Cristiano de Macêdo Neto

Utilização de Aspectos como Mecanismo de Implementação de Variabilidade para Instanciação de Componentes de Negócio Reutilizáveis

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica, da Pontifícia Universidade Católica de Minas Gerais, como requisito parcial para obtenção do Grau de Mestre em Engenharia Elétrica.
Linha de Pesquisa: Sistemas Distribuídos e Inteligência Computacional – SIDIC
Orientador: Prof. Dr. Carlos Alberto Marques Pietrobon
Co-Orientador: Prof. Dr. Carlos Augusto P. S. Martins

Belo Horizonte
Pontifícia Universidade Católica de Minas Gerais
2006

"É preciso correr riscos, seguir certos caminhos e abandonar outros.
Nenhuma pessoa é capaz de escolher sem medo"
Paulo Coelho

Ao meu pai Othílio, meu grande mestre e

a minha mãe Miriam meu eterno exemplo.

Agradecimentos

Em especial ao meu orientador Carlos Alberto Marques Pietrobon, pela confiança, amizade, incentivo, suporte e paciência.

Ao meu co-orientador Carlos Augusto P. S. Martins pelo incentivo e amizade.

Aos membros da banca José Luis Braga e Alexei Manso Corrêa Machado por terem aceitado o convite para participarem da banca e contribuírem com o presente trabalho.

À minha irmã Maria Cláudia (Cackau) pelo carinho, amor, paciência e força.

À minha irmã Giovanna pelo carinho, amor, paciência e força nos momentos mais difíceis.

Ao meu irmão Gustavo pelo amor, carinho, paciência e força.

Ao grande amigo Rodrigo Arêas (Maguila) pelos auxílios e por estar sempre presente nos momentos mais difíceis.

Aos meus amigos Dênio, Sávio Grossi, Fabrício Bittencout, João Paulo Barbosa, Fábio Martins, Rodrigo Pagliares e ao Professor Hugo de Paula pelo incentivo e força.

A Maria Isabel Siqueira, secretária do Programa de pós-graduação, pela presteza e atendimento sempre cordial.

Aos professores e funcionários do PPGEE que me transmitiram o conhecimento, a atenção e o suporte necessário.

Ao Diogo Parra dos Santos que me auxiliou com troca de experiências e informações que tanto contribuiu para a conclusão deste trabalho.

À Tia Zélia pelo carinho, atenção, paciência e puxões de orelha.

À Tia Aparecida pela alegria e carinho.

À minha madrinha, Tia Marilda, que esteve sempre ao meu lado, com amor, carinho e atenção.

Ao Tio Pascoal pela atenção e interesse no andamento do trabalho.

À Delane, pelo companheirismo, amor, paciência, atenção e compreensão.

E a toda minha família e todos os meus amigos, que participando da minha vida, participaram também deste trabalho.

Resumo

A Reutilização de software envolve engenharia de domínio, frameworks, padrões, arquitetura de software e desenvolvimento baseado em componentes. Entretanto é necessária uma sistematização para se realizar a reutilização. Este trabalho apresenta uma proposta de utilização de programação orientada a aspectos para instanciar pontos de variabilidade em um framework que descreve um componente de software através de um mecanismo de reuso para componentes de negócios orientados a aspectos. Esta proposta integra os conceitos de aplicação, componentes reutilizáveis, frameworks e aspectos através de um modelo de implementação de variantes em componentes de software.

Palavras Chaves: Programação Orientada a Aspectos, POA, Reuso de Softwares, Componentes, Frameworks.

Abstract

The software reuse involves domain engineering, frameworks, patterns, software architecture and development based on components. However a systematization to the reuse become fulfilled is necessary. This paper presents a proposal of use of aspect oriented programming to instantiate points of variability in a framework that describes a software component through a mechanism of reuse for aspect-oriented business components. This proposal integrates the concepts of application, reusable components, frameworks and aspects into a model of implementation of variants in software components.

Keywords: Aspect-Oriented Programming, AOP, Software Reuse, Components, Frameworks.

Sumário

Lista de Figuras

Lista de Tabelas

Siglas

POA: Programação Orientada a Aspectos.

OO: Orientação a Objetos.

DSOA: Desenvolvimento Sistemas Orientado a Aspectos

DSOO: Desenvolvimento de Sistemas Orientado a Objetos

FOO: Framework Orientado a Objetos.

FOA: Framework Orientado a Aspectos.

OA: Orientação a Aspectos.

FT: Frameworks Transversais.

FAOA: Framework de Aplicação Orientado a Aspectos.

HS: Hot Spot.

FCOA: Framework de Componente Orientado a Aspectos.

PV: Ponto de Variabilidade.

XML: Extensible Markup Language.

1. Introdução

A Reutilização de software é uma das áreas de Engenharia de Software que propõe um conjunto sistemático de processos, técnicas e ferramentas para melhorar a produtividade, a manutenibilidade e a qualidade tanto do software quanto do processo de desenvolvimento. Segundo [FRA 96] a reutilização de software pressupõe o uso de artefatos existentes do software ou o conhecimento para a criação de novo software. Segundo [BOS 99], reutilizar é um propósito enunciado quase que simultaneamente com o surgimento da própria Engenharia de Software.

A Reutilização de software envolve diversos conceitos, ferramentas, tecnologias, tais como engenharia de domínio [SOM 01], frameworks [JOH 92; FAY 97], padrões [FRY 01; BUS 96], linguagem de padrões [GAM 93; GAM 95], desenvolvimento baseado em componentes [DSO 99; CRN 02], geradores de aplicação [FRA 01; BAT 03], e linha de produtos de software [CLE 01]. Entretanto é necessária uma sistematização para se realizar a reutilização, ou seja, é necessária uma forma metodizada e estruturada de se fazer reuso.

O enfoque de linha de produtos propõe a construção sistemática de software baseado em uma família de produtos. Família de produtos de software é 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 (membros) e a representação das diferenças entre produtos. Entre um produto e outro, temos variações.

Segundo [CLE 01] variações ocorrem em pontos de variabilidade – diferenças tangíveis entre produtos que podem ser reveladas e distribuídas entre os artefatos da Linha de Produto (ex. arquitetura, componentes, interfaces e conexões). Variações podem ser reveladas desde a fase de captura dos requisitos. Um ponto de variação identifica uma ou mais localizações nas quais a variação pode ocorrer.

Este conjunto de produtos de uma mesma família de software compartilha uma estrutura comum, que pode ser definida separadamente e que é conhecida por Framework de Software.

Segundo [JOH 97; PRE 97] um framework pode ser definido como um esqueleto da aplicação que será instanciado por um desenvolvedor. [FOO 88] Trata-se de uma aplicação semi-completa reutilizável que, quando especializada, produz aplicações personalizadas.

A estrutura de um framework pode ser definida como um conjunto de classes que contém o projeto abstrato de soluções para uma família de problemas, propiciando o reuso com granularidade maior do que classes [FOO 88]. Um framework é composto por uma coleção de classes abstratas e concretas e de interfaces entre elas, representando o projeto de um subsistema [SOM 01].

O Framework possui pontos, chamados hot spots, onde a variabilidade pode ser implementada, por exemplo, através de especialização de classes que implementam estes pontos.

Além desse, existem vários outros mecanismos para se implementar a variabilidade, como pode ser visto no capítulo 2 do presente trabalho.

Um dos grandes problemas de se trabalhar com frameworks vem a ser a dificuldade de se encontrar, entender e alterar os pontos de variações.

Neste trabalho estamos propondo que se utilize o mecanismo e ferramentas de programação orientada para aspectos (POA) para instanciar pontos de variabilidade em um framework que descreve um componente de software, fazendo uso de conceitos como reuso de componentes.

Para isto, pretende-se especificar como construir um framework que represente um componente de software, a partir de informações obtidas de casos de uso, e que seja estendido para suportar tratamento por aspectos. Esta extensão é importante para se permitir a instanciação os pontos variáveis (*hot spots*) do mesmo. Esta instanciação pode ser realizada por edição direta do aspecto, acrescentando novas instruções ou por meio da inclusão de novos aspectos obtidos de um repositório específico (repositório de aspectos).

1.1. Motivação

Frameworks orientados a objeto (FOO) são coleções de classes organizadas em uma arquitetura abstrata objetivando implementar uma família de problemas relacionados. Como são aplicações incompletas que precisam ser especializadas para gerar aplicações concretas, podemos a partir do reuso do framework, gerar um número indeterminado de novas aplicações.

Com o aparecimento da Programação Orientada a Aspectos (POA) surgiu o interesse em investigar como esses novos conceitos e mecanismos podem ser utilizados no desenvolvimento de frameworks [CAM 04].

Vários autores têm usado o conceito de framework orientado a aspectos (FOA) – um framework orientado a objetos que também utiliza estrutura de aspectos em sua implementação [COM 00], [HAN 2001], [PIN 02], [RAS 03] e [SOA 02].

Entretanto, a maioria dos artigos envolvendo FOA trata de interesses transversais não funcionais como performance, segurança, etc. Encontramos poucas referências ao uso de aspectos para codificação de regras de negócio [SUV 05], [CIB 03] e nenhuma onde elas são utilizadas dentro de um processo intensivo e metódico de reuso de software com componentes de software.

Assim consideramos motivante estudar como aspectos podem auxiliar no reuso de componentes que implementam regras de negócio e como este

reuso pode ser realizado de forma sistemática, apoiada por ferramentas e métodos apropriados.

1.2. Objetivos

Esta dissertação tem como objetivo principal propor um mecanismo de reuso de software através da construção de frameworks de aplicação para um domínio especifico e a construção de frameworks de componentes de negócios que tenham seus pontos de variação implementados através da programação orientada a aspectos.

1.3. Objetivos Secundários

Os objetivos secundários são a apresentação do processo geral que visa atender às expectativas de diversos profissionais da Engenharia de Software: o construtor de framework poderá beneficiar-se do processo de construção de frameworks com base na utilização de aspectos; o construtor de aplicações experiente poderá usufruir do framework para obter suas aplicações concretas em um tempo muito menor do que se as implementasse do zero.

1.4. Organização da dissertação

Este trabalho está organizado em oito capítulos e um apêndice. No Capítulo 2, são fornecidos os conceitos sobre: Reuso e Variabilidade, Aspecto, Variabilidade em Linhas de Produto de Software, Framework, Casos de Uso e Features, Componentes e Reuso de Componentes que proporcionam o

embasamento teórico necessário à compreensão do trabalho de mestrado aqui descrito.

No Capítulo 3, são apresentados os passos seguidos para se integrar frameworks, componentes e aspectos, criando assim uma proposta de modelo lógico de utilização de aspectos para o reuso de alto nível de software para características funcionais em componentes.

No Capítulo 4, apresenta-se a proposta de modelo de utilização de frameworks de componentes de negócios, baseado em variabilidade de casos de uso, baseado no modelo lógico proposto no capitulo 3.

No Capitulo 5, é apresentado um estudo de caso onde são aplicados os modelos propostos nos capítulos 3 e 4, tendo como objetivo a validação dos modelos proposto no trabalho corrente.

No Capítulo 6 apresentam-se as conclusões desta dissertação, com um resumo do trabalho efetuado, as contribuições ao estado da arte, as limitações da abordagem proposta e sugestões de trabalhos futuros.

No Apêndice estão os códigos fontes do Framework Orientado a Objetos e dos Frameworks de Componentes Orientados a Aspectos gerados para o estudo de caso de Sistema de Assinatura de Correio Eletrônico apresentado no capítulo 5.

2. Revisão da Literatura

No decorrer deste capítulo são abordadas as principais técnicas de reutilização de código, bem como, suas variações com maior grau de aceitação tanto no meio científico quanto no mercado de desenvolvimento.

2.1. Reuso e Variabilidade

No universo da engenharia de software, a incessante busca pela produtividade levou à criação de tecnologias voltadas para a produtividade, tanto no momento da criação quanto no momento da manutenção de um produto de software.

Se um grupo de aplicações desenvolvidas tende a atender necessidades semelhantes, o engenheiro de software pode optar por fazer um levantamento de quais partes do software serão comuns a todas as implementações e quais não serão.

Daí vem o conceito de invariabilidade, sendo considerado invariável todas as partes do software que são comuns entre os membros de uma mesma família de aplicações. Por outro lado, são consideradas variáveis aquelas porções que são diferentes entre as diversas aplicações.

O reuso de software pode ser feito em vários níveis de abstração. O código fonte pode ser reusado no modo mais simples que é o copiar e colar,

que não requer nenhuma engenharia. Uma outra idéia para reusar fragmentos do código é a criação de funções e sua organização em bibliotecas como vem acontecendo desde o final da década de 50. Mais recentemente esse procedimento levou à criação de classes de forma arbitrária, quando a programação orientada a objetos ficou popular. Com a idéia de criar componentes estruturados contendo apenas as classes pertencentes a um contexto, o reuso de software se tornou mais sofisticado e engenheiros de software desenvolveram melhores formas de abstração, padrões, frameworks e arquiteturas.

[PAE 99] propôs uma taxonomia geral para reutilização, conforme ilustrado na Tabela 2.1.

Característica	Tipo	Descrição
Aproximação Tecnológica	Por Geração	Parte de uma especificação do problema que será refinada em interações sucessivas.
	Por composição	Utiliza pequenas partes de software como invariantes para prover nova funcionalidade por meio da sua integração.
Enfoque Metodológico	Desenvolvimento para reutilização	Centrado no desenvolvimento de componentes adequados e específicos para um problema especifico – visão provedor.
	Desenvolvimento com reutilização	Centrado no processo de construção de soluções de software a partir de componentes armazenados numa biblioteca – visão demanda.
Modificação	Caixa-Branca	Componentes que são reutilizados por modificação e adaptação.
	Caixa-Preta	Componentes que são reutilizados sem nenhuma modificação.
	Adaptativo	Utiliza grandes estruturas de software como invariantes e restringe a variabilidade a um conjunto de argumentos ou parâmetros.

Alcance do Desenvolvimento	Interno	Mede o nível de reutilização de componentes de um repositório do mesmo projeto.
	Externo	Mede o nível de reutilização de componentes que provêm de repositórios externos ou a proporção de produtos que foram adquiridos.
Alcance do Domínio	Vertical	Reutilização dentro do mesmo domínio de aplicação.
	Horizontal	Reutilização dentro de diferentes domínios de aplicação.

Tabela 2-1: Taxonomia de Reutilização [PAE 99, p. 52]

Segundo [CLE 01] a expressão de uma variação pode ser obtida pela introdução de parâmetros instanciáveis em tempo de construção associados a componentes, subsistemas ou coleção de subsistemas a partir dos quais um produto pode ser configurado atribuindo-se um conjunto de valores a esses parâmetros.

Um tipo de variação simples de ser representado é a escolha de componentes diferentes para uma mesma arquitetura. Assim produtos podem ter maior ou menor capacidade, ou características diferentes dependendo do tipo de componente escolhido pela arquitetura.

Na figura 2-1 tem-se um exemplo de extrato bancário que tem como parte fixa ou estrutura o Histórico de Lançamentos e como ponto de variabilidade o saldo.

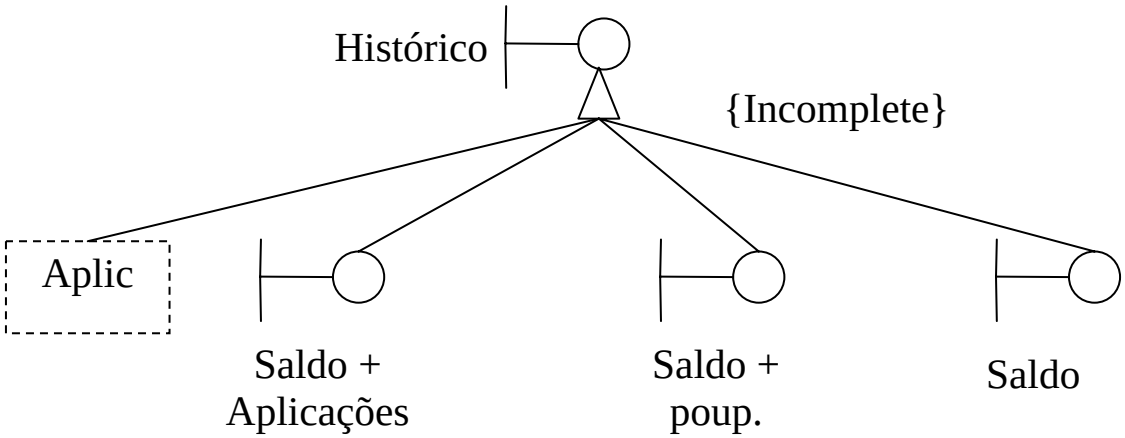


Figura 2-1: Exemplo de variabilidade em Extrato Bancário

Segundo [JAC 97] tem-se, entre outros, os seguintes mecanismos para implementar variabilidade:

- Herança – quando especialização e adição de operações selecionadas, mantendo outras, são necessárias.

- Extensão – quando é necessária a incorporação de uma ou mais variantes nos pontos de variação (ver seção 2.1.5).

- Uses – quando reutilização de um caso de uso abstrato para criar casos de uso especializados é necessária.

- Configuração – quando existem escolhas de funções e implementações alternativas.

- Parametrização – utilizada quando existem muito pontos de variabilidade para cada característica variável.

- Template instantiation – utilizada quando adaptação de tipos ou seleção de partes alternativas de código são necessárias.

- Geração – utilizada quando criando, em larga escala, um ou mais tipos ou classes a partir de uma linguagem específica do problema (ex. Microsoft Wizard).

Neste trabalho propõe-se um outro mecanismo, baseado em programação orientada a aspectos e frameworks, cujos conceitos são abordados a seguir.

2.1.1. Sistema de Aplicação

Segundo [JAC 97] um sistema de aplicação é um produto do sistema entregue fora do negócio de reuso. Quando instalado, um sistema de aplicação oferece um conjunto coerente de casos de uso para o usuário final. Uma família de sistemas de aplicação é um conjunto de sistemas de aplicação com características comuns.

2.1.2. Componente

Segundo [JAC 97] um componente é um tipo, classe ou qualquer outro produto de trabalho que foi especificamente criado para reuso.

Um sistema de componentes é um sistema de produtos que oferece um conjunto de características reusáveis. Características são implementadas como um conjunto de componentes relacionados e interconectados.

