o mesmo ponto de junção configurado para entrecortar todos os métodos `handleRequestInternal` [Spring 2008], porém o aspecto de autenticação precede o de autorização. A Figura 5 apresenta os atributos, adendos e operações dos aspectos *autenticação* e *autorização*.

O aspecto autenticação requer algumas operações da interface *ISistemaAutenticacao* do componente *SistemaAutenticacao*: *atualizarSessao*, *estaAutenticado* e *estaExpirado* e seu adendo é do tipo *before*, pois a autenticação deve ocorrer antes do usuário acessar a página web.

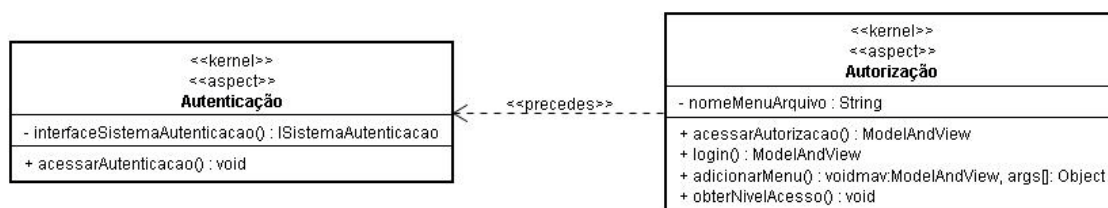


Figura 5. Especificação dos Aspectos Autenticação e Autorização [Donegan 2008]

Os aspectos autenticação e autorização entrecortam as interfaces dos componentes *GerênciaCtrl*, *Aquisição Cartão* e *Carga Cartão*, conforme ilustrado na Figura 6. Esses são os três componentes nos quais o interesse de autenticação é necessário.

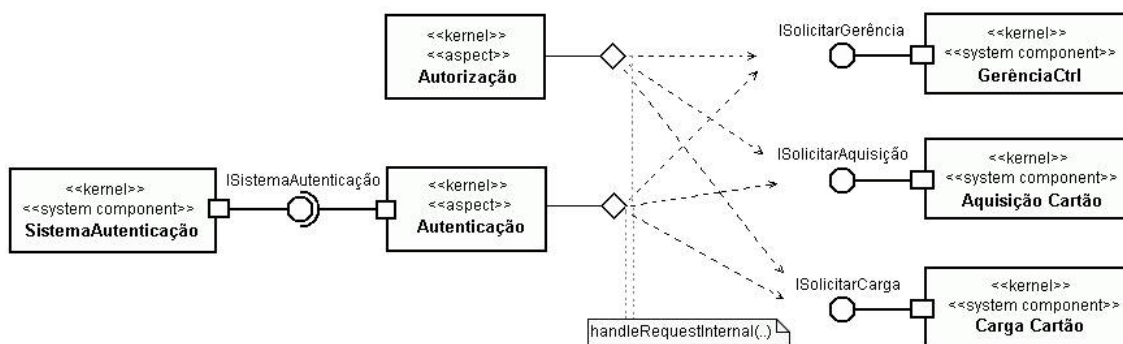


Figura 6. Arquitetura dos aspectos Autenticação e Autorização e dos componentes entrecortados [Donegan 2008]

Com a implementação desse aspecto, não houve retrabalho ao longo do desenvolvimento dos componentes da LPS-BET, pois este foi configurado para entrecortar todos os métodos *handleRequestInternal* (método usado especificamente por controladores do MVC (*Model-View-Controller*) [Spring 2009]) dos componentes que fazem parte do sub-sistema Servidor WEB, com exceção do *Acesso Básico*.

2.3.2. Implementação de Requisito Funcional da LPS-BET usando POA

A implementação inicial de variabilidades da LPS-BET foi elaborada utilizando apenas componentes. Porém, Donegan [2008] ilustrou o uso de POA para implementar requisitos não-funcionais por meio do grupo de características de Integração (*Tempo*, *Linha Integrada* e *Número de Viagens de Integração*). Esta solução foi projetada de tal forma que o aspecto interceptasse as interfaces dos componentes relacionados à característica de integração, por serem do tipo caixa-preta.

O uso de linhas integradas pré-definidas, e/ou o uso de ônibus dentro de um intervalo de tempo, marcam a característica integração, podendo ainda haver um número máximo de integração dentro de um período de tempo.

Todas as características do grupo têm um comportamento semelhante, requerem alterações do mesmo componente (*ViagemCtrl*), devendo ser capazes de processar a integração pela requisição das interfaces *IRegistrarViagem* e *IRegistrarArrecadação*. Essas características diferem em relação ao componente de negócio que requerem,

solicitando interfaces diferentes, e, por isso, terão implementações diferentes para fazer a verificação da integração.