2.1.3. Ponto de Variação

Segundo [JAC 97] um ponto de variação identifica um ou mais locais onde a variação irá ocorrer. Para cada ponto de variação podem-se ter diversas variações implementadas.

No projeto existe a necessidade de representar o ponto e o tempo em que as variações são geradas, desenvolvendo assim uma arquitetura que suporta a variabilidade.

Os tipos de variabilidade importantes de serem determinadas são:

- Variação opcional;

Uma variação opcional ocorre quando uma funcionalidade especifica ocorrer em um determinado componente e não ocorrer em outro.

- Uma Variação entre várias alternativas

Uma variação entre várias alternativas ocorre quando para se implementar um ponto de variabilidade é necessário fazer a escolha de apenas uma das alternativas de implementação do ponto de variação em questão.

- Uma combinação de variações entre varias alternativas.

Uma combinação de variações ocorre quando para se implementar um ponto de variabilidade é necessário fazer a escolha de uma combinação de alternativas entre as várias alternativas.

2.2. Programação Orientada a Aspectos

Atualmente as preocupações com manipulação de exceção [LIP 00], distribuição [SOA 02], [GAR 01] e [BAD 99], segurança [FRA 98] e [SHA 03], persistência [YOD 98] e [RAS 03], tolerância a falhas [TIM 02] e caching [NEL 93], juntamente com sua necessidade de integração com as aplicações, obrigam engenheiros de software a investirem muito tempo com a solução destas dificuldades secundárias, em detrimento do trabalho com a regra de negócios que é o principal motivo de se desenvolver software. A Orientação a Objetos muitas vezes, não oferece a melhor solução para solucionar este problema sozinha e com isto, nasceu a necessidade de uma novo paradigma para auxiliar na solução deste problema, a POA.

Segundo [KIC 96] POA se propõe a resolver um problema comum no paradigma OO: a implementação dispersa (scattered, ou “espalhada”) de funcionalidades ortogonais ao modelo de objetos (ver seção 2.2.2).

A POA tem como principal objetivo a consideração e separação de preocupações que se baseiam na teoria de separação de preocupações,

representando uma possibilidade para a modularização de implementação de características de natureza ortogonal (ver seção 2.2.1).

Segundo [RES 05] a POA deve ser vista como um estilo de programação que propicia:

- A implementação e a integração de diversos requisitos funcionais e não funcionais que foram divididos convenientemente em aspectos. Fisicamente essas divisões podem ser os pacotes, uma classe ou um aspecto.

- A alteração da estrutura dos componentes em tempo de compilação, podendo-se inserir novos atributos e métodos.

- A alteração do fluxo de execução de um sistema inserindo uma funcionalidade que não existia anteriormente, sem alterar o código fonte do módulo ou componente alterado.

Segundo [KIC 96] o aspecto é a unidade de modularização da POA que define uma funcionalidade especifica que pode afetar diferentes partes de um sistema e tem como conceitos básicos: pontos de combinações, que são pontos específicos do código da aplicação; entrelaçamento e sugestões, que são a especificação de comportamento associados a um ponto de junção. Estes itens serão discutidos nas sessões seguintes.

Pode-se citar como vantagens de se utilizar aspectos:

- Redução do tamanho do código do elemento

- Redução do emaranhamento do código

- Facilidade de mudanças de implementações técnicas

A figura 2-2 mostra esquematicamente o desenvolvimento orientado a aspectos combinado com o desenvolvimento orientado a objetos apresentado por [SOA 04]. Nela, as preocupações relativas à interface com o usuário e regras de negócios são implementadas utilizando técnicas de orientação a objetos e as preocupações relativas à distribuição, gerenciamento de dados e controle de concorrência utilizam técnicas de Orientação a Aspectos.

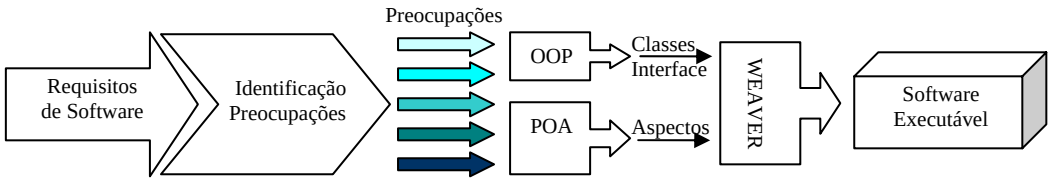


Figura 2-2: DSOA integrado com DSOO [SOA 04]

2.2.1. Separação de Preocupações

Uma preocupação é um requisito ou conjunto de requisitos diretamente ligados ao sistema, apesar de não necessariamente estar diretamente

relacionada com as funcionalidades do mesmo, como por exemplo, persistência, *logging* e tratamento de exceções.

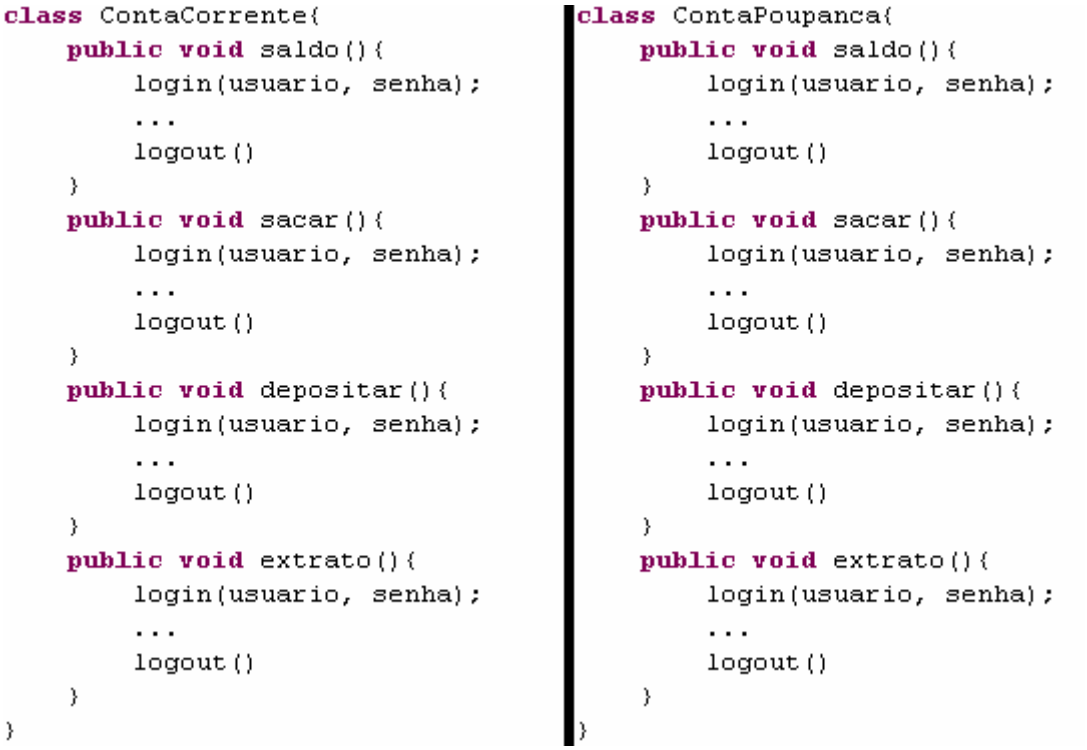
Segundo [RES 05] separação de preocupações é a teoria que investiga como separar as diversas preocupações pertinentes a um sistema, propiciando que cada uma das preocupações seja tratada separadamente a fim de reduzir a complexidade do desenvolvimento, evolução e integração de software. Isto é separação de preocupações é o trabalho de organizar os requisitos de software em nichos de funcionalidades, interesse, responsabilidades, dentre outros.

2.2.2. Espalhamento de Código

Espalhamento de código (spread code) é o fenômeno no qual a implementação de uma ou mais propriedades espalha-se por diversas partes do sistema.

No exemplo da figura 2-3 pode-se observar que os serviços das classes “ContaCorrente” e “ContaPoupanca” precisam executar o *login* antes de atender a cada um dos serviços e *logout* no final dos mesmos [RES 05].

Fazer *login* e *logout* são duas funções relacionadas à segurança que se faz necessária em uma aplicação bancária, apesar de não ser funções diretamente ligadas às classes questão. Por este motivo, são classificadas como características ou aspectos não funcionais da aplicação ou do problema.



```
class ContaCorrente{
    public void saldo(){
        login(usuario, senha);
        ...
        logout()
    }
    public void sacar(){
        login(usuario, senha);
        ...
        logout()
    }
    public void depositar(){
        login(usuario, senha);
        ...
        logout()
    }
    public void extrato(){
        login(usuario, senha);
        ...
        logout()
    }
}
```

```
class ContaPoupanca{
    public void saldo(){
        login(usuario, senha);
        ...
        logout()
    }
    public void sacar(){
        login(usuario, senha);
        ...
        logout()
    }
    public void depositar(){
        login(usuario, senha);
        ...
        logout()
    }
    public void extrato(){
        login(usuario, senha);
        ...
        logout()
    }
}
```

Figura 2-3: Exemplo de Espalhamento de Código em uma aplicação bancária.

2.2.3. Entrelaçamento de código

Entrelaçamento de código (tangling ou tangled code) é o fenômeno no qual a implementação de uma ou mais propriedades mistura-se à implementação de um ou mais partes do sistema.

Segundo [RES 05] quando se faz necessário inserir as chamadas de responsabilidades de uma classe em outra, realizando o entrelaçamento que integra os componentes ao software final, estas linhas de código inseridas em outro componente para integração são denominados código intrusivo ou invasivo.

Por exemplo, na figura 2-4 são mostradas as classes Ponto e Reta. Pode-se observar que o construtor de reta faz chamada de métodos da classe Ponto, o que leva a um acoplamento das classes. As linhas na classe reta, sublinhadas, que chamam métodos de Ponto, são chamadas de códigos intrusivos. [RES 05]

```
class Ponto {
    int x;
    int y;
    public Ponto (int x, int y){
        this.x = x;
        this.y = y;
    }
    public int getX(){return this.x}
    public int getY(){return this.y}
}
class Reta {
    int x1, y1;
    int x2, y2;
    public Reta(Ponto x, Ponto Y){
        this.x1 = x.getX();
        this.y1 = x.getY();
        this.x2 = x.getX();
        this.y2 = x.getY();
    }
}
```

Figura 2-4: Exemplo de Entrelaçamento de Código [RES 05]

Nestes casos, têm-se vários aspectos ou características repetidas em diferentes classes que são características que não estão diretamente ligadas à finalidade ou funcionalidade da classe em questão. Estes aspectos ou características são o que se chama de aspectos não funcionais do objeto.

2.2.4. Ponto de Junção

Ponto de Junção (Join Points) é qualquer ponto identificável de um programa onde se pode interromper o fluxo natural do programa e executar outra rotina denominada sugestões (advice).

Geralmente utilizam-se os pontos de junção para denominar pontos onde pode-se ter diferentes comportamentos ou códigos ou então para denominar uma característica que pode-se comportar de maneira semelhante em diferentes objetos ou classes.

Se for considerada a linguagem de aspectos AspectJ, os JoinPoints são os representados na tabela 2-1 abaixo.

Pontos de Junção	**Descrição**
Chamada e execução de Métodos	Pode-se interromper um programa na chamada ou execução do método. Palavra reservada para isto: call e execution.
Chamada e execução de Construtores	Pode-se interromper um programa na chamada ou execução de construtores. Palavra reservada para isto: new
Métodos de acesso a atributos da classe (get e set)	Pode-se interromper um programa na chamada ou execução de métodos getters e setters (gets e sets) responsáveis por alterar ou obter o valor de um atributo. Palavra reservada para isto: get e set.
Execução de exceções	Pode-se interromper um programa na chamada ou execução do método. Palavra reservada para isto: handler.
Execução de inicialização de classe e objeto.	Pode-se interromper um programa na chamada ou execução do método. Palavra reservada para isto: staticinitialization.

Tabela 2-2: Pontos de Junção

2.2.5. Pontos de Interseção

Os pontos de interseção (Pointcuts) é a declaração dos pontos onde se deseja interceptar a execução de um programa para inserir um novo comportamento.

Um pointcut é declarado utilizando a assinatura do método que se deseja interceptar. Por exemplo, se o interesse é interromper o fluxo (execução) do programa quando houver uma chamada (call) para o método Mult(int, int) que retorna int da classe “Calculos”, então a declaração seria a vista na figura 2-5.

```
import java.util.*; //Biblioteca para usar a classe Vector
public aspect Exemplo_PointCut{
    pointcut PointCut_Exemplo(): call (int Calculos.Mult(int, int));
}
```

Figura 2-5: Declaração de PointCut

2.2.6. Sugestões

As sugestões (advices) vem associada a um ponto de junção e é formado por um conjunto de instruções que poderão ser executadas no momento em que o ponto de junção for alcançado.

As sugestões podem ser executados em três momentos diferentes em relação ao ponto de junção e é determinado através de alguma das três palavras chaves: before, after e around.

Como os próprios nomes definem, o before determina que a sugestão deve ser executado imediatamente antes de um ponto de junção ser interceptado. O after determina que a sugestão deve ser executada imediatamente após de um ponto de junção ser interceptado, podendo ter uma especificação mais detalhada se será executado após um retorno normal ou uma exceção.

O around toma o controle de fluxo de execução do programa, permitindo ao programador determinar qual o próximo método do fluxo de execução que será executado, podendo substituir a execução de métodos e manipular os valores dos argumentos.

As sugestões da linguagem AspectJ estão representados na tabela 2-2 abaixo.

Advices	**Descrição**
Before	Executado após do joinpoint ser alcançado, mas imediatamente antes de seu processamento.
after returning	Executado após o método interceptado rodar com sucesso, ou seja, não ocorrer uma exceção.
after throwing	Executado após o método interceptado rodar sem sucesso, ou seja, ocorrer uma exceção.
Around	Executado quando o joinpoint é alcançado e tem total controle sobre o fluxo de execução do programa.

Tabela 2-3: Descrição dos Advices

Uma sugestão pode ser vista no exemplo da figura 2-6, onde é inserido o comportamento implementado pelo método p1() que imprime na tela e executa uma chamada ao método FunçãoFora(). Este comportamento é

inserido antes do ponto de junção determinado por p1() que é a chamada de instanciação do objeto Cliente.

```
package lib.entidades;
public aspect FuncaoForaAdvice {
    pointcut p1() : call (Cliente.new());
    before() : p1(){
        System.out.println("Capturado: "+thisJoinPoint);
        this.FuncaoFora();
    }
    private void FuncaoFora(){
        System.out.println("Executou FuncaoFora()");
    }
}
```

Figura 2-6: Exemplo de Advice

2.2.7. Introduções

Introdução (Introduction) é utilizada para alterar a estrutura da classe, introduzindo novos atributos, métodos, construtores, getters, setters, herança e interface.

```
public aspect Introducao {
    private String Cliente.nome;
    //Inserindo um construtor
    public Cliente.new(String nome){
        this.nome = nome;
    }
    //Inserindo um setter na classe cliente
    public void Cliente.setNome(String var){
        this.nome = var;
    }
    //Inserindo um getter na classe cliente
    public String Cliente.getNome(){
        return this.nome;
    }
    //Inserindo novos métodos na classe Cliente
    public void Cliente.imprimir(){
        System.out.println("Nome: "+getNome());
    }
}
```

Figura 2-7: Exemplo de Introduction (introdução)

O exemplo da figura 2-7 acima apresenta o funcionamento de uma introdução que altera várias características da classe Cliente. Criando o novo atributo: nome para a classe, bem como os seu getter e setter necessário para o seu funcionamento e criando ainda o método imprimir par a classe. Alterando assim a estrutura da mesma no momento em que o código passa pelo processo de costura.

2.2.8. Combinador

Combinador (weaving) é o processo de composição do código orientado a objetos (aplicação principal) com os aspectos existentes que ocorre geralmente antes do processo de compilação final e é realizada automaticamente por pré-processadores ou compiladores POA.

Segundo [PIV 01] o combinador de aspectos é como um pré-processador que recebe como entrada o(s) programa(s) de componentes e o(s) programa(s) de aspectos e gera arquivos Java que podem ser interpretados normalmente.

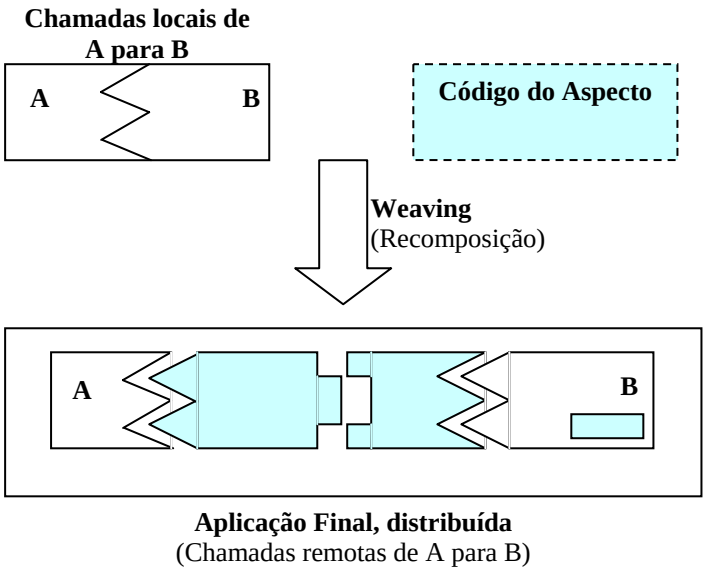


Figura 2-8: O processo de weaving ilustrado através de um aspecto de distribuição [SOA 04]

A figura 2-8 acima, mostra uma aplicação com chamada local de A para B, e o código do aspecto de aspectos (representada na figura por "Código Aspecto") que trata a chamada remota de objetos (distribuição) que após passar pelo weaving gera uma aplicação final distribuída que substitui as chamadas locais de A para B em chamadas remotas.

2.3. Variabilidade em Linhas de Produtos de Software

De acordo com [BAS 98], uma linha de produto de software é uma coleção de sistemas que compartilham um conjunto gerenciado de características através de seus principais artefatos. Estes artefatos incluem uma arquitetura base e um conjunto de componentes para preencher esta arquitetura. O projeto de uma arquitetura para uma família de produtos deve considerar aspectos comuns e as variabilidades entre os produtos.