Um aspecto abstrato foi usado para generalizar a similaridade entre os três tipos de integração, cada variante representa um aspecto e implementa o aspecto abstrato (*IntegracaoCtrl*), herdando o ponto de junção (*validarIntegracao*), um adendo e um método (*processarIntegracao*), além de duas interfaces comuns requeridas (Figura 7).

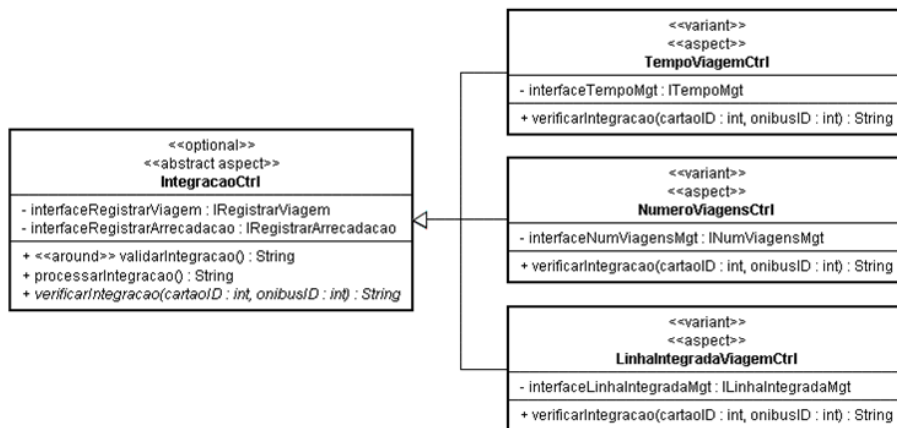


Figura 7. Aspecto abstrato e aspectos concretos para representar a característica Integração [Donegan 2008]

A implementação do aspecto abstrato *IntegraçãoCtrl* usando AspectJ [Kiczales et al 2001] é apresentada na Figura 8. 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 8. Implementação do Aspecto IntegraçãoCtrl [Donegan 2008]

O aspecto *IntegracaoCtrl* entrecorta a interface *IProcessarViagem* (linha 6) e é estendido pelo aspecto *TempoViagemCtrl* que requer as operações da interface *ITempoMgt* do componente de negócio *TempoMgr*.

A implementação do aspecto *TempoViagemCtrl* é exibida na Figura 9. Ele estende o aspecto abstrato *IntegracaoCtrl* (linha 01) e, portanto, pode utilizar o método *processarIntegracao* (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 *IntegracaoCtrl*, 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 = interfaceCartaoMgt.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 9. Implementação do Aspecto TempoViagemCtrl [Donegan 2008]

A solução para adicionar a característica *Tempo* usando POA na arquitetura da LPS-BET, conforme ilustrado na Figura 10, não requer a alteração dos componentes, bastando configurar o contexto de aplicação do componente que implementa a interface requerida pelo componente básico. Assim, o componente básico não terá conhecimento da existência das variabilidades, pois serão utilizadas interfaces com nomes análogos.

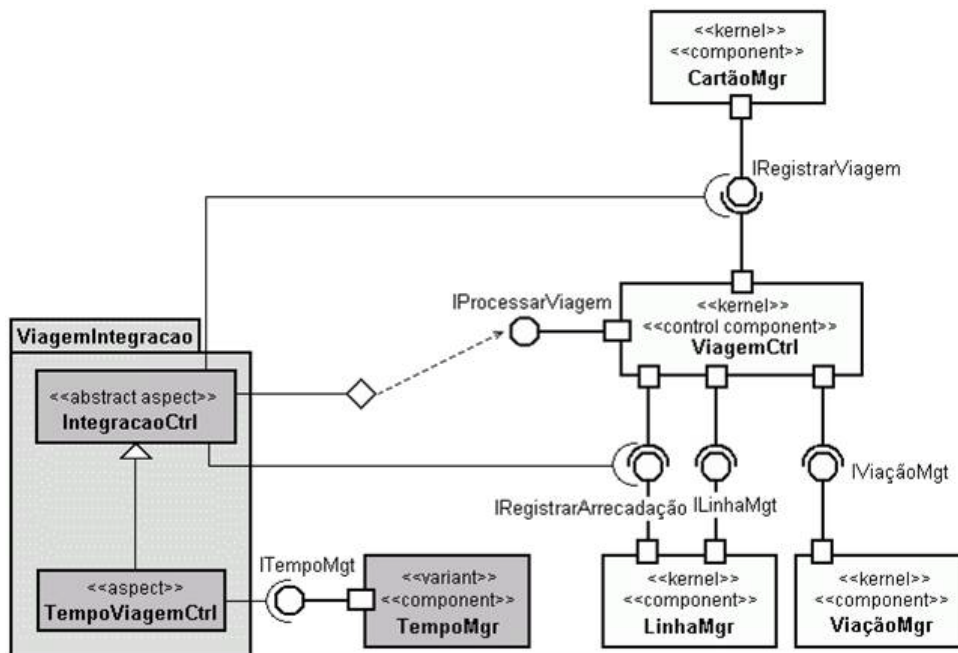


Figura 10. Uma Solução usando Aspectos para Adicionar a Característica Tempo na Arquitetura da LPS-BET[Donegan 2008].