Variações são diferenças tangíveis entre produtos que podem ser reveladas e distribuídas entre os artefatos da linha de produto, sejam eles a arquitetura, os componentes, as interfaces entre componentes ou as conexões entre componentes. As variações podem ser reveladas em qualquer fase do ciclo de desenvolvimento de uma linha de produtos, a começar da análise de requisitos. A variação mais conhecida em construção de sistemas é a existência de várias implementações para uma mesma especificação.

A expressão de uma variação pode ser obtida pela introdução de parâmetros instanciáveis em tempo de construção associados a componentes, subsistemas ou coleção de subsistemas a partir dos quais um produto pode ser configurado atribuindo-se um conjunto de valores a esses parâmetros [CLE 01]. Um tipo de variação simples de ser representada é a escolha de componentes diferentes para uma mesma arquitetura. Assim, produtos podem ter maior ou menor capacidade, ou características diferentes dependendo do tipo de componente escolhido para a arquitetura.

Existem vários mecanismos para tratamento de variabilidade. Por tratamento de variabilidade entende-se desde o momento de reconhecimento de um ponto de variação até o mapeamento do ponto para uma instância de variação. Entre estes vários mecanismos podemos citar os sete mecanismos citados por [JAC 97] que são: Herança, extensão, uso, configuração, parametrização, instanciação de templates e geração, todos eles já citados no capítulo 02.

Um conceito importante na representação de pontos de variação é o de *feature*. Este conceito tem origem na engenharia de domínio [KAN 90]. Uma "feature" é uma característica de um produto que usuários e clientes consideram importante na descrição e distinção de membros de uma família de produtos [GRI 00]. Uma "feature" pode ser um requisito específico, uma seleção entre os requisitos opcionais e alternativos; ou pode estar relacionada a certas características do produto como funcionalidade, usabilidade e desempenho, ou pode estar relacionado às características de implementação como o tamanho, a plataforma de execução ou compatibilidade com certos padrões.

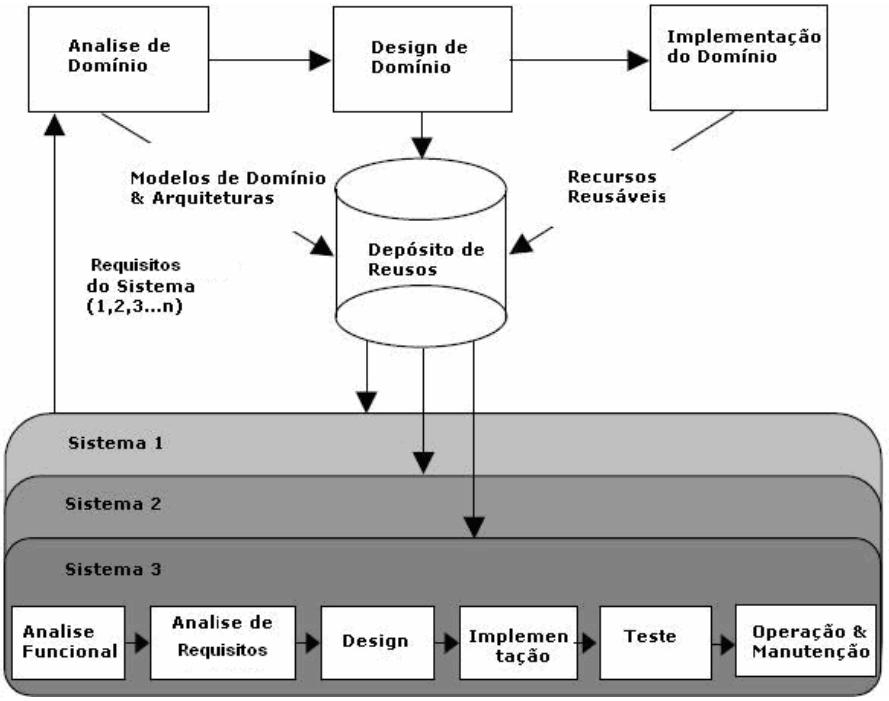


Figura 2-9: Desenvolvendo famílias de software com Engenharia de Domínio [ROB 02]

Segundo [GRI 98] a construção do modelo de features é um processo concorrente que tem como entrada sistemas exemplo, requisitos, especialistas do domínio e modelos anteriores do domínio. Essas entradas são alimentadas em um processo concorrente e contínuo de definição de contexto, modelagem

de features e modelagem de casos de uso. O processo de construção do modelo de features pode ser resumido como segue:

1. mescle os modelos de casos de uso individuais para formar o modelo de caso de uso do domínio, mostrando os pontos de variações;

2. crie um modelo de features inicial como as features funcionais derivadas do modelo de casos de uso do domínio;

3. crie o modelo de objetos de análise, aumentando o modelo de features com features de arquitetura. Essas features estão relacionadas com a estrutura e configuração do sistema ao invés de funções especificas.

4. crie o modelo de projeto, aumentando o modelo de features com as features de implementação.

2.4. Framework

Segundo [GAM 93] framework é um conjunto de classes cooperantes que proporcionam o reuso para uma categoria específica de software. Um framework fornece guia estrutural particionando o design em classes abstratas e definindo suas responsabilidades e colaborações. Um desenvolvedor customiza a framework para uma aplicação particular subclassificando e compondo ocorrências de classes de framework.

[WIA 91] propõe a seguinte definição para framework: “um esqueleto de implementação de uma aplicação ou de um subsistema de aplicação, em um domínio de problema particular. É composto de classes abstratas e concretas e provê um modelo de interação ou colaboração entre as instâncias de classes definidas pelo framework. Um framework é utilizado através de configuração ou conexão de classes concretas e derivação de novas classes concretas a partir das classes abstratas do framework”. R. Wirfs-Brock [WIR 90] estabelece que um framework é “uma coleção de classes concretas e abstratas e as interfaces entre elas, e é o projeto de um subsistema”. Johnson ressalta: “não apenas classes, mas a forma como as instâncias das classes colaboram” [JOH 93].

Com o objetivo de permitir que o framework de software seja reutilizado de forma prática e efetiva, é necessário que ele ofereça um grau de variabilidade de forma que seja útil. Esta variabilidade vai permitir que diferentes sistemas aplicativos sejam construídos.

Para se conseguir esta variabilidade, devem-se identificar as características que variam e modelá-las em pontos do framework que deverão ser compostos e instanciados posteriormente para gerar a aplicação. Estes pontos são chamados de hot spots. Para implementar o hot spot, podemos fazer uso de padrões que auxiliam nas implementações de diversas variantes através de mecanismos de extensão e composição. Segundo [FAY 99][FON 02] e [FON 99], a identificação destes pontos é a atividade mais importante relacionada à construção e instanciação de um framework. Entretanto,

identificá-los apenas não é suficiente: é necessário entendê-los e para poder implementá-los uma boa documentação se torna extremamente importante.

Dois aspectos caracterizam um framework [TAL 95]. O primeiro é que os frameworks oferecem infra-estrutura e projeto: frameworks portam infra-estrutura de projeto disponibilizada, testada e depurada. As interconexões pré-estabelecidas definem a arquitetura da aplicação, liberando o desenvolvedor desta responsabilidade. O código escrito pelo desenvolvedor visa estender ou particularizar o comportamento do framework, de forma a moldá-lo a uma necessidade especifica.

O segundo é que os frameworks “chamam”, não são “chamados”: um papel do framework é fornecer o fluxo de controle da aplicação. Assim, em tempo de execução, as instâncias das classes desenvolvidas esperam ser chamadas pelas instâncias das classes do framework.

Os frameworks são estruturas de classes inter-relacionadas, que permitem não apenas a reutilização de classes, mas minimizam o esforço para o desenvolvimento de aplicações – por conterem o protocolo de controle da aplicação (a definição da arquitetura), liberando o desenvolvedor de software desta preocupação. Os frameworks invertem a ótica do reuso de classes, da abordagem *bottom-up* para a abordagem *top-down*: o desenvolvimento inicia com o entendimento do sistema contido no projeto do framework, e segue no detalhamento das particularidades da aplicação específica, o que é definido pelo usuário do framework. Assim, a implementação de uma aplicação a partir

do framework é feita pela adaptação de sua estrutura de classes, fazendo com que esta inclua as particularidades da aplicação [TAL 95].

De acordo com a forma como deve ser utilizado, um framework pode se classificar como dirigido a arquitetura ou dirigido a dados. No primeiro caso, a aplicação deve ser gerada a partir da criação de subclasses das classes do framework. No segundo caso, diferentes aplicações são produzidas a partir de diferentes combinações de objetos, instâncias das classes presentes no framework. Frameworks dirigidos a arquitetura são mais difíceis de usar, pois, a geração de subclasses exige um conhecimento do projeto do framework, bem como esforço no desenvolvimento do código. Frameworks dirigidos a dados são mais fáceis de usar, porem menos flexíveis. Uma outra abordagem seria uma combinação dos dois casos, onde o framework apresentaria uma base dirigida à arquitetura e uma camada dirigida a dados. Com isto, possibilita a geração de aplicações a partir da combinação de objetos, mas permite a geração de subclasses [TAL 94]. Johnson adota esta classificação de frameworks, porém usa a denominação caixa-branca para o framework dirigido a arquitetura e caixa-preta para o dirigido a dados [JOH 92]. A expressão caixa-cinza é utilizada para se referir à combinação dos dois casos.

Segundo [JOH 97] frameworks orientados a objetos (FOO) são coleções de classes organizadas em uma arquitetura (design) abstrata com o objetivo de implementar uma família de problemas relacionados.

Em termos ideais, um framework deve abranger todos os conceitos gerais de um domínio de aplicação, deixando apenas aspectos particulares para serem definidos nas aplicações especificas. No caso ideal, na geração de aplicações, o usuário do framework não precisa criar classes que não sejam subclasses de classes abstratas do framework. Se isso for alcançado, o framework terá conseguido de fato ser uma generalização do domínio modelado.

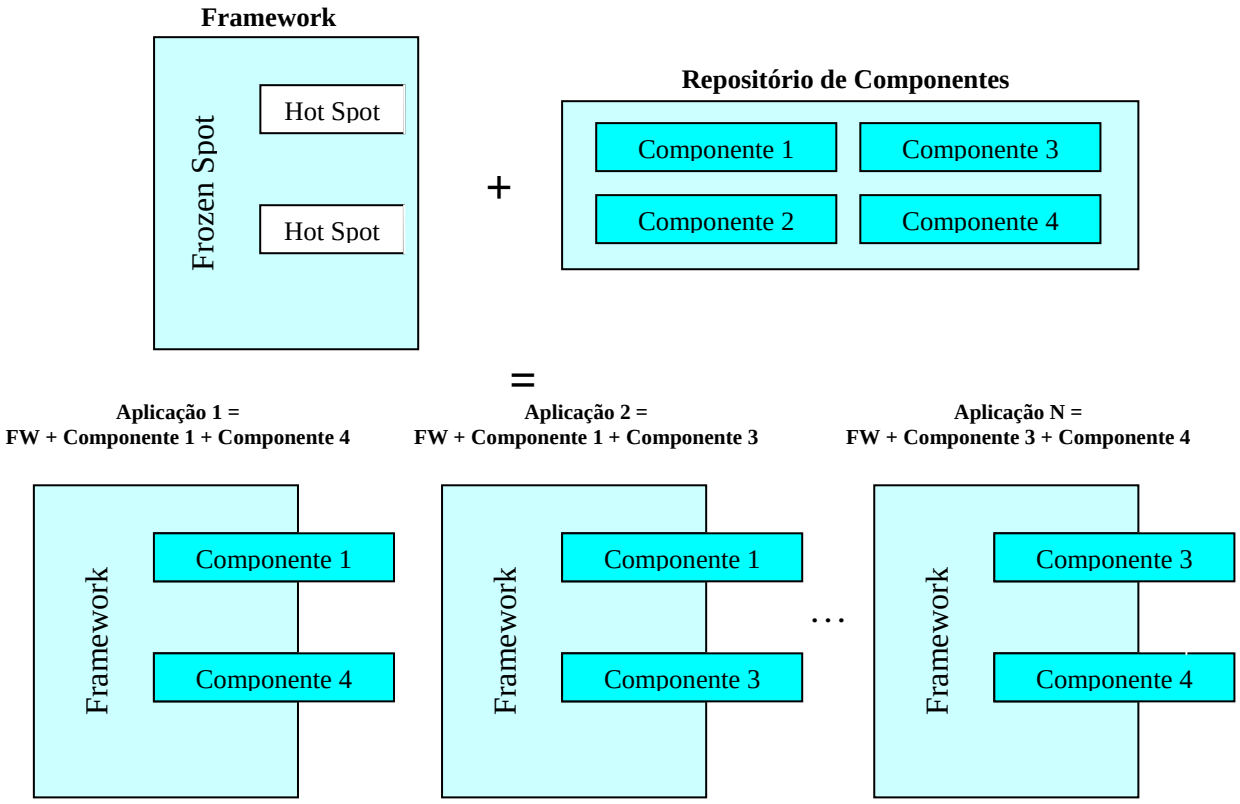


Figura 2-10: Framework Orientado a Objetos

2.4.1. Framework Orientado a Aspecto (FOA)

Segundo [CON 00] um Framework Orientado a Aspectos (FOA) é um Framework Orientado a Objetos (FOO) que também usa estruturas de aspectos em sua implementação.

Framework Orientado a Aspecto (FOA) é um conjunto de componentes (classes, interfaces e aspectos) que agrupa um comportamento ou preocupação genérica e básica de um determinado requisito. A sua utilização vem sendo aplicada a requisitos não funcionais como segurança, entre outros, como citado em [CON 00][RAS 03][SOA 02] e não vem sendo utilizada para instanciar requisitos funcionais ou de negócios de aplicações.

Segundo [JON 98] um FOA é um conjunto formado por unidades básicas de programação OO (classes), cuja presença não é obrigatória, e unidades básicas de programação OA (aspectos). A não obrigatoriedade de classes significa que um FOA pode ser composto apenas por aspectos.

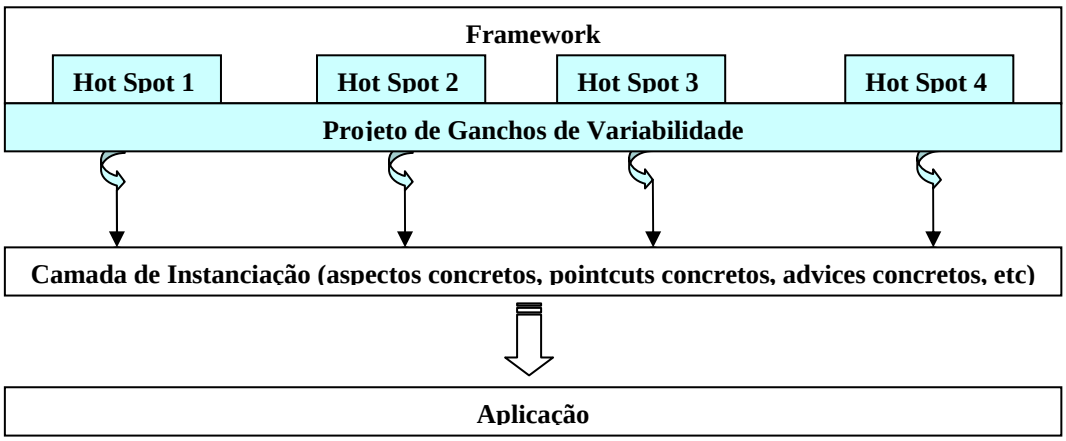


Figura 2-11: Arquitetura de Frameworks Orientando a Aspectos

A figura 2-11 acima mostra a arquitetura de um FOA onde utiliza-se os conceitos de pontos de interseção, pontos de junção da POA podem ser utilizados para transformar os pontos de variabilidade em ganchos a serem implementados pelas sugestões após ser combinado pelo weaver que gera uma aplicação que é a combinação do framework com os aspectos implementados.

Segundo [CAM 05] há dois tipos de FOAs quanto a natureza: Frameworks Transversais e Frameworks de Aplicação Orientados a Aspectos.

2.4.2.1. Framework Transversal (FT)

Um Framework Transversal (Crosscuting Frameworks) ou (FT) é um FOA que possui mecanismos de composição abstratos e variabilidades correspondentes a um único interesse transversal, como por exemplo: persistência, distribuição, segurança e regras de negócios.

2.4.2.2. Framework de Aplicação Orientado a Aspectos (FAOA)

Um framework de aplicação orientado a aspectos (FAOA) é um FOA que implementa uma arquitetura genérica de um domínio e sua instanciação produz uma aplicação desse domínio. A arquitetura FAOA é projetada com classes e aspectos integrados dentro da arquitetura de um framework. Eventualmente podem ser acoplados outros FTs que implementem interesses que estejam fora

de seu escopo. Os FAOA são bastante parecidos com os frameworks de aplicação OO com relação a seu propósito e suas diferenças estão na sua arquitetura, que usa aspectos abstratos e concretos, bem como classes abstratas e concretas para implementar partes variáveis de seu código e que, portanto, são concretizados quando ocorre a instanciação de uma aplicação.

2.4.2.3. Formas de Reuso

Segundo [CAM 05] há três formas possíveis de reuso quando se trata de frameworks orientado a aspectos e muitas vezes, sua natureza (FT ou FAOA) determina essa forma.

As formas possíveis são:

1) Apenas instanciação;

2) Apenas composição

3) Instanciação e composição.

A primeira forma ocorre somente com FAOAs porque esses já possuem o código-base embutido em sua estrutura, não necessitando ser acoplados a nenhum outro código-base. Além disso, a maioria de seus interesses transversais já está codificada internamente, assim como suas regras de composição.

A segunda ocorre somente com FTs de uma única funcionalidade.

A terceira forma de reuso é um pouco mais complexa porque utiliza as duas etapas. Essa forma ocorre com mais freqüência com FTs que possuem mais de uma variabilidade, mas também pode ocorrer com FAOAs.

Nas duas últimas formas de reuso, a composição pode ocorrer entre dois ou mais FTs e regras de prioridade de atuação precisam ser definidas. Os FTs são os que ocorrem mais comumente nessa categoria. Contudo, nada impede a existência de FAOA cujo processo de instanciação também inclua a etapa de composição.