3. Implementação das Variabilidades da LPS-BET

Nesta seção são apresentadas as implementações referentes às variabilidades *Acesso Adicional* e *Terminal Integrado* do grupo de característica *Integração* usando POA. A implementação da característica *Acesso Adicional* como um componente caixa preta havia sido desenvolvida por Donegan [2008]. Já a característica *Terminal Integrado* não havia sido desenvolvida. Para fins de comparação, primeiramente, foi desenvolvido o componente caixa-preta referente a esta variabilidade, de acordo com o modelo de Donegan [2008] e, em seguida, foi desenvolvida uma versão equivalente utilizando aspectos.

A implementação da LPS-BET realizada por Donegan [2008] foi feita utilizando: (i) o ambiente Eclipse [Eclipse 2008] para o desenvolvimento de programas em Java, (ii) o Hibernate [Hibernate 2008] para persistência dos dados e (iii) o banco de dados PostGreSQL [Postgresql 2008]. Portanto, na realização deste projeto os mesmos recursos tecnológicos foram utilizados. Para a implementação dos aspectos que representam as variabilidades foi usada a linguagem para desenvolvimento de aspectos AspectJ [Aspectj 2009].

3.1 Acesso Adicional

Esta característica opcional aplica-se aos produtos de Campo Grande e São Carlos. Refere-se a um *acesso adicional* ao sistema, acesso do tipo Web, em que o passageiro tem a possibilidade de consultar viagens e imprimir extratos.

Como se pode observar na Figura 12, que demonstra parcialmente a arquitetura de componentes da variabilidade *Acesso Adicional*, o componente *AcessoAdicional* requer uma interface denominada *ICartaoMgt* que é implementada pelo componente *CartaoMgr*.

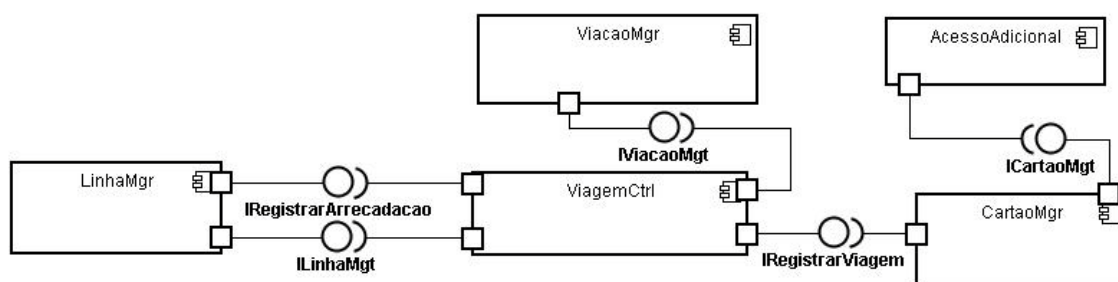


Figura 11. Arquitetura de Componentes para a Variabilidade *AcessoAdicional*

Utilizando-se como base a estrutura de componentes implementada, foi desenvolvida a versão desta variabilidade usando POA, que pode ser observada na Figura 12. O aspecto *AcessoAdicionalAspect* intercepta as operações da interface *ICartaoMgt*, utilizando-se do adendo do tipo *around*, pois ele direciona o fluxo de execução, que pode ser notado nas linhas 14 e 29 da Figura 13. A solução adotada neste trabalho foi elaborada com base no trabalho de Donegan [2008] que diz:

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; p.99].

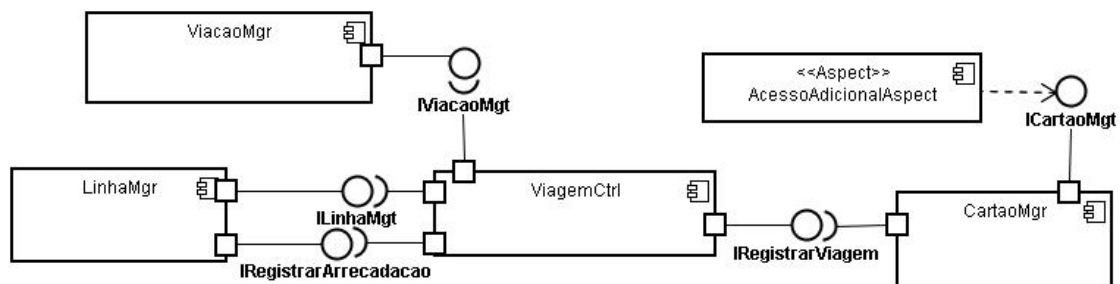


Figura 12. Uma Solução usando Aspectos para Adicionar a Característica Acesso Adicional na Arquitetura da LPS-BET

Desta forma, foi implementada uma solução baseada no trabalho de Donegan [2008], em que as operações da variabilidade Acesso Adicional fornecidas por meio da interface *ICartaoMgt* sejam entrecortadas pelo aspecto *AcessoAdicionalAspect*, fazendo com que os componentes da LPS-BET não consigam utilizá-las diretamente.

```

1 package lps.bet.basico.cartaoMgr;
2 import java.util.List;
3
4 public aspect AcessoAdicionalAspect {
5     public ICartaoMgt interfaceCartaoMgt;
6     pointcut buscarViagensPorCartao(int cartaoID):
7         call (List ICartaoMgt.buscarViagensPorCartao(int))
8         && args(cartaoID)
9         && !within(CartaoPgtoCartaoCtrl)
10        && !within(AcessoAdicionalAspect);
11    @SuppressWarnings("unchecked")
12    List around(int cartaoID): buscarViagensPorCartao(cartaoID){
13        try {
14            return interfaceCartaoMgt.buscarViagensPorCartao(cartaoID);
15        } catch (Exception e) {
16            e.printStackTrace();
17            List retorno = null;
18            return retorno;
19        }
20    }
21
22    pointcut buscarViagensTerminalPorCartao(int cartaoID):
23        call (List ICartaoMgt.buscarViagensTerminalPorCartao(int))
24        && args(cartaoID)
25        && !within(CartaoPgtoCartaoCtrl)
26        && !within(AcessoAdicionalAspect);
27    @SuppressWarnings("unchecked")
28    List around(int cartaoID): buscarViagensTerminalPorCartao(cartaoID){
29        try {
30            return interfaceCartaoMgt.buscarViagensTerminalPorCartao(cartaoID);
31        } catch (Exception e) {
32            e.printStackTrace();
33            List retorno = null;
34            return retorno;
35        }
36    }
37
38    public ICartaoMgt getInterfaceCartaoMgt() {
39        return interfaceCartaoMgt;
40    }
41    public void setInterfaceCartaoMgt(ICartaoMgt interfaceCartaoMgt) {
42        this.interfaceCartaoMgt = interfaceCartaoMgt;
43    }
44 }
  
```

Figura 13: Código fonte do Aspecto *AcessoAdicionalAspect*

Na Figura 16 o ponto de atuação *buscarViagensPorCartao* é apresentado nas linhas 8 a 12, criado com a denominação *buscarViagensPorCartao*. Ainda na Figura 13 observa-se nas linhas 14 a 22 a criação do adendo, que possui as funcionalidades tratadas pelo aspecto. A solução adotada na implementação do aspecto da variabilidade *Acesso Adicional*, leva em conta a solução citada no trabalho de Donegan [2008], isto é, o aspecto substitui o controlador que implementa as funcionalidades, e essas são tratadas nos adendos do aspecto. As operações que foram interceptadas pelo aspecto *AcessoAdicionalAspect* foram as operações para buscar as viagens ocorridas nas linhas da BET e buscar as viagens que utilizaram terminais.

Na Figura 14 é apresentado o resultado da implementação da variabilidade *Acesso Adicional*, usando aspectos, em que são listadas as viagens realizadas por cartão, listando as linhas que o passageiro utilizou o cartão, bem como os terminais em que ele utilizou o cartão para ter acesso.

BET Gestão
Campo Grande

Consultas Viação Cartão Funcionário Passageiro Linha Terminal Relatórios Sair

Lista de Viagens - Cartão 17

ID	Data Viagem	Hora Entrada	Nome Linha
<input type="checkbox"/> 1	06/11/2007	15:05:52	linha 1
<input type="checkbox"/> 3	06/11/2007	15:17:32	linha 1
<input type="checkbox"/> 4	06/11/2007	15:31:42	linha 1

ID	Data Viagem	Hora Entrada	Nome Terminal
<input type="checkbox"/> 1	12/11/2009	11:53:47	Lagoa Redonda
<input type="checkbox"/> 2	12/11/2009	14:13:41	Novo Terminal
<input type="checkbox"/> 7	13/11/2009	11:51:53	Lagoa Redonda
<input type="checkbox"/> 8	13/11/2009	11:58:19	Papicu

Cancelar

© Copyright 2008 Donegan Paula

Figura 14. Tela de Listagem de Viagens do Passageiro

3.2 Terminal Integrado

Essa característica opcional faz parte dos produtos de Fortaleza e de Campo Grande. A integração do tipo terminal permite que o passageiro embarque e desembarque de terminais definidos na linha sem a necessidade de pagar uma nova passagem. A entrada nos terminais é feita através de validadores na entrada dos terminais, catracas parecidas com as instaladas nos ônibus.