2.4.2.4. Idiomas para Construção de Frameworks

Os idiomas são padrões de projetos dependentes de linguagem que já fornecem soluções implementadas.

Em [HAN 03] são apresentados quatro idiomas avançados: Template Advice, Pointcut Method, Chained Advice e Factory Advice. Eles são idiomas que foram utilizados em larga escala com sucesso em aplicações corporativas.

2.4.2.4.1. Template Advice

Segundo [HAN 03] o idioma Template Advice é bastante parecido ao *Template Method* proposto por [GAM 00] e deve ser usado sempre que um procedimento adicional for executado por um determinado join-point. Isto

significa que o código a ser executado consiste de uma parte fixa e uma parte variável, onde a parte variável muda de aplicação para aplicação. O modelo corresponde diretamente ao padrão de projeto, onde o template é especificado dentro da recomendação ao invés do método, conforme apresentado na figura 2-12.

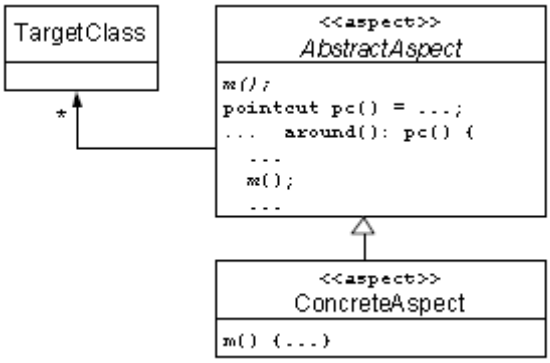


Figura 2-12: Template Advice [HAN 03]

A motivação deste idioma é a incapacidade da linguagem AspectJ em fornecer sobreposição de sugestões e consequentemente de fornecer diferentes comportamentos para uma única sugestão.

2.4.2.4.2. Pointcut Method

Segundo [HAN 03] o idioma Pointcut Method é usado quando um certo advice é necessário, cuja execução depende de elementos específicos da execução ou de valores específicos do pointcut, conforme apresentado na figura 2-13.

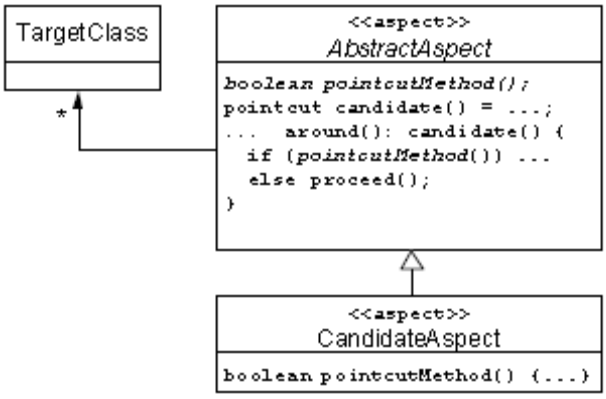


Figura 2-13: Pointcut Method [HAN 03]

Esse idioma é geralmente utilizado em conjunto com o *Template Advice*, em que o método *pointcutMethod()* faz o papel de método gancho.

2.4.2.4.3. Chained Advice

Segundo [HAN 03] o idioma Chained Advice é usado quando um comportamento de objetos pode mudar os seus métodos por causa de diferentes decisões e mais adiante o numero de aspectos e o uso de advices em cadeia é recomendado, conforme apresentado na figura2-14.

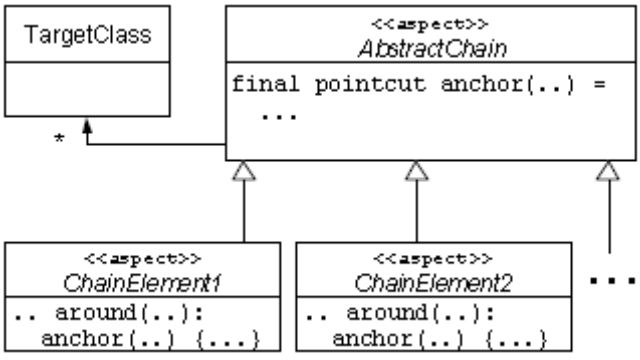


Figura 2-14: Chained Advice [HAN 03]

Esse idioma é geralmente utilizado com o *Pointcut Method* para determinar qual dos elementos da cadeia deve ser executado.

O idioma *Chained Advice* é recomendado quando há um conjunto de pontos de junção que são compartilhados por vários aspectos, pois ele permite especificar vários trechos de sugestões no mesmo ponto de corte por meio da criação de vários aspectos concretos e um abstrato.

2.4.2.4.4. Factory Advice

Segundo [HAN 03] o idioma Factory Advice é usado sempre que a criação de um objeto depender de um aspecto especifico que pode variar de aplicação para aplicação ou a execução de um aplicação, o uso de uma construção de advices é recomendada, conforme apresentado na figura2-15.

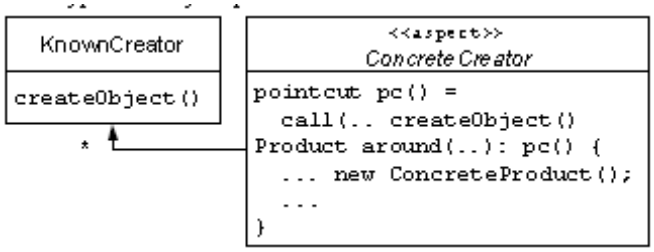


Figura 2-15: Factory Advice [HAN 03]

Este idioma é utilizado quando a instanciação de um objeto deve considerar características especificas de uma aplicação e/ou depende do contexto de execução da aplicação.

2.5. Casos de Uso

Tem-se diferentes definições de casos se uso, entre elas:

- Um caso de uso especifica uma seqüência de ações, inclusive variantes, que em um sistema realiza e que produz um resultado observável de valor para um ator particular [JAC 99].

- Um caso de uso é uma descrição de todas as possíveis seqüências de interações entre o sistema e um ou mais atores em resposta a algum estimulo inicial devido a um dos atores [RUM 94].

- Um caso de uso é uma coleção de possíveis seqüências de interações entre o sistema sob discussão e seus atores externos, relacionada com um objetivo em particular [COC 00].

Segundo [FUR 98] a modelagem de caso de uso é utilizada não somente para capturar necessidades de um novo sistema, mas também para desenvolver novas versões de um sistema. Nesse sentido, a nova funcionalidade é adicionada ao contexto do modelo de caso de uso através da inserção de novos atores e casos.

Segundo [AHM 02] existem dois conceitos fundamentais na modelagem do caso de uso:

- Ator: um ator representa algo (ou alguém) fora do sistema, geralmente um usuário do sistema. Os atores interagem com o sistema, que resulta em alguma ação feita pelo sistema. Cada papel distinto é representado por um ator.

- Caso de uso: um caso de uso encapsula uma seqüência de etapas executadas pelo sistema em nome de um ator. Os casos de uso fornecem algum valor para o ator. Um caso de uso consiste em uma seqüência primária de eventos e pode ter uma ou mais seqüências alternativas de eventos.

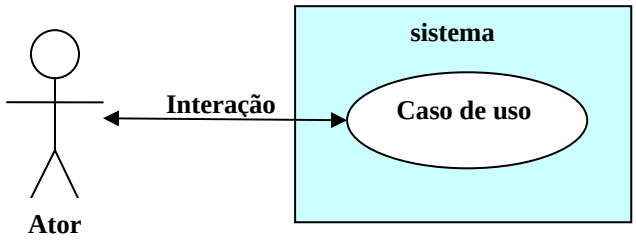


Figura 2-16: Diagrama de Caso de Uso

Segundo [JAC 97] o primeiro passo para criar uma aplicação particular é transformar os requisitos específicos em um modelo de caso de uso. Quando se deseja reusar os componentes, deve-se começar expressando os casos de usos específicos e as características da aplicação nos termos mais genéricos dos casos de uso oferecidos pelos componentes de casos de uso.

O modelo de casos de uso deve precisamente descrever como responsabilidades são dadas a atores e casos de uso.

Segundo [JAC 97] uma responsabilidade é algo que um ator ou caso de uso precisa fazer ou ter conhecimento, enquanto o sistema é usado. No geral uma responsabilidade é um contrato ou uma obrigação de uma classe ou tipo.

Como exemplo pode-se citar o caso de uso **Autenticação de Usuário** que pode, por exemplo, ter as responsabilidades: aguardar pela tentativa de autenticação do usuário, validar a tentativa de autenticação de usuário, reconhecer a tentativa correta de autenticação de usuário e desconectar usuário.

Segundo [JAC 97] atores e componentes de caso de uso podem ser concretos e reusáveis, como componentes comuns sem mudanças, ou serem abstratos, o que significa que eles precisam ser especializados antes de reusados. Por exemplo, um componente de caso de uso **Autenticação de Usuários** pode ser reusado sem especialização, se vários sistemas precisam de exatamente o mesmo procedimento de autenticação.

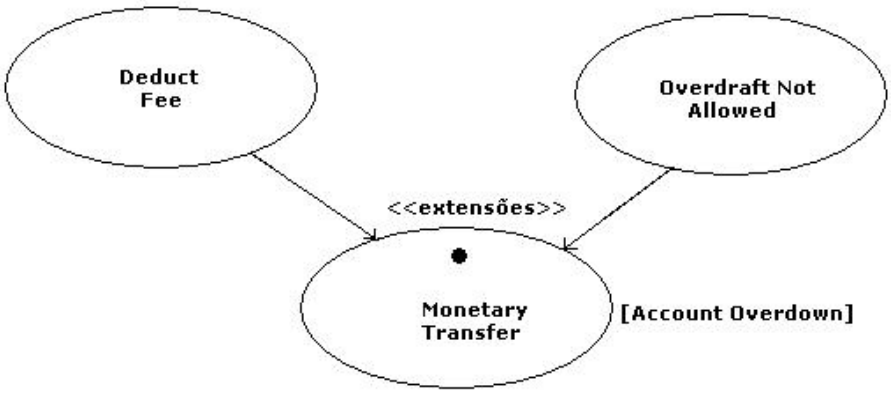
Por outro lado há várias situações quando casos de uso abstratos e componentes atores precisam ser especializados. Um exemplo é quando o caso de uso de várias aplicações tem quase a mesma descrição, mas com variações como usuário ou erros do sistema.

2.5.1. Caso de Uso e Variabilidade

Como vimos casos de uso podem ser definidos de forma abstrata precisando ser especializados para poderem ser utilizados ou reutilizados em aplicações ou componentes específicos.

Isto acontece porque cada aplicação ou componente especifico requer uma variabilidade do caso de uso. Esta variante é função da instanciação dos diversos pontos de variabilidade encontrados no caso de uso.

Existem diversas maneiras para expressar os pontos comuns e os pontos variáveis de um caso de uso, além de diferentes mecanismos para implementar a variabilidade, tais como templates de caso de uso, macros, parâmetros, heranças de caso de uso (uses) e extensões de caso de uso.



Exemplo: Como mostrado, o componente de sistema *Account Management* oferece o componente caso de uso *Monetary Transfer*. Junto com o caso de uso estão também dois componentes variáveis com diferentes estratégias para saque na conta. Uma variável deduz uma taxa para cada saque. A outra não permite saques. Essas variantes são associadas com o ponto de variação *{Account Overdraw}*.

Figura 2-17: Exemplo de Pontos de Variabilidade em Casos de Uso. [JAC 97]

A variabilidade é usada para permitir variações nos requisitos funcionais e também nos requisitos não funcionais. É mais fácil entender a necessidade de variabilidade quando observamos as diferentes categorias de variabilidade para ver como elas podem ser representadas e implementadas. Algumas razões típicas para explorar a variabilidade em um componente de caso de uso estão mostradas abaixo [JAC 97].

- Variando usuários ou interfaces de sistema – Diferentes tipos de usuários, ou diferentes especializações de casos de uso podem especificar interfaces de usuário diferentes. Também, instalações diferentes de um sistema ou ambientes de operação diferentes podem requerer interfaces de sistemas diferentes ou *drivers* customizados. Isto é observado, por exemplo, em caixas automáticos de bancos.

- Diferentes tipos de entidades referenciadas – Por exemplo, um caso de uso *Account* pode ser uma conta de cheque ou conta conjunta. Tais variações freqüentemente correspondem diretamente a um tipo-entidade no objeto modelo.

Exemplo: O uso do componente de caso de uso *Instrument Transfer* pode ser expresso em termos de *Portfolio* e *Instrument*. Tal como o caso de uso *Instrument Transfer* é especializado para *Monetary Transfer*, um *Portfolio* e um *Instrument* são especializados para um *Account* ou *Money*.

Figura 2-18: Diferentes Tipos de Entidades Referenciadas em Caso de Uso [JAC 97]

- Funcionalidades Alternativas e Opcionais – Variantes diferentes de casos de uso podem oferecer comportamentos alternativos ou opcionais.

Exemplo: Um *Bank Costumer*, atuando no papel de *Buyer*, pode desejar uma confirmação quando o ator *Seller* receber o pagamento de uma fatura. Mas apenas algumas instalações do caso de uso *Pay Schedule Invoices* iram dar ao *Buyer* tal confirmação, já isso pode ser um requerimento adicional processando durante a instalação. Pode haver varias variantes alternativas para o componente de caso de uso *Pay And Schedule Invoices* para suportar as alternativas.

Figura 2-19: Funcionalidades Alternativas e Opcionais em Caso de Uso [JAC 97]

- Variando restrições e regras de negócio – Variantes diferentes de caso de uso podem ter várias restrições na ordem em que as tarefas são processadas. Elas podem ter pré-condições diferentes ou restrições do que os casos podem executar e pode haver diferentes regras de negócio para serem aplicadas.

Exemplo: Em algumas instalações o componente de caso de uso *Pay And Schedule Invoices* precisa checar se a conta não irá ficar sem fundos quando pagar a fatura.

Em outras instalações, cada instancia do componente de caso de uso precisa checar por uma possível fraude antes da aprovação. Isso é feito comparando as faturas contra padrões normais de pagamentos de fatura associadas com o *Bank Costumer* e *Account* usados.

Figura 2-20: Variando Restrições e Regras de Negocio em Caso de Uso [JAC 97]

- Detecção de erros – A melhor estratégia para detecção de erros e recuperação de um caso de uso freqüentemente depende do tipo de interface do usuário, ou do tipo de conta ou outro objeto relacionado.

Exemplo: Algumas instalações podem violar o saque em *Account* durante o caso de uso *Pay And Schedule Invoices* ao invés de pagar uma fatura com uma conta diferente da conta padrão.

Figura 2-21: Detecção de Erros em Caso de Uso [JAC 97]

- Performance e diferenças de escalabilidade – Diferenças em requerimento de performance, restrições de tempo e um numero de atividades urgentes podem variar entre diferentes instalações de um sistema.

Exemplo: Diferentes instalações de sistemas de caixas automáticos podem ser liberadas para tolerar tempos de resposta diferentes no caso de uso *Withdraw Money* antes de tratar a demora como erro. Também, a quantidade de tempo para tentar conectar a outro banco antes da probabilidade de *time out* depende da qualidade da rede de telecomunicação.

Finalmente, algumas instalações do internet *home-banking* do sistema *Digicash Cashier* precisam ser capazes de simultaneamente servir 100 requisitos de usuários. Em algumas instalações 20 podem ser suficientes.

Figura 2-22: Performance e Diferenças de Escalabilidade em Caso de Uso [JAC 97]

2.6. Features (Características)

Segundo [JAC 97] característica é um caso de uso ou uma responsabilidade de um caso de uso.

Segundo [KAN 90] um conceito importante na representação de pontos de variação é o de feature. Este conceito tem origem na engenharia de domínio. Segundo [GRI 00] uma feature é uma característica de um produto que usuários e clientes consideram importante na descrição e distinção de membros de uma família de produtos.

Segundo [GRI 98] apesar de features apresentar alguma relação com o modelo de casos de uso, apontam algumas diferenças entre esses modelos, que incluem:

- o modelo de caso de uso é orientado para o usuário e o modelo de features é orientado ao reutilizador;

- o modelo de casos de uso descreve os requisitos do usuário em termos de funcionalidades do sistema. O modelo de features organiza o resultado da análise dos aspectos comuns e variáveis, preparando uma base para reutilização;

- o modelo de casos de uso deve cobrir todos os requisitos de um sistema individual do domínio, enquanto que o modelo de features deve incluir apenas aquelas características que o analista do domínio considera importante pois este resume apenas os itens essenciais relativos aos objetivos do domínio;

- a notação utilizada para representar features é diferente daquela utilizada nos modelos de casos de uso pois nem todas as features podem ser automaticamente relacionadas com casos de uso;

- nem todas essas features aparecem nos casos de uso apesar de existir um grupo de features básicas que o usuário vê como capacidades do sistema. Algumas features surgem: no detalhamento da implementação; nas opções de configuração do sistema, ou por sugestão de especialistas do domínio. Tais features podem aparecer somente nas fases de projeto e implementação.

Um modelo de features é, geralmente, representado por um grafo em que os nós contêm as features e seus atributos. Este grafo contém decorações para indicar features opcionais, obrigatórias e composição de features, conforme diagrama das características de um sistema de correio representado na figura 2-23 abaixo.

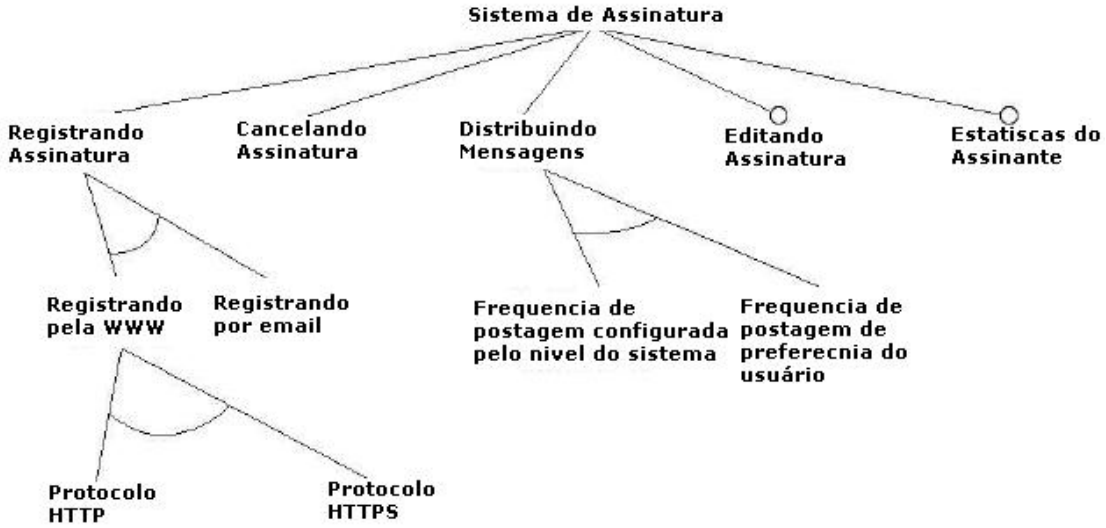


Figura 2-23: Diagrama de característica de um sistema de assinatura de e-mail [ROB 02].

Na figura 2-23 acima temos um de diagrama de características de um sistema de assinatura de correio que é composto pelas seguintes características obrigatórias: Registrando Assinatura, Cancelando Assinatura, Distribuindo Mensagem. E as seguintes características opcionais: Editando Assinatura e Estatísticas do Assinante. A característica Registrando Assinatura apresenta duas variantes que são: Registrando pela www e Registrando por E-mail. A variante Registrando pela www possui também duas variantes que são: Protocolo http e Protocolo https. A característica distribuindo mensagem também possui duas variantes que são: Freqüência de postagem configurada por nível do sistema e Freqüência de postagem de preferência do usuário.

2.7. Características e Componentes

No presente trabalho fala-se de características relacionadas a casos de uso e requisitos. Dar-se o nome de características às diferenças encontradas entre sistemas similares na base da funcionalidade, serviços, e muitas outras características que estes podem oferecer, ou seja, o nome característica é dado para se discutir variabilidade em sistemas de aplicação ou componentes.

Se analisarmos o modelo de features apresentado na seção 2.6 do presente capítulo, pode-se perceber que uma das maneiras de implementá-lo de forma eficiente para o reuso é através da utilização do conceito de componentes, onde pode-se transformar os nós superiores do grafo de features em componentes e os nos filhos ou folhas em pontos de variante do componente, conforme representado pela figura 2-24 abaixo.

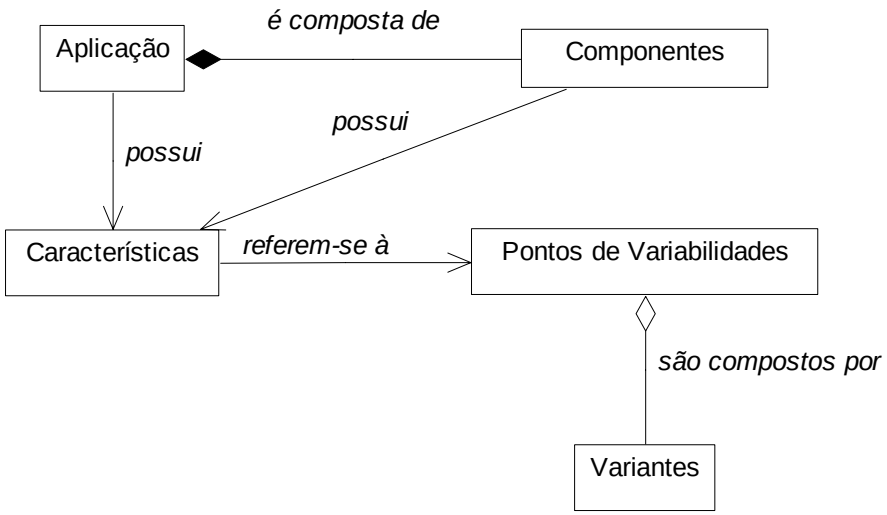


Figura 2-24: Relacionamento entre Features e componentes

2.8. Componentes e Variabilidade

Segundo [JAC 97] desenvolvendo se vários sistemas relacionados, sempre faz sentido criar uma série de componentes reutilizáveis que desenvolvedores de quaisquer modelos possam reutilizar sistematicamente.

Os componentes são compostos por pontos de variações que podem ou não conter variações dentro de um contexto ou quando especializado que é representado pela figura 2-25.

Segundo [JAC 97] um componente abstrato é especializado integrando uma ou mais variantes a cada um dos seus pontos de variação.

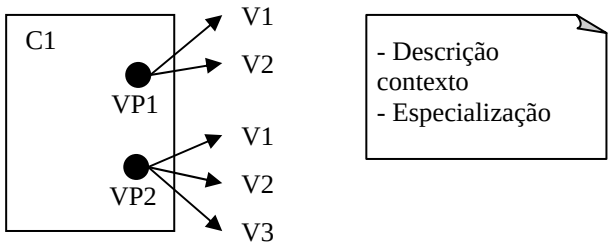


Figura 2-25: Representação de pontos de variabilidades em componentes

A figura 2-25 apresenta um componente C1 que é composto por 2 pontos de variabilidades que são chamados de VP1 e VP2, estes pontos podem conter ou não variantes e no caso do exemplo apresentado, VP1 contem duas variantes que são V1 e V2 e VP2 contem 3 variantes que são V1,

V2 e V3. Que conforme o contexto de utilização do componente terá a utilização de diferentes variantes.

Para se implementar o modelo de variabilidade de componentes acima, pode-se utilizar os 7 (sete) mecanismos propostos por [JAC 97] e apresentado na tabela abaixo:

Mecanismo	Tipo de ponto variante	Tipo de variante	Uso especializado quando
Interface	Operação virtual	Subclasse ou subtipo	Especializando ou adicionando operações selecionadas, enquanto mantendo outras.
Extensões	Ponto de extensão	Extensão	É necessário incluir várias variantes a cada ponto de extensão ao mesmo tempo.
Usos	Ponto de uso	Caso de uso	Reusando um caso de uso abstrato para criar um caso de uso especializado
Configuração	Slot de configuraçao de item	Item de configuração	Escolhendo funções alternativas e implementações
Parâmetros	Parâmetro	Parâmetro de limite	Há vários pequenos pontos de variação para cada característica variável
Incialização de templates	Parâmetro template	Instancia template	Fazendo adaptação de tipo ou selecionando partes alternadas do código
Geração	Parâmetro ou linguagem scrit	Parâmetro de limite ou expressao	Fazendo criação em larga escala de um ou mais tipos ou classes de uma linguagem de problemas

Tabela 2-4: Mecanismos de Implementação de variante em componentes

O presente trabalho propõe a utilização de POA (Programação Orientada a Aspectos) como um mecanismo alternativo de implementação de variante em componentes seguindo o modelo apresentado no capitulo 3 para a utilização nos tipos de ponto variante em operações virtuais e em pontos de extensão, servindo como alternativa de implementação aos modelos de Interface e extensões proposto por [JAC 97].

2.9. Reflexão Computacional

Segundo [FAB 95] a reflexão computacional é o processo em que um sistema pode analisar seu próprio comportamento e atuar sobre o mesmo. Em um sistema convencional, a computação é realizada sobre dados que representam entidades externas ao sistema. No entanto, um sistema reflexivo deve conter dados que representam aspectos estruturais e computacionais do próprio sistema. Também deve ser possível acessar e manipular tais dados a partir do sistema e, principalmente, tais dados devem estar conectados causalmente ao comportamento do sistema. Ou seja, alterações nesses dados devem causar alterações no comportamento do sistema e vice-versa.

```
public class Exemplo1 {
    public static void main(String[] args) {
        Integer i = new Integer(5);
        Class c = i.getClass();
        System.out.println("Classe do objeto: " + c.getName());
    }
}
```

Figura 2-26: Obtenção do Nome da Classe através de Reflexão Computacional.

No exemplo da figura 2-26, o nome da classe referente ao objeto i é obtida através de reflexão computacional e impressa.

3. Framework, Componentes e Aspectos

Neste capítulo apresenta-se como integrar framework, componentes e aspectos objetivando o reuso de componentes de software.

Mais especificamente, mostra-se como desenvolver um componente de software a partir de um framework, onde os hot spots são resolvidos por meio de aspectos. O processo para se construir estes componentes e integrá-los à aplicação alvo, é apresentado no próximo capítulo.

O que queremos é que possamos definir novas aplicações a partir da escolha de componentes (casos) apropriados e de configuração destes casos/componentes para situações especificas. A configuração destes casos/componentes se faz por meio da seleção de variantes para cada ponto de variabilidade importante para a aplicação.

A utilização destas variantes ocorre localizando os pontos de variabilidade no código e inserindo um novo código. Encontrar estes pontos de variabilidade não é uma atividade fácil. Entender o que codificar no ponto de variabilidade também não é trivial. Buscando soluções para este problema, percebe-se que a tecnologia desenvolvida para POA ajuda a executar estas duas atividades.

Existe um paralelo entre aspectos e frameworks: uma estrutura principal do programa aplicativo (equivale ao framework) e pontos (pointcuts) onde se

tem que acrescentar código para instanciar o programa para que ele vire uma aplicação real, executável (equivale aos hot spots).

O mecanismo de costura Weaving permite encontrar pontos no código onde se tem a necessidade de acrescentar o aspecto. No framework, temos que encontrar os pontos no mesmo onde vamos instanciar o framework. Entretanto, a tecnologia de frameworks não possui ferramenta ou técnica para se encontrar estes pontos. Por outro lado, ao projetar-se um framework sabem-se quais são os pontos de variação que recebem o nome de hot spot. Neste momento, devemos registrar esta informação para podermos utilizá-la no futuro. Como fazer isto será abordado no capítulo seguinte. Por ora, vai ser apresentado como associar hot spots a aspectos.

3.1. Caracterização do problema

A proposta do presente capítulo é mostrar o caminho para a criação e implementação de componentes de negócio configuráveis que possam ser facilmente utilizados em diferentes aplicações.

3.2. Solução Estrutural

Para a criação e implementação de componentes de negócio configuráveis propõe-se a criação de um FOO de aplicação que tem os seus hot spots compostos por frameworks de componentes com a utilização de

programação orientada a aspectos que se chama de Framework de Componente Orientado a Aspectos (FCOA), conforme apresentado na figura abaixo.

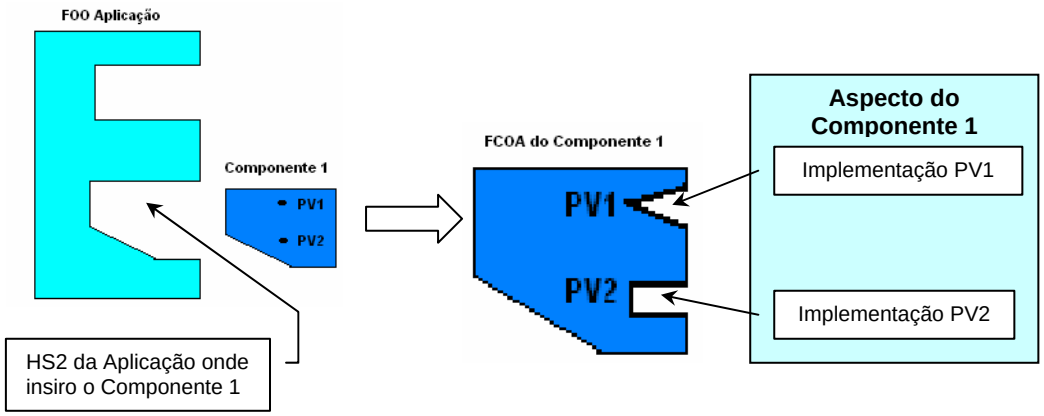


Figura 3-1: Solução Estrutural

3.3. Associação de Aplicação com Componentes

Diferentes aplicações de uma mesma família de software podem ser compostas por componentes que são criados a partir de um conjunto de casos de uso ou de apenas um caso de uso, que pode ter diferentes atividades implementadas em função de características variáveis deste componente.

Desta forma, temos dois frameworks: o primeiro para a família de aplicações e o segundo para o componente de software.

Quanto ao desenvolvimento do primeiro framework citado acima, temos uma vasta quantidade de material e técnicas para sua definição e construção e

que já estão consolidados na área de engenharia de software e por este motivo, não serão abordados no presente trabalho.

Assim, o modelo que vislumbramos consiste no desenvolvedor encontrar e ou selecionar os casos de uso pertinentes para sua nova aplicação e para cada caso de uso ajustar, selecionar e ou implementar as variantes do caso de uso, de acordo com a aplicação que esta sendo construída, construindo assim o primeiro framework que é relativo a família de software da aplicação.

E como um componente pode conter características variáveis, onde cada versão alternativa das características deve ser similarmente associada à uma ou mais variantes. É perfeitamente possível a utilização do frameworks para a implementação de componentes a serem projetados e utilizados para reuso, conforme ilustra a figura 3-2.

Figura 3-2: Framework como alternativa de Implementação de componentes

A figura 3-2 mostra a utilização de frameworks como uma alternativa de implementação de componentes, onde os pontos de variabilidades do componente são transformados em hot spots e a estrutura do componente

onde não ocorre a variação é transformada em frozen spots ou estrutura estática do framework de componente. E os novos componentes são gerados a partir da combinação do framework de componentes com as variantes escolhidas para cada hot spot do mesmo.

3.3.1. Componente Exemplo

Nesta seção utiliza-se um exemplo para facilitar a implementação dos conceitos de framework de componentes e aspectos. O exemplo consiste no componente de extrato bancário, por ser um exemplo de fácil entendimento e apresentar variações entre os componentes de uma aplicação dentro de uma mesma instituição bancária ou em diferentes instituições bancárias.

3.3.2. Identificação das Estruturas da Aplicação

O componente deve gerar um relatório com a seguinte estrutura:

- Cabeçalho.

  O cabeçalho será a parte do relatório comum a todas as páginas, onde deve conter informações como instituição bancária, data, cliente, entre outros.

- Histórico de Lançamentos.

O Histórico de lançamentos é a parte idêntica a todos os extratos bancários, onde consta todas as operações realizadas na conta em questão.

- Saldo.

O saldo pode variar entre tipos de clientes, tipos de contas e instituições bancárias, uma vez que pode ser associados uma série de novos produtos a ela.

- Rodapé.

O rodapé será a parte do relatório comum a todas as páginas, onde deve conter informações como instituição bancária, data, cliente, entre outros.

Analisando-se estes elementos definiu-se os pontos de variabilidade: Cabeçalho, Saldo, Rodapé.

Como premissa do exemplo o Histórico de Lançamento não possui variações dentro de diferentes componentes, e por este motivo, no desenvolvimento do framework, ele é considerado um frozen spot ou ponto fixo.

3.3.3. Hot Spots

Os Hot spots tratam-se de pontos de variabilidade e têm diferentes implementações e soluções.

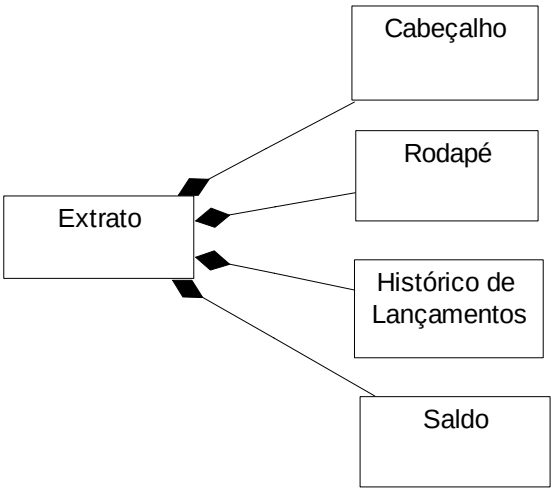


Figura 3-3: Diagrama de Classes que representa o componente de extrato bancário

A figura 3-3 acima representa o componente de extrato bancário que é composto por Cabeçalho, Rodapé, Histórico de Lançamento e Saldo e pode conter mais alguma característica não definida neste problema, representada pela *label {Incomplete}*. Deve-se levar em consideração que a única característica fixa no componente apresentado é o Histórico de Lançamento e que todos os outros apresentam pontos de variabilidade que podem conter várias implementações diferentes, dependendo do contexto em que está sendo aplicada.

3.3.4. Separação de Preocupações

Neste trabalho, a utilização da separação de preocupações difere um pouco da maioria das aplicações em que vem sendo utilizada, uma vez que se tem a utilização de aspectos para a implementação de preocupações funcionais do problema em forma de aspecto, desde que a mesma represente pelo menos um ponto de variação.

Para se chegar à escolha da melhor maneira de fazer a separação de preocupações, foram formuladas as seguintes questões: “tem-se um ponto de variação para cada preocupação?” ou “pode-se ter um ou mais pontos de variação para cada preocupação?”.

No primeiro caso, tem-se uma preocupação para cada ponto de variação e no segundo caso, tem-se uma preocupação para cada grupo de pontos de variação de uma mesma preocupação.

Tendo em vista que é perfeitamente possível a existência de diferentes preocupações em uma mesmo componente ou aplicação, e dentro destas preocupações, podem existir diferentes grupos de pontos de variação, é necessário fazer a separação destas diferentes preocupações e posteriormente a organização dos pontos de variação dentro das mesmas.

Para a utilização de programação orientada a aspectos em um framework é necessário se transformar cada uma das preocupações

encontradas dentro de um componente ou aplicação em um aspecto, onde serão implementadas as variações dos hot spots encontrados.

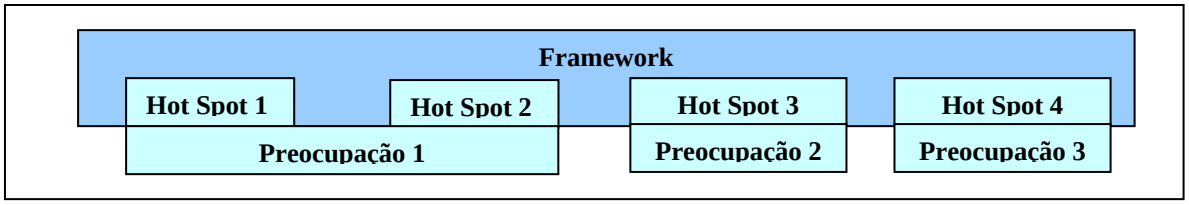


Figura 3-4: Hot Spots agrupados por preocupações

Aplicando o agrupamento por preocupações no exemplo proposto de extratos bancários para os hot spots, teremos as seguintes preocupações: cabeçalho, saldo e rodapé, que no momento de implementação terão suas variações implementadas nos aspectos cabeçalho, saldo e rodapé, respectivamente, conforme figura 3-5 apresentada abaixo.