Como mencionado anteriormente, não havia implementação em componentes para esta característica. Então, foi implementada uma versão em componentes, com base na arquitetura em componentes ilustrada na Figura 15. Como se pode observar, foi criado um componente *TerminalIntegradoGUIMgr*, que implementa as operações exigidas pela interface *ITerminalIntegradoGUIMgt*.

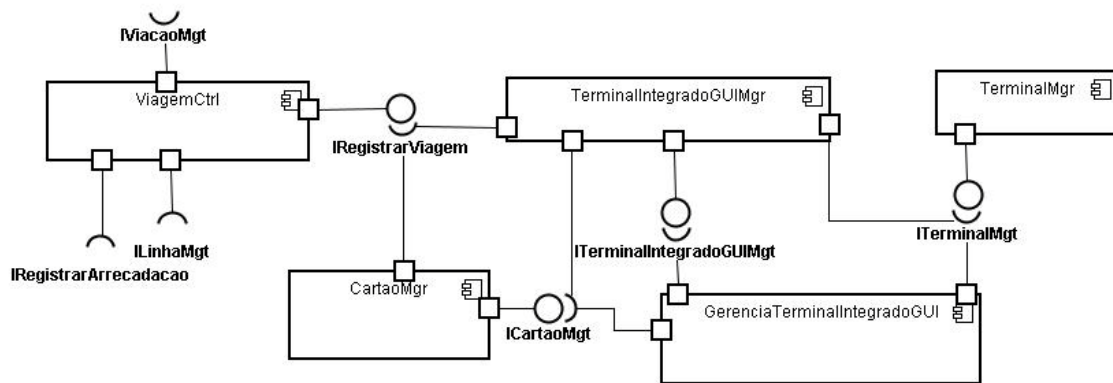


Figura 15. Arquitetura de Componentes para a Variabilidade *Terminal*

Na Figura 14 é ilustrado o diagrama de classes da referida variabilidade. Para o projeto dessa variabilidade as operações obrigatórias que devem ser implementadas foram definidas na interface. Essas operações foram implementadas pelo componente concreto *TerminalIntegradoGUIMgr*, que as fornece para a LPS-BET.

Pode-se observar na Figura 16 que foi criada a operação *tratarCartao*, que tem a função de localizar o cartão do passageiro, verificar o tipo de passageiro, verificar a tarifa a ser debitada, registrando a viagem, liberando ou não a catraca para o embarque.

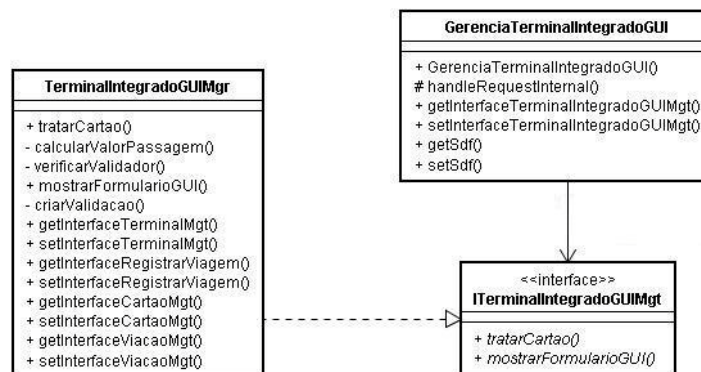


Figura 16. Diagrama de Classes - Integração Terminal

Para essa variabilidade foi implementado ainda o componente controlador *GerenciaTerminalIntegradoGUI*.

Considerando a implementação baseada em componentes, foi implementada uma versão usando POA para essa variabilidade. O seu diagrama de componentes é ilustrado na Figura 17, na qual é possível notar que o aspecto *GerenciaTerminalIntegradoAspect* intercepta as operações da interface *ITerminalIntegradoGUIMgt*. O código do aspecto é apresentado na Figura 18 a seguir.