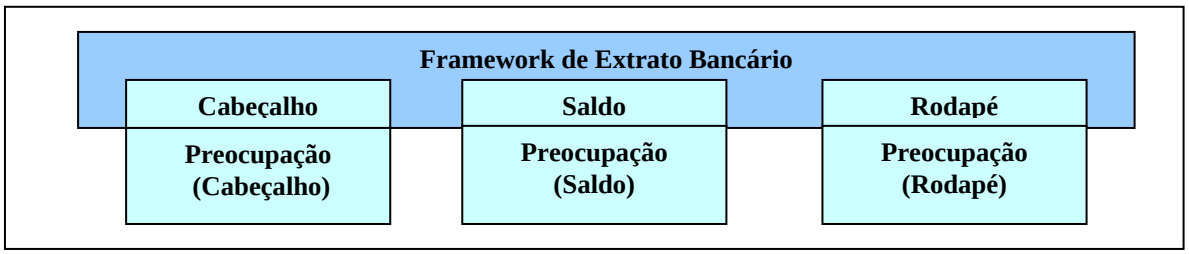


Figura 3-5: Hot Spots agrupados por preocupações em Componente de Software de extratos bancários.

No exemplo citado na figura acima, temos as preocupações relativas a cabeçalho, saldo e rodapé. Cada uma das preocupações por sua vez, tem um único ponto de variabilidade encontrado que deve ser implementado no framework de Extrato Bancário como hot spots e suas implementações devem ser implementadas dentro dos aspectos criados para cada uma das áreas de

preocupação, representadas na figura como Preocupação Cabeçalho, Preocupação Saldo e Preocupação Rodapé.

3.3.5. Associação dos Hot Spots com Aspectos

O presente trabalho propõe a utilização de frameworks como uma opção para a implementação de componentes que possuem pontos de variabilidade. Tendo em vista esta proposta, pode-se dizer que um hot spot representa um ponto de variabilidade em um componente conforme ilustrado pela figura3-6.

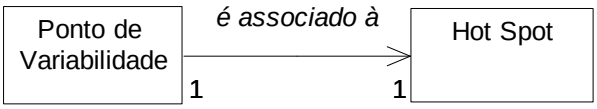


Figura 3-6: Representação da Implementação dos Pontos de Variabilidades por Hot Spots

Aplicando a associação dos Hot Spots com Aspectos no exemplo proposto de extratos bancários para os pontos de variação, teremos os seguintes hot spots: cabeçalho, saldo e rodapé, que no momento de implementação serão implementados, conforme figura 3-7 apresentada abaixo.

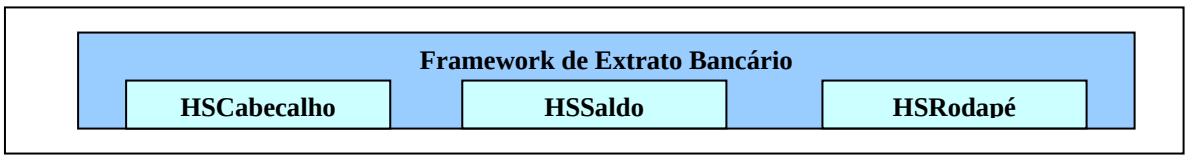


Figura 3-7: – Implementação dos pontos de variabilidades por Hot Spots em Componente de Software de extratos bancários.

3.3.5.1. Hot Spots x Join Points x Pointcuts

Tendo-se definido que os componentes serão implementados através de frameworks de componentes com a utilização de aspectos, tem-se a necessidades de relacionar o hot spot com o conceito de POA. Para que isto se torne possível, é necessário fazer a transformação dos hot spots em join point a serem interceptados por pointcuts, que é a maneira com a qual, uma aplicação POA implementa o conceito de ganchos, conforme ilustrado pela figura 3-8.

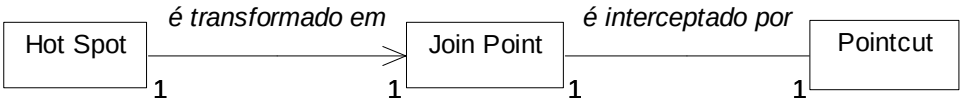


Figura 3-8: Representação dos Hot Spot como Join Points a serem Interceptados por Pointcuts

3.3.5.2. Padronização dos Hot Spots em Join Points

Tendo em vista que uma das propostas do trabalho é a facilitação da identificação dos hot spots e suas implementações, para a interceptação do mesmo, através da programação orientada a aspectos, deve-se utilizar um padrão para nome e identificação dos hot spots, fazendo assim a sua transformação em join points. O padrão utilizado no presente trabalho é a utilização de nomes com a seguinte sintaxe: *hsFunção*, onde *hs* refere-se a hot-spot e *Função* refere-se à função do ponto de variabilidade implementado em formato de hot spot.

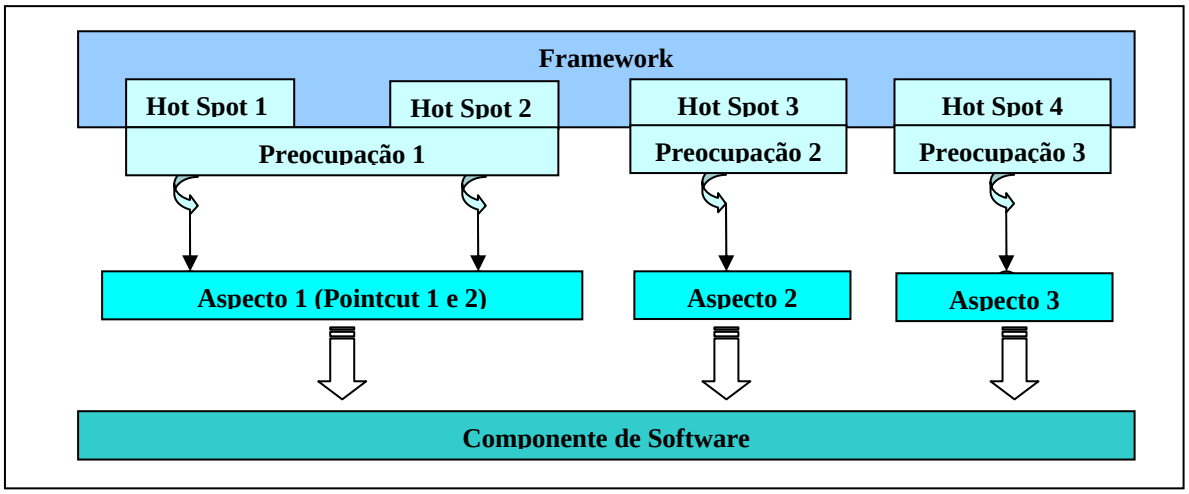


Figura 3-9: Hot Spots agrupados por preocupações sendo tratados como ganchos

No exemplo para componente de software de extratos bancários temos os hot spots sendo nomeados como: hsCabecalho, hsSaldo e hsRodape, conforme ilustrado na figura 3-10 abaixo.

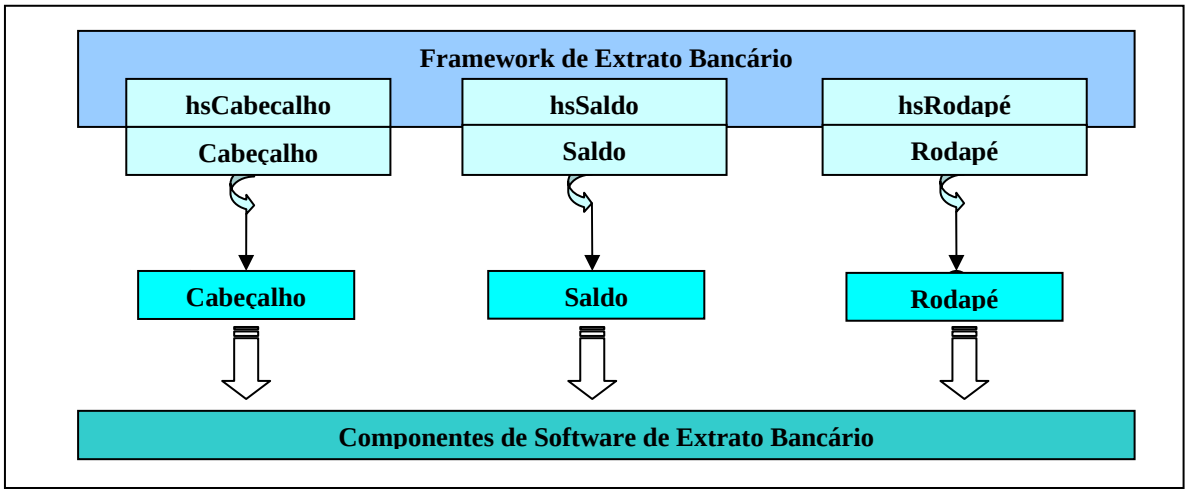


Figura 3-10: Hot Spots agrupados por preocupações sendo tratados como ganchos para Componentes de software de extratos bancários

3.3.5.3. Implementação dos Pontos de Variabilidades

O próximo passo a ser seguido é a implementação dos pontos de variabilidade dos componentes, que no presente trabalho é representado por

hot spots e interceptado por pointcuts. Na teoria de POA, têm-se duas maneiras de se implementar um pointcut: advice e introduction.

3.3.5.4. Advices ou Sugestões

Advice é uma das maneiras proposta pela POA para a implementação dos pointcuts, e se mostrou adequado para a implementação de pontos de variabilidades no modelo proposto.

A sugestão para a implementação de advices para pontos de variabilidades que representam características funcionais como as apresentadas neste presente trabalho é a utilização do padrão de projeto proposto por [GAM 00], Abstract Factory transformando o participante Abstract Factory em um gancho para o pointcut relacionado no aspecto e substituindo o participante Concrete Factory por um advice, conforme representado na figura 3-11.

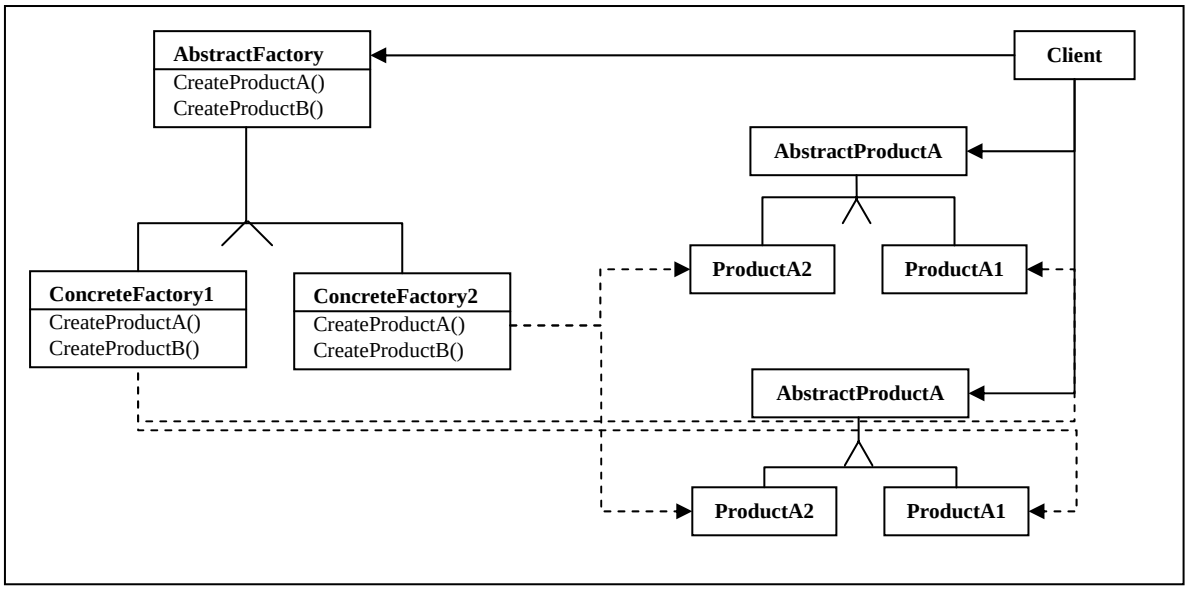


Figura 3-11: Padrão de Projeto AbstractFactory [GAM 00]

Como a intenção é fornecer uma interface para criação de famílias de objetos relacionados ou dependentes sem especificar suas classes concretas [GAM 00], este padrão se mostrou extremamente eficaz para a utilização onde sua aplicação requer um reuso de código por composição. Uma vez que a especificação das classes concretas só seriam implementadas nos advices e incorporadas ao código no processo de weaver. E as classes abstratas serviriam como ganchos ou joinpoints para a interceptação dos pointcuts a serem implementados.

3.3.5.5. Definição do Tempo de Execução das Sugestões

Outra definição importante para o funcionamento e aplicação do modelo proposto é a definição do tempo de execução das sugestões que devem ser definidos com o tempo de interceptação *around*, uma vez que ele permite a substituição da execução de métodos e manipulação dos valores dos argumentos interceptados.

A definição do tempo de execução das sugestões definida com o tempo de interceptação *around* se mostrou necessária, uma vez que a aplicação do padrão de projeto fábrica abstrata apresentou a necessidade de um controle total sobre as ações a serem realizadas no ponto de interceptação. E apenas este tempo de interceptação nos oferece um total controle sobre o ponto interceptado, podendo assim, descartar a declaração abstrata contida no método de criação da fábrica.

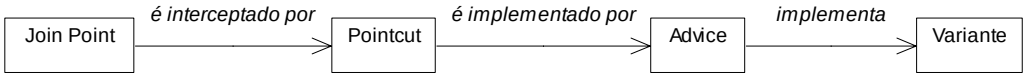


Figura 3-12: Representação dos Advices como alternativa de implementação de variantes.

A figura acima demonstra o relacionamento das sugestões (*advices*) como uma alternativa de implementação de variantes de um componente de software para reuso, onde temos os pontos de junção (*join points*) que conforme a sessão 3.2.5.2 do presente trabalho representa um hot spot e que ao ser interceptado por um pointcuts se comporta como um gancho, se adequando perfeitamente a utilização proposta deste trabalho.

3.2.6. Relacionamentos entre Hot Spots e Aspectos

Dos estudos realizados conclui-se que para a criação de modelos para realizar o relacionamento entre hot spots e aspectos é necessário relacionar as entidades Hot Spot com Pointcut, Pointcut com Advice e Advice com variação (implementação do ponto de variabilidade).

3.2.6.1. Combinações Possíveis entre Hot Spots, Pointcuts, Advices e Variação.

A seguir apresentamos as combinações possíveis entre Hot Spots, Pointcuts, Advices e Variações.

As combinações possíveis entre estas entidades geram 16 combinações possíveis que são mostradas na tabela 3-1 abaixo:

Combinação	Relacionamento entre Hot spots e Pointcuts	Relacionamento entre Pointcuts e Advices	Relacionamento entre Advices e Variações
1	1 para 1	1 para 1	1 para 1
2	1 para 1	1 para n	1 para n
3	1 para 1	1 para 1	1 para n
4	1 para 1	1 para n	1 para 1
5	1 para n	1 para 1	1 para 1
6	1 para n	1 para 1	1 para n
7	1 para n	1 para n	1 para 1
8	1 para n	1 para n	1 para n
9	N para 1	1 para 1	1 para 1
10	N para 1	1 para 1	1 para n
11	N para 1	1 para n	1 para 1
12	N para 1	1 para n	1 para n
13	N para n	1 para 1	1 para 1
14	N para n	1 para 1	1 para n
15	N para n	1 para n	1 para 1
16	N para n	1 para n	1 para n

Tabela 3-1: Combinações possíveis entre Hot Spots, Pointcuts, Advices e Variações.

3.2.6.2. Eliminação das Combinações inválidas para a criação dos modelos

Dada as combinações apresentadas no item acima, o próximo passo a ser seguido, é eliminar quais combinações que não são válidas para criação de modelos que se encaixe na solução do problema proposto.

3.2.6.2. 1. Eliminação da Combinação 3 (três)

Eliminou-se a combinação 3 (três) uma vez que a implementação desta combinação se mostra inviável, uma vez que o relacionamento entre pointcut, advice e variação possui um relacionamento n para n, criando assim a

possibilidade de perda de controle de fluxo e consequentemente a possibilidade de não se conseguir atingir os objetivos propostos.

3.2.6.2. 2. Eliminação das Combinações 5 (cinco) a 8 (oito)

Eliminaram-se as combinações 5 (cinco) a 8 (oito) uma vez que a implementação destas combinações se mostra inviável, pois se tem em um único ponto de interceptação a possibilidade de inclusão de n implementações, podendo assim, ter a perca de controle de fluxo e não se conseguindo atingir os objetivos propostos.

3.2.6.2. 3. Eliminação das Combinações 9 (nove) a 16 (dezesseis)

Eliminaram-se as combinações 9 (nove) a 16 (dezesseis) uma vez que não se consegue mapear n hot spots para um único pointcut, fugindo assim do objetivo inicial do modelo proposto, apesar de ser viável para outros problemas modelados pela Programação Orientada a Aspectos.

3.2.6.3. Criação de Modelos com as Combinações Adequadas

Tendo-se eliminado as combinações não adequadas para a solução do problema proposto, realizou-se a criação dos modelos com as combinações que foram consideradas válidas e chegou-se aos seguintes modelos:

- **Modelo I:** 1 Hot Spot está relacionado a 1 Pointcut, 1 pointcut está relacionado a 1 advice e 1 advice esta relacionado a 1 variação;

- **Modelo II:** 1 Hot Spot está relacionado a 1 Pointcut, 1 pointcut está relacionado a n advice e 1 advice esta relacionado a 1 variação;

- **Modelo III:** 1 Hot Spot está relacionado a 1 Pointcut, 1 pointcut está relacionado a 1 advice e 1 advice esta relacionado a n variação;

3.2.6.3. 1. Modelo I - 1 HS x 1 PC, 1 PC x 1 Ad e 1 Ad x 1 V

Este modelo é criado a partir da primeira combinação proposta entre Hot Spots, Pointcuts, Advices e Variações e é representado pela figura abaixo:

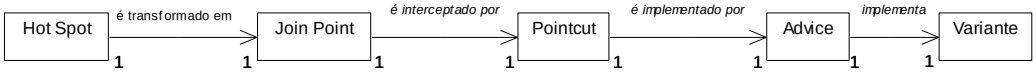


Figura 3-13: Modelo Proposto I

A desvantagem deste modelo é a necessidade de se ter um advice para cada hot spot, tendo assim, para cada componente, a necessidade de implementação de um novo aspecto para cada implementação dos pontos de variação dentro de um mesmo grupo de preocupações, uma vez que não se pode ter mais de uma implementação para cada ponto de interceptação.

Como se tem um único aspecto para um grupo de hot spots corre-se o risco de se ter pointcuts idênticos em diferentes aspectos, tendo assim, a repetição de código em diferentes lugares, gerando um retrabalho ou uma dificuldade no gerenciamento e manutenção dos mesmos.

Este modelo se mostrou capaz de implementar os tipos de variabilidades "*variação opcional*" e "*uma variação entre várias alternativas*", tendo como limitação a implementação do tipo de variabilidade "*uma combinação de variações entre várias alternativas*".

3.2.6.3. 2. Modelo II - 1 HS x 1 PC, 1 PC x n Ad e 1 Ad x 1 V

Este modelo é criado a partir da segunda combinação proposta entre Hot Spots, Pointcuts, Advices e Variações e é representado pela figura abaixo:

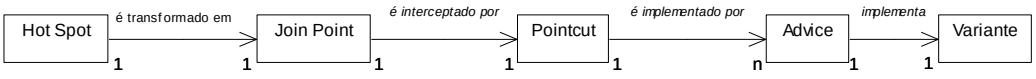


Figura 3-14: Modelo Proposto II

Este modelo resolve o problema gerado pela necessidade de implementação de um novo aspecto para cada implementação dos pontos de variação dentro de um mesmo grupo de preocupações presente no modelo I, apresentado no item anterior, eliminando assim o risco de se ter pointcuts idênticos em diferentes aspectos que poderia gerar a repetição de código em diferentes lugares.

A implementação deste modelo apresenta a necessidade de criação de um pacote de controle, que é composto por classes e um aspecto, que utilizam o conceito de reflexão computacional que observa e faz as modificações necessárias de propriedades relativas ao seu comportamento. Também apresentou a necessidade de criação de um pacote de configuração que é composto por um arquivo XML de opções de alteração de comportamento e classes que faz a busca de comportamentos contidas no arquivo XML citado.

Com a utilização do conceito de reflexão, tem-se a necessidade de se explicitar na implementação do framework quais serão os prováveis comportamentos representados pelas variações.

Este modelo mantém a limitação do modelo anterior de se implementar o tipo de variabilidade "*uma combinação de variações entre várias alternativas*".

3.2.6.3. 3. Modelo III - 1 HS x 1 PC, 1 PC x 1 Ad e 1 Ad x n V

Este modelo é criado a partir da segunda combinação proposta entre Hot Spots, Pointcuts, Advices e Variações e é representado pela figura abaixo:

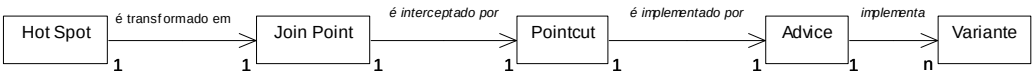


Figura 3-15: Modelo Proposto III

Este modelo manteve a necessidade da criação e implementação de um pacote de configuração composta por um arquivo XML de opções de alteração de comportamento e classes que fazem a leitura e identificação de comportamentos contidas no arquivo XML citado.

E eliminou a necessidade da criação do pacote de controle que faz a observação e as modificações necessárias de propriedades relativas ao seu comportamento.

Este modelo não conseguiu eliminar a limitação dos dois modelos anteriores de se implementar o tipo de variabilidade "*uma combinação de variações entre várias alternativas*", criando assim a limitação deste tipo de implementação com o mecanismo proposto pelo presente trabalho.

3.2.6.4. Comparativo entre os Modelos Propostos

Os modelos apresentados devem ser aplicados em um componente de software, que apresenta várias implementações possíveis para um único ponto de variabilidade.

Quando se escolhe qual o melhor modelo a ser utilizado em determinado problema, tem-se que levar em conta uma série de características para se tomar uma decisão. Entre estas características, está o fato do modelo I apresentar a existência de apenas uma implementação para cada hot spot, criando assim o risco da existência de repetição de códigos em diferentes componentes.

O modelo II apresenta a necessidade de utilização do conceito de reflexão computacional, que é um recurso relativamente lento e que só deve-se utilizar quando não for possível atingir o objetivo proposto com outro recurso. Além de apresentar limitações em relação à quantidade de variações que podem ser implementadas em cada um dos pontos de variabilidade, uma vez que existe a necessidade de se explicitar quais serão as prováveis variações tanto no FCOA como no aspecto que implemente a preocupação relacionada ao ponto de variação.

O modelo III é o modelo mais indicado para a utilização no modelo de utilização para implementação de Frameworks de Componentes Orientado a Aspectos proposto no próximo capitulo do presente trabalho; uma vez que o

mesmo consegue suprir; a deficiência do Modelo I relativo ao risco da existência de repetição de códigos em diferentes componentes, os problemas de lentidão que podem ser causados pelo uso de reflexão computacional do Modelo II e não possui as limitações em relação à quantidade de variações que podem ser implementadas, também presentes no Modelo II.

3.2.6.5. Limitação dos Modelos Propostos

A limitação relativa à implementação do tipo de variabilidade "*uma combinação de variações entre várias alternativas*" é causada devido ao fato de se utilizar como ponto de interceptação a criação de uma fabrica abstrata, onde tem-se a implementação de uma fábrica concreta para cada alternativa de variação para um ponto de variabilidade.

Para a implementação do tipo de variabilidade, uma combinação de variações entre várias alternativas, seria necessária a utilização de diferentes fábricas concretas por uma única fábrica abstrata.

3.2.7. Modelo Lógico do Mecanismo de Implementação de Variante em Componentes com utilização de POA

O modelo proposto pelo presente trabalho é a implementação de componentes de software através de frameworks orientados aspectos da categoria FAOA, que como sugestão deve-se chamar de Framework de

Componentes Orientado a Aspectos (FCOA) e que é representado pelo modelo lógico de implementação de variante em componentes.

Baseado no que foi abordado nas seções anteriores, propomos o modelo visto na figura 3-15 para integrar Frameworks de Aplicação com componentes e aspectos.

Na figura 3-16 abaixo se tem representado o modelo lógico de implementação de componentes onde se representa a junção dos três mundos envolvidos no problema que são componentes, frameworks e aspectos e foi gerada pela junção dos modelos representados nas sessões anteriores do presente capítulo.

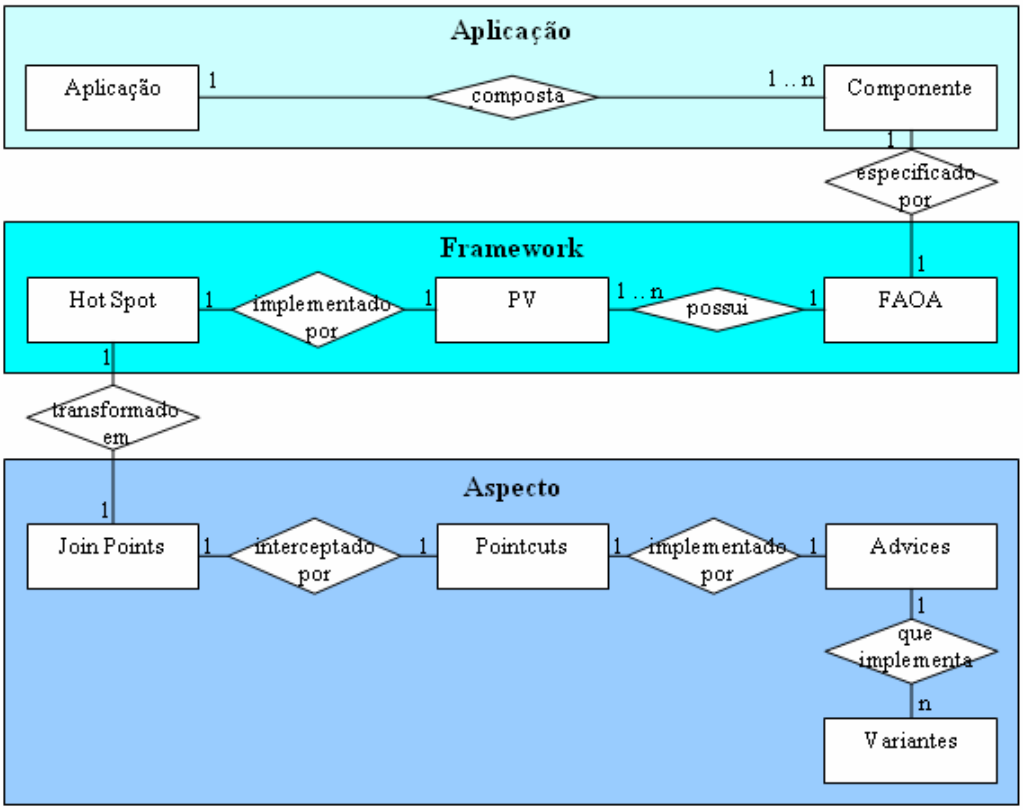


Figura 3-16: Modelo Lógico do Mecanismo de implementação de variante em componentes com a utilização de POA.

3.2.8. Pacote de Configuração dos Modelos II e III

Os modelos II e III criados apresentam a necessidade de um pacote de configuração que deve ser criado contendo as classes responsáveis pela leitura da configuração dos componentes que são armazenadas em um arquivo de configuração XML que deverá ser lido por uma ferramenta de configuração de componentes.

O arquivo XML de configuração utilizado deve seguir o padrão apresentado na figura 3-17 abaixo.

```
<?xml version="1.0"?>
    <DOCUMENTO>
        <IMPLEMENTACAO cont_id="3">
            <VERSAO cont_id="5" flg_ativo="0" id="1">
                <NOME>Sistema 1</NOME>
                <DESCRICAO>Sistema 1 - Lista de Discursao</DESCRICAO>
                <PV id="1">
                    <CLASSE>FactoryTipoRegistro</CLASSE>
                    <METODO>createHSTipoRegistro()</METODO>
                    <VARIACAO>registerMailSubscribtion()</VARIACAO>
                </PV>
            </VERSAO>
        </IMPLEMENTACAO>
        <CONFIGURACAO cont_id="7">
            <PV id="1">
                <CLASSE>FactoryTipoRegistro</CLASSE>
                <METODO>createHSTipoRegistro()</METODO>
                <VARIACAO>registerMailSubscribtion()</VARIACAO>
                <VARIACAO>registerWWWSubscription()</VARIACAO>
            </PV>
        </CONFIGURACAO>
    </DOCUMENTO>
```

Figura 3-17: Arquivo XML de Configuração

Na figura acima, a *tag* <*implementacao*> representa quais são as implementações possíveis dos componentes. A *tag* <*versao*> identifica uma

versão do componente, onde o atributo *flg_ativo* seta se esta versão esta ativa ou não, quando se tem o atributo *flg_ativo* com o valor 1, quer dizer que esta é a versão a ser utilizada para a configuração dos componentes da aplicação em questão.

A *tag* <*classe*> determina a qual classe pertence o método a ser interceptado pelo aspecto de configuração do componente, interceptando o método indicado pela *tag* <*metodo*> e substituindo a sua chamada pela execução do advice representando pela *tag* <*variacao*>.

No capítulo 4 do presente trabalho apresenta-se um modelo de implementação e utilização de um Framework de Componentes Orientado a Aspectos seguindo o modelo lógico proposto no presente capitulo e no capitulo 5 do presente trabalho apresentam-se casos de uso implementação e utilização de Frameworks de Componentes Orientado a Aspectos (FCOA) seguindo os modelos propostos nos capítulos 3 e 4.

4. Modelo de Utilização para Implementação de Frameworks de Componentes Orientado a Aspectos Baseado em Variabilidade de Caso de Uso

Apresenta-se neste capítulo o modelo denominado Modelo de Utilização para Implementação de Frameworks de Componentes Orientados a Aspectos Baseados em Variabilidade de Caso de Uso. Ele consiste em um modelo de alto nível de reuso de software através da utilização de aspectos funcionais na construção de frameworks de componentes orientados a aspectos (FCOA).

Neste capítulo mostram-se quais os passos devem-se seguir e quais os participantes necessários para a utilização de um modelo de alto nível de reuso de software. Parte-se das etapas iniciais de análise de software até a implementação de frameworks de componentes orientados a aspectos (FCOA) que utiliza o modelo lógico proposto no capítulo anterior para o desenvolvimento de componentes de software.

4.1. O Modelo Proposto

Como se abordou anteriormente, um componente possui uma estrutura comum (arquitetura) com pontos bem definidos que determinam onde se encontram os pontos de variabilidade. Posto de outra forma, caso se deseje gerar uma nova versão do componente basta acessar estes pontos e instanciá-los de forma a criar um novo componente.

Embora conceitualmente simples, é difícil implementar a proposta. O problema para se realizar esta atividade está justamente em encontrar estes pontos de variabilidade no código fonte e entender o que eles significam para realizar as alterações necessárias durante a implementação do componente.

Em busca da solução, percebeu-se que quando se projeta o framework, pode-se identificar e modelar os pontos de variabilidade (hot spots), associando informação sobre o seu significado. A programação orientada a aspectos nos ajuda também neste problema. Ela nos fornece recursos para interceptar pontos bem determinados do programa (joinpoints ou ganchos), no nosso caso os hot spots, e nos fornece o código que será inserido nesta posição (pointcut e advice), no nosso caso, a variabilidade.

O modelo proposto preocupa-se com a construção de dois tipos de arquiteturas distintas: Framework de Aplicação Orientado a Objetos (FOO) e Framework de Componentes Orientado a Aspectos (FCOA).

O Framework Orientando a Objetos (FOO), deve ser projetado e implementado de maneira tradicional, utilizando preferencialmente a metodologia e notação proposta por [FON 99], representando uma família de aplicação.

O FCOA deve ser deve ser projetado e implementado utilizando os conceitos de Framework de Aplicação Orientado a Aspectos (FAOA) e

representa um conjunto de componentes que se diferenciam pelas implementações das alternativas dos seus pontos de variabilidades.

No presente capítulo será discutida uma maneira de projetá-lo, implementá-lo e adaptá-lo aos conceitos necessários para o desenvolvimento de componentes de software.

A seguir, apresentamos a figura 4-1 que ilustra os participantes, etapas e processos presentes no modelo lógico.

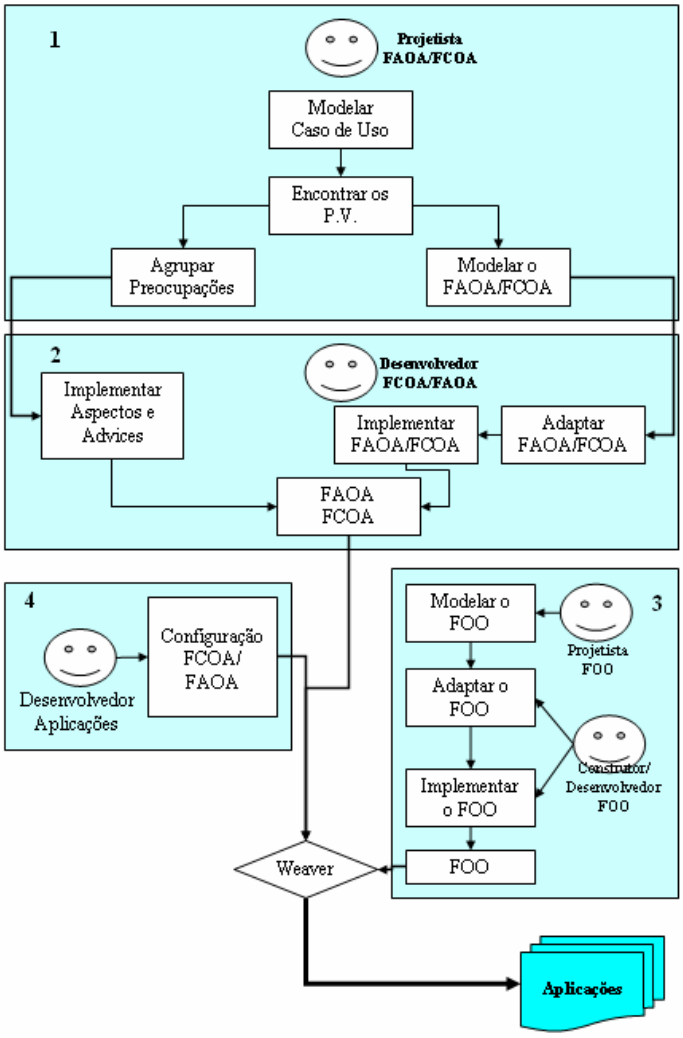


Figura 4-1: Modelo Lógico de Utilização para Implementação de Frameworks de Componentes Orientados a Aspectos Baseados em Variabilidade de Caso de Uso.

A figura 4-1 é composta por quatro etapas fundamentais no processo de Utilização para Implementação de Frameworks de Componentes Orientados a Aspectos, que são:

1. Projeto do FCOA/FAOA.

2. Desenvolvimento do FCOA/FAOA.

3. Projeto e Desenvolvimento do FOO.

4. Desenvolvimento de Aplicações.

A 1ª Etapa do Modelo Proposto é o Projeto e desenvolvimento do FCOA/FAOA e é onde ocorre o projeto da arquitetura do componente a ser implementada utilizando os conceitos de FAOA, transformando-o em um FCOA.

O principal participante desta etapa é o Projetista de FAOA/FCOA que é responsável pela modelagem dos Casos de Uso, e a detecção dos pontos de variabilidade existentes nos mesmos, identificando não só os pontos de variabilidade e suas variações.

Após a modelagem dos casos de uso e seus pontos de variabilidade, o Projetista de FAOA/FCOA fica responsável por agrupar os pontos de variabilidade por preocupações, pela padronização dos nomes dos pontos de

variabilidade a serem implementados como hot spots e pela modelagem da arquitetura dos componentes em forma de um Framework de Componentes Orientado a Aspectos (FCOA).