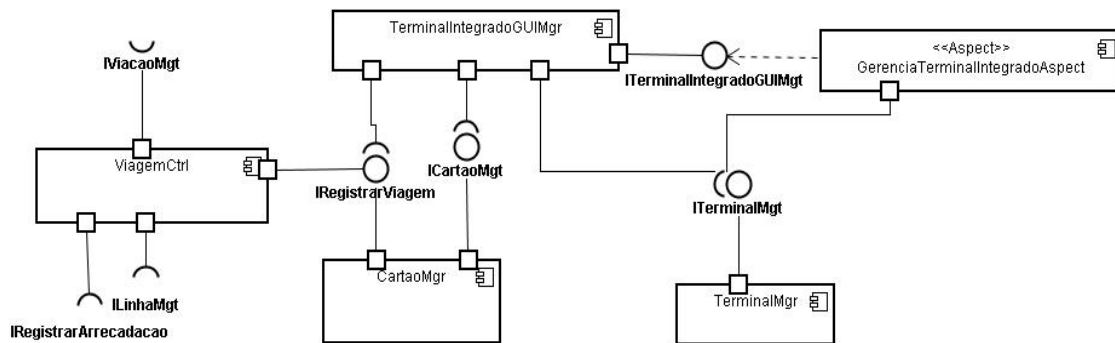


Figura 17. Uma Solução usando Aspectos para Adicionar a Característica Terminal Integrado na Arquitetura da LPS-BET

```

1 package lps.bet.variabilidades;
2 import javax.servlet.http.HttpServletRequest;
3 import lps.bet.variabilidades.terminalIntegradoMgr.ITerminalIntegradoGUIMgt;
4 import org.springframework.web.servlet.ModelAndView;
5
6 public aspect GerenciaTerminalIntegradoAspect {
7     ITerminalIntegradoGUIMgt interfaceITerminalIntegradoGUIMgt;
8
9     pointcut tratarCartao(HttpServletRequest request, String dados):
10         //Aqui o aspecto intercepta a interface ITerminalIntegradoGUIMgt
11         call (ModelAndView ITerminalIntegradoGUIMgt.tratarCartao( HttpServletRequest , String ))
12         && args(request, dados) && !within(GerenciaTerminalIntegradoAspect) ;
13     ModelAndView around(HttpServletRequest request, String dados):tratarCartao(request, dados){
14         //implementação da variabilidade
15         return interfaceITerminalIntegradoGUIMgt.tratarCartao(request, dados);
16     }
17     pointcut mostrarForm(String erro, HttpServletRequest request ):
18         //Aqui o aspecto intercepta a interface ITerminalIntegradoGUIMgt
19         call (ModelAndView ITerminalIntegradoGUIMgt.mostrarFormularioGUI( String, HttpServletRequest))
20         && args( erro, request) && !within(GerenciaTerminalIntegradoAspect) ;
21     ModelAndView around( String erro,HttpServletRequest request):mostrarForm( erro,request){
22         //implementação da variabilidade
23         return interfaceITerminalIntegradoGUIMgt.mostrarFormularioGUI(erro,request);
24     }
25     public ITerminalIntegradoGUIMgt getInterfaceITerminalIntegradoGUIMgt() {
26         return interfaceITerminalIntegradoGUIMgt;Métodos Necessários para configuração do Spring
27     }
28     public void setInterfaceITerminalIntegradoGUIMgt(
29         ITerminalIntegradoGUIMgt interfaceITerminalIntegradoGUIMgt) {
30         this.interfaceITerminalIntegradoGUIMgt = interfaceITerminalIntegradoGUIMgt;
31     }
32 }

```

Figura 18. Código do aspecto Terminal Integrado

4. Análise dos resultados

Para analisar os resultados obtidos neste trabalho, foram adotadas as métricas para componentes utilizadas por Donegan [2008]. As métricas referentes a aspectos foram definidas neste trabalho como segue:

- Número de interfaces interceptadas pelo aspecto: somatório das interfaces que são interceptadas pelo aspecto em questão;
- Número de adendos por aspecto: somatório do número de adendos que o aspecto em questão contém;

- Número de operações de cada interface interceptada: somatório das operações interceptadas pelo aspecto em questão;
- Número de classes que referenciam a interface substituída por aspecto: somatório das classes que referenciam a interface substituída pelo aspecto em questão;
- ALOC (linhas de código dos adendos): somatório das linhas de código dos adendos de um determinado aspecto;
- LOC do aspecto: somatório das linhas de código de um aspecto.

Na Tabela 2 são apresentadas as métricas obtidas da implementação da variabilidade *Acesso Adicional* e as métricas referentes à *Integração Terminal* são apresentadas na Tabela 3.

Tabela 2. Métricas relacionadas à Variabilidade Acesso Adicional

Métrica		Valor
Componente	Número de interfaces envolvidas com a variabilidade	1
	Número de componentes envolvidos com a variabilidade	4
	Número de aspectos envolvidos com a variabilidade	1
	LOC do controlador	54
	MLOC (Linhas de código dos métodos) buscarViagensPorCartao(cartaoID)	2
	MLOC (Linhas de código dos métodos) buscarViagensTerminalPorCartao(cartaoID)	2
Aspecto	Número de interfaces interceptadas pelo aspecto	1
	Número de adendos por aspecto	2
	Número de operações de cada interface interceptada	2
	Número de classes que referenciam a interface substituída por aspecto	4
	ALOC (linhas de código dos adendos)	6
	LOC do aspecto	58

Tabela 3. Métricas relacionadas à Variabilidade Terminal Integrado

Métrica		Valor
Componente	Número de interfaces envolvidas com a variabilidade	1
	Número de componentes envolvidos com a variabilidade	5
	Número de aspectos envolvidos com a variabilidade	1
	LOC do controlador	40
	MLOC (Linhas de código dos métodos) tratarCartao(HttpServletRequest request, String dados)	65
	MLOC (Linhas de código dos métodos) mostrarFormularioGUI(String erro,HttpServletRequest request)	32
Aspecto	Número de interfaces interceptadas pelo aspecto	1
	Número de adendos por aspecto	2
	Número de operações de cada interface interceptada	2
	Número de classes que referenciam a interface substituída por aspecto	3
	ALOC (linhas de código dos adendos)	7
	LOC do aspecto	36

Pode-se observar por meio das métricas apresentadas, que a substituição proposta não trouxe vantagens visíveis. Como citado na Subseção 3.2, os aspectos foram implementados com base na sugestão de Donegan [2008], isto é, implementar os aspectos e interceptar as operações das interfaces requeridas utilizando os componentes de negócio existentes. Analisando as métricas relacionadas à variabilidade Acesso Adicional (Tabela 2), observa-se que não há muita diferença na implementação utilizando-se aspectos. Por exemplo, entre o LOC do controlador e o LOC do aspecto observa-se uma diferença mínima de 4 (quatro) LOC de linhas de código. Observando ainda a soma do MLOC dos métodos e o ALOC dos adendos, os números resultantes são bem próximos, apontado, assim, pouca diferença de implementação. As métricas relacionadas à variabilidade Terminal Integrado, refletem uma situação muito similar. Outro ponto a ser observado é o número de adendos por aspecto e o número de operações das interfaces interceptadas, que são iguais. Dessa forma, a utilização de aspectos não trouxe indícios para a LPS-BET na substituição das variabilidades propostas. Como nenhuma alteração na forma como a variabilidade funciona foi necessária, o aspecto apenas substituiu as funções dos componentes de negócio. Os componentes de negócio originais, por sua vez, não são mais utilizados por outras partes da arquitetura, embora ainda continuem conectados. Assim confirmou-se a afirmação de Donegan [2008 p. 99] de que *“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”*. Isto é, constatou-se que não é vantajosa a substituição dos componentes de negócio da arquitetura da LPS-BET por aspectos.

5. Conclusão

Este trabalho teve por objetivo implementar uma solução baseada em aspectos para as variabilidades *Acesso Adicional* e *Integração Terminal* e analisar os resultados alcançados em relação às implementações das mesmas variabilidades usando componentes. Constatou-se que, no contexto da LPS-BET, a solução baseada em componentes é mais facilmente mantida que a solução usando POA. Isto porque os componentes de negócio originais continuam conectados a outros componentes, embora as operações sejam executadas a partir do aspecto, diferentemente da arquitetura projetada para a LPS-BET, podendo prejudicar a manutenção futura da LPS.

Pela experiência adquirida com o estudo dos conceitos de LPS, POA e da arquitetura da LPS-BET, pode-se afirmar que a implementação de novas variabilidades utilizando aspectos pode ser benéfica, desde que os aspectos não alterem a configuração da estrutura dos componentes da arquitetura ou ainda quando o seu uso tiver sido planejado desde o início do projeto da LPS.

Assim, o objetivo do trabalho foi alcançado, além da experiência adquirida tanto no desenvolvimento e manutenção de uma arquitetura de LPS baseada em componentes e aspectos, quanto na configuração de um ambiente com os recursos tecnológicos necessários ao desenvolvimento deste trabalho.

Foram encontradas algumas dificuldades em relação ao projeto existente como, por exemplo, na documentação constava uma nomenclatura e no código fonte outro nome. Também foram encontradas dificuldades em relação à configuração do ambiente, sendo

necessário estudar sobre as ferramentas utilizadas para alcançar a configuração adequada.

Como trabalho futuro, pretende-se fazer novos estudos de casos para produzir evidências das situações nas quais o uso de POA para implementar variabilidades de LPS é vantajoso. Outro trabalho possível é fazer uma evolução da LPS-BET para adicionar novas variabilidades usando aspectos e usando componentes a fim de comparar resultados alcançados. Por fim, pesquisadores da USP/ São Carlos que fazem parte deste projeto, utilizarão uma versão do Captor implementada com POA para gerar os produtos com as variabilidades implementadas usando aspectos.

Referências

- Anastasopoulos, M.; Muthig, D. (2004) An Evaluation of Aspect-Oriented Programming as a Product Line Implementation Technology. In: Proceedings of the 8th International Conference Software Reuse - ICSR 2004, Springer, Madrid, Spain, p. 141–156.
- Apel, S.; Batory, D. (2006) When to Use Features and Aspects? A Case Study. In: Proceedings of the 5th International Conference on Generative Programming and Component Engineering, ACM Press New York, NY, USA, p. 59–68.
- Aspectj Team (2009) The AspectJ Programming Guide. Acessado em maio, 2009. Disponível em: <http://www.eclipse.org/aspectj/doc/released/progguide/>
- Brown, Alan W. (2000) Large-scale, component-based development. Upper-Saddle River : Prentice-Hall.
- Donegan, P. M. (2008) Geração de Famílias de Produtos de Software com Arquitetura Baseada em Componentes. 160p. Dissertação (Mestrado em Ciência da Computação e Matemática Computacional). ICMC-USP/São Carlos. São Carlos, SP.
- Eclipse IDE (2008). Eclipse Project. Disponível para acesso na URL: <http://www.eclipse.org/>, em junho de 2008.
- Gomaa, H. (2004) Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures. Addison-Wesley Boston, 736 p.
- Heo, S.; Choi, E. (2006) Representation of Variability in Software Product Line Using Aspect-Oriented Programming. In: Proceedings of the Fourth International Conference on Software Engineering Research, Management and Applications - SERA 2006, IEEE Computer Society, Washington, USA, p. 66–73.
- Kiczales, G.; Hilsdale, E.; Hugunin, J.; Kersten, M.; Palm, J.; Griswold, W. (2001) An Overview of AspectJ. In: Proceedings of the 15th European Conference on Object-Oriented Programming, Lecture Notes in Computer Science, Springer-Verlag, London, UK, p. 327–353.
- Lee, K.; Kang, K.; Kim, M.; Park, S. (2006) Combining Feature-Oriented Analysis and Aspect-Oriented Programming for Product Line Asset Development. In: Proceedings of 10th International Software Product Line Conference - SPLC 2006, Baltimore, USA, p. 103–112.

- Kiczales, G.; Lamping, J.; Mendhekar, A.; Maeda, C.; Lopes, C. V.; Loingtier, J. M.; Irwin, J (1997). Aspect-Oriented Programming. European Conference on Object-Oriented Programming (ECOOP), LNCS (1241), Springer-Verlag, Finland., June.
- Kiczales, G.; Hilsdale, E.; Hugunin, J.; Kersten, M.; Palm, J.; Griswold, W.(2001) An Overview of AspectJ. In: Proceedings of the 15th European Conference on Object-Oriented Programming, Lecture Notes in Computer Science, Springer-Verlag, London, UK, p. 327–353.
- Mezini, M.; Ostermann, K. (2004) Variability Management with Feature-Oriented Programming and Aspects. Disponível em <http://www.daimi.au.dk/~ko/papers/fse.pdf>
- Hibernate (2008). Hibernate Java persistence framework project. Disponível para acesso na URL: <http://www.hibernate.org/>, em junho de 2008.
- Kang, K.; Cohen, S.; Hess, J.; Novak, W.; Peterson S. (1990). Feature-Oriented Domain Analysis (FODA): Feasibility Study. CMU/SEI-90-TR-21, SEI, USA.
- van der Linden, F.; Pohl K.; Böckle G. (2005) Software Product Line Engineering, Foundations, Principles, and Techniques.
- Meilsmith, G.A (2008). Uma investigação quanto ao uso de aspectos para implementação de variabilidades de Linha de Produto de Software. 65p. Trabalho de Conclusão de Curso (Graduação em Ciência da Computação) Universidade Estadual de Maringá.
- Oliveira Junior, E. A. ; Gimenes, I. M. de S.; Huzita, E. H. M.; Maldonado, J. C. (2005) A Variability Management Process for Software Product Lines. Proceedings of the 15th Annual International Conference of Computer Science and Software Engineering (CASCON), 2005, Ontario - Toronto - Canada : IBM, v. 1. p. 30-44.
- Postgresql (2008). PostgreSQL Global Development Group. Disponível em <http://www.postgresql.org/>. Acessado em junho, 2008.
- SEI - Software Engineering Institute. (2009) <http://www.sei.cmu.edu>. Acessado em outubro, 2009.
- Shimabukuro, E. K. J (2006). Um Gerador de Aplicações Configurável. 134p. Dissertação (Mestrado em Ciência da Computação e Matemática Computacional). ICMC-USP/São Carlos. São Carlos, SP.
- Spring (2009). Spring Framework. Disponível para acesso na URL: <http://www.springsource.org>, em junho de 2009.
- van Gorp, J.; Bosch, J. (2001) On the notion of variability in software product lines. In: The Working IEEE/IFIP Conference on Software Architecture, Amsterdam. *Proceedings*. Amsterdam, 2001.