Nesta Etapa temos como produto final, o modelo do FCOA a ser implementado, bem como os Grupos de Preocupações necessários para a implementação das Variabilidades encontradas.

A 2ª Etapa do Modelo Proposto é o Desenvolvimento do FCOA/FAOA que é onde ocorre a implementação do Framework de Componentes Orientado a Aspectos (FCOA), bem como a implementação dos aspectos e advices a serem utilizados pelo mesmo.

O principal participante desta etapa é o Construtor/Desenvolvedor de FCOA/FAOA que é o responsável pela adaptação e implementação do FCOA. O desenvolvedor deve seguir os padrões gerados para nomes de hot spots criados na Primeira Etapa, para que ele seja viável de ser implementado na arquitetura escolhida.

Para a implementação dos aspectos e sugestões deve-se seguir o agrupamento de preocupações realizado na primeira etapa, transformando cada preocupação em um aspecto e transformando os pontos de variabilidade em sugestões que serão os responsáveis pela implementação dos mesmos.

Nesta etapa, temos como produto final os Frameworks de Componentes Orientado a Aspectos que é composto pelo framework e pelos Aspectos e Advices necessários para a sua instanciação.

A 3ª Etapa do Modelo Proposto é o Projeto e Desenvolvimento do Framework Orientado a Aspectos (FOO) que deve ser composto pelos Frameworks de Componentes necessários criados na Segunda Etapa.

Para o projeto e desenvolvimento do framework utilizam-se as metodologias tradicionais que o presente trabalho não detalha, deixando como sugestão a fonte [FON 99].

Os principais participantes desta etapa são o Projetista de FOO e o Construtor/Desenvolvedor de FOO.

O produto final a ser gerado por esta etapa é o projeto e a implementação do Framework de Aplicação Orientado a Objetos (FOO).

A 4ª Etapa do modelo proposto é o desenvolvimento de aplicações onde temos a configuração do FOO, gerado na terceira etapa, conforme a aplicação que se quer gerar.

O participante desta etapa é o desenvolvedor de componentes, que será responsável pela configuração dos FCOAs a partir de um arquivo de configuração de componentes e aplicação em XML que contém as

configurações de implementação dos pontos de variabilidade a serem utilizados pelo weaver no momento de geração dos componentes e da aplicação.

Nesta etapa deve-se primeiramente selecionar quais são os aspectos/advices a serem utilizados por cada um dos FCOAS implementados na Segunda Etapa e passá-los pelo processo de weaving para a criação dos componentes a serem compostos ao FOO para a criação de novas Aplicações.

A seguir detalhamos cada atividade citada acima.

4.2. Modelar Casos de Uso e Encontrar Pontos de Variabilidade

Segundo [JAC 97] para se reusar componentes, deve-se expressar os casos de uso específicos e característicos da aplicação nos termos mais genéricos possíveis e para garantir o desenvolvimento baseando em componentes em todos os estágios, do levantamento de requisitos à implementação, é crítico identificar os componentes de reuso mais prováveis logo no início do processo de desenvolvimento.

Para alcançar o objetivo de aplicação de reuso de alto nível, e ter como resultado componentes com alto grau de reuso, deve-se aplicar os conceitos de reuso desde os momentos iniciais do projeto de software. Seguindo este

raciocínio, propõe-se a utilização das técnicas e conceitos de reuso desde o momento de desenvolvimento de casos de uso.

A maneira proposta para se aplicar estes conceitos de reuso no momento de desenvolvimento de casos de usos, é a utilização do conceito de generalização para o desenvolvimento de casos de uso, criando assim, casos de usos preparados e pensados para reuso.

Dado que um caso de uso possui pré-condições, pós-condições e episódios e que estes episódios são compostos por um ou mais passos. Analisando-se os passos que compõem um determinado episódio em diferentes membros de uma mesma família de componentes, nota-se que estes passos podem conter pontos de variação e que estes pontos de variação são compostos por um ou mais alternativas ou conjunto de alternativas de resolução que podem ser representados como variações para um mesmo ponto de variação.

Analisando-se os casos de uso referentes a um componente de software, nota-se a possibilidade de construção de um caso de uso que se pode chamar de caso de uso padrão ou genérico. Este caso de uso conterá pontos de variação que serão instanciados pelas variantes, gerando diferentes componentes de uma mesma família.

Segundo [LEE 01] um mesmo episódio geral poderá ser realizado em vários casos de uso, mas existem detalhes significativamente diferentes

dependentes das entidades que neles participam. Mesmo que as generalizações tenham sido identificadas em um estágio anterior a este ponto, ainda será uma boa idéia examinar os casos de uso para determinar se podem ser acrescentadas novas generalizações.

Com isto podemos representar o caso de uso conforme a figura abaixo:

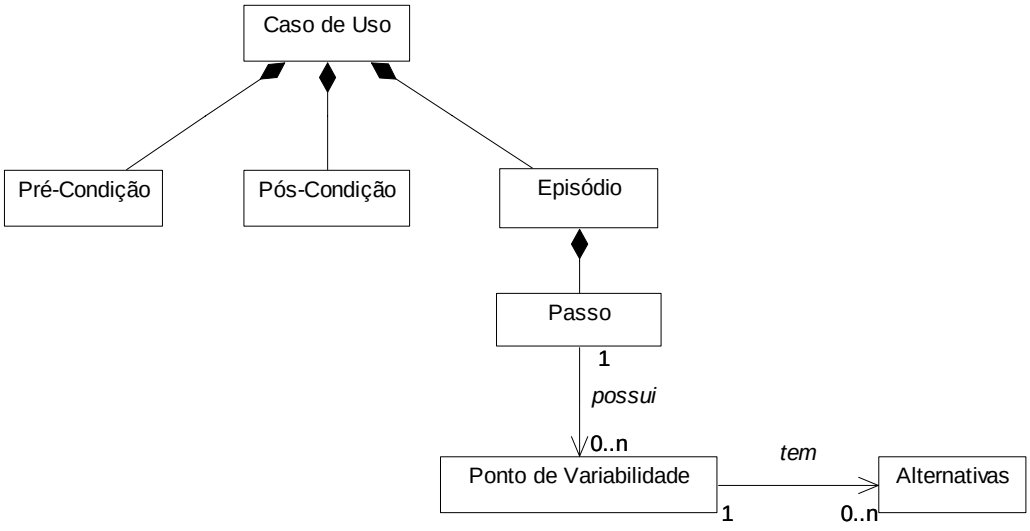


Figura 4-2: Modelo de Variação em Caso de Uso

Apesar de [LEE 01] não ter pensado em sua aplicação na construção de um framework para um componente de software, é um problema que pode ser eficientemente resolvido pela utilização desta técnica e que mostra de forma clara a utilização de generalização em casos de uso.

No exemplo dado por [LEE 01] um *Daemon* de Eventos Assíncronos faz disparar o caso de uso "*Procurar condições de trabalho*". Esse caso de uso inclui o caso de uso "*Exame do Ambiente*", que é uma generalização para os casos de uso "*Verificar Grama*" e "*Verificar o Carpete*". O caso de uso "*Procurar*

por Condições de Trabalho” também inclui o caso de uso “*Planejar e Atribuir Trabalhos*”, que é uma generalização para os casos de uso “*Planejar Cortar a Grama*” e “*Planejar Aspirar o Pó do Carpete*”. Conforme ilustra a figura abaixo:

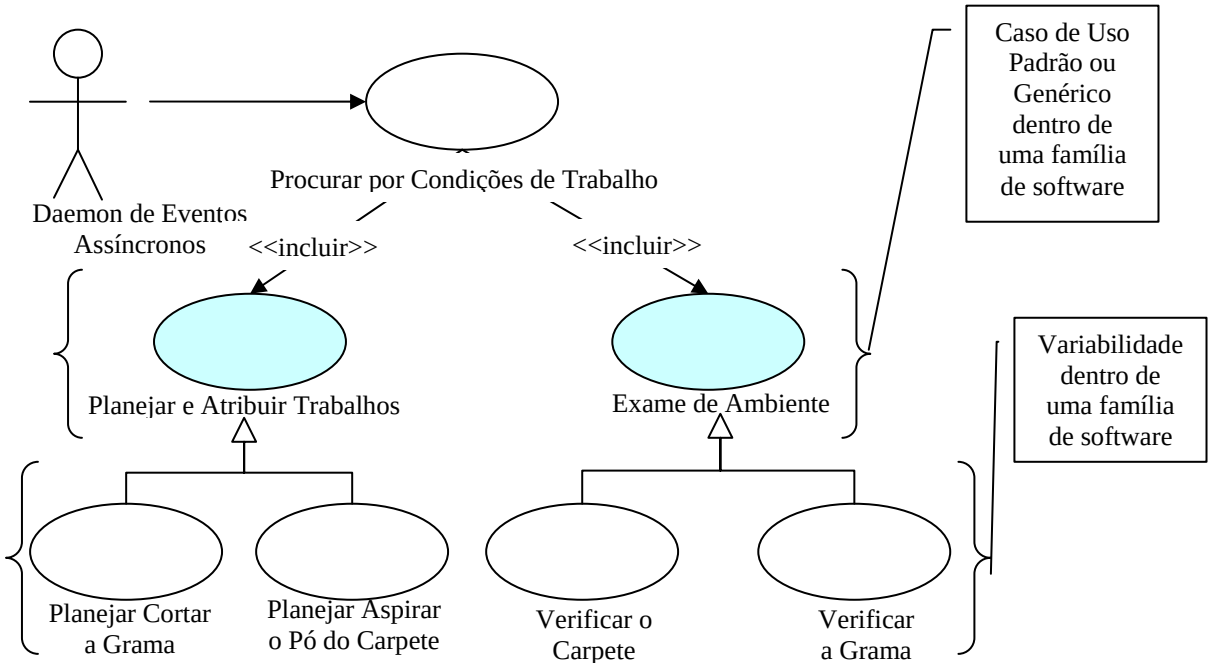


Figura 4-3: Diagrama de caso de uso com generalização. Adaptado de [LEE 01]

Se aplicarmos no exemplo de caso de uso mostrado na figura acima o modelo proposto, podemos classificar como caso de uso genérico os casos de uso “*Exame de Ambiente*” e “*Planejar e Atribuir Trabalhos*” que podem ser implementados como componentes. O episódio “*Exame de Ambiente*” possui as alternativas “*Verificar Carpetes*” e “*Verificar Grama*” que pode ser considerado pontos de variação. E o episódio “*Planejar e Atribuir Trabalhos*” possui as alternativas “*Planejar Cortar a Grama*” e “*Planejar Aspirar o Pó do Carpete*”.

4.3. Modelar o Framework de Componentes Orientado a Aspectos

Segundo [FON 99] na fase de modelagem de um framework a estrutura do mesmo é definida pelo projetista de estrutura, que no modelo proposto esta sendo chamado de Projetista de FAOA/FCOA, que usa informações geradas na fase de análise de requisitos do domínio (conhecimento do domínio) e a representa com a linguagem de projeto da estrutura. Além disto, para a modelagem no modelo proposto, deve ter conhecimentos também de agrupamento de preocupações e aspectos, conforme representado na figura 4-4.

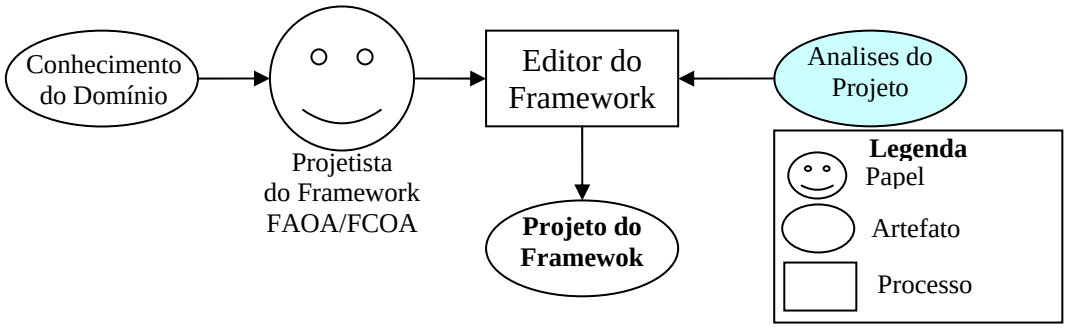


Figura 4-4: Estrutura do Editor de Framework Adaptado de [FON 99]

Para o desenvolvimento de um framework deve-se, primeiramente, desenvolver várias aplicações e ou componentes de uma mesma família de software para que se possa gerar o conhecimento do domínio. Tendo-se gerado este conhecimento do domínio, deve-se utilizá-lo para se projetar o framework, uma vez que já foi adquirido o conhecimento necessário.

A proposta deste trabalho sugere que sejam utilizados para registrar o conhecimento do domínio, os casos de usos gerados para os componentes de uma mesma família de software.

Tendo-se analisado os casos de usos de diferentes aplicações de uma mesma família de software, deve-se aplicar a técnica de generalização proposta por [LEE 01], que foi apresentada na seção anterior deste capítulo, chamado de Caso de Uso Padrão ou Genérico de um componente.

Os pontos onde ocorrerem as generalizações deverão ser transformados em hot spots no momento de projeto do framework, uma vez que as generalizações representam as variabilidades dentro de um componente de software e o caso de uso padrão representa exatamente a estrutura do framework de componente de software em questão.

Após o desenvolvimento do Caso de Uso Padrão ou Genérico, deve-se fazer a etapa de agrupamento de preocupação.

A notação utilizada para modelar o framework é a proposta por [FON 99]. Na tabela 4.1 temos a lista de estereótipos proposta por [FON 99] e na figura 4.5 temos um exemplo de uso desta notação. Nela pode-se ver que tem-se um Hot spot em *actor* pois pode-se criar novos papéis e por este motivo a especialização esta sendo rotulada *Incomplete*.

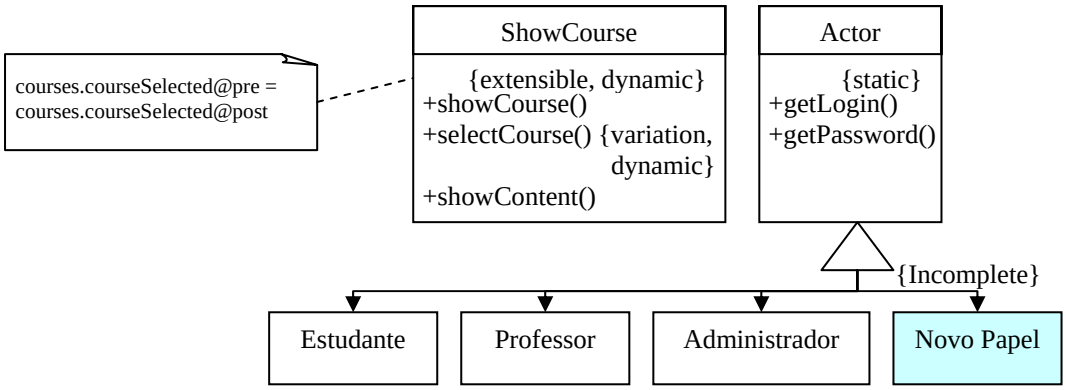


Figura 4-5: Diagrama de Classes Estendido [FON 99]

Seguindo o sumário de novos elementos apresentado por [FON 99] na tabela 4-1 abaixo:

Elemento	Tipo	Aplicação	Semântica
Instance Class	Estereotipo	Classes	Classe existe apenas se o framework estiver instanciado. Uma nova classe instanciada é definida durante a instanciação do framework.
Variation	Etiqueta de Valor	Métodos	Significa que o método de implementação depende da instancialização do framework. É usado para identificar métodos de variação hot spot.
Extensible	Etiqueta de Valor	Classe	Significa que a classe de interface depende da instancialização do framework: novos métodos podem ser definidos para estender a funcionalidade da classe. Esses elementos identificam extensões de classes hot spots.
Static	Etiqueta de Valor	Extensão de Interface, Método de Variação, Extensão de Clases.	Isso significa que o hot spot não precisa de reconfiguração do run-time.
Dynamic	Etiqueta de Valor	Extensão de Interface, Método de Variação, Extensão de Clases.	Isso significa que o hot spot precisa da reconfiguração de run-time.

Incomplete	Restrições	Generalização e realização.	Tem quase o mesmo significado que em uma UML padrão. Incomplete significa que as novas classes satisfazem um relacionamento dado (generalização ou realização). Podem ser adicionadas durante a instanciação do framework, identificando extensões da interface hot spot.
Restricted	Restrições	Generalizações incompletas e realizações incompletas.	Significa que a classe instanciada criada pelo relacionamento não deve estender esta interface. Este elemento identifica extensões restritas da interface.

Tabela 4-1: Sumário de novos elementos e seus significados [FON 99]

4.4. Adaptar o Framework

A etapa de adaptação do framework para a utilização de aspectos como mecanismo de variabilidade de aspectos funcionais em componentes consiste em utilizar a padronização de nomes proposta no capítulo 3 do presente trabalho.

Esta padronização se mostra necessária, para que se possa transformar os hot spots do framework que esta sendo desenvolvido em join points, tornando-se possível a sua interceptação por um pointcut correspondente e assim, utilizando o mecanismo de reuso através de aspectos proposto.

A padronização de nomes de hot spots consiste em transformar os nomes dos hot spots seguindo o padrão hsFuncao, onde hs indica ser um hot spots e Função é a função ou nome do hot spot dentro da família de software proposta. Com esta padronização, estamos transformando os hot spots em join points e consequentemente transformando-os em ganchos a serem

interceptados pelos pointcuts a serem implementados nos aspectos que farão parte do repositório de aspectos reutilizáveis.

Feita a padronização de nomes dos hot spots, deve-se utilizar os perfis UML-AOD proposto por [CAM e MAS 04] e UML-AOF proposto por [CAM 04] para se modelar o framework de componentes orientado a aspectos, estendendo o framework modelado anteriormente.

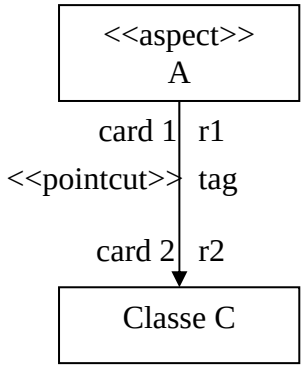


Figura 4-6: Notação para Programas Orientados a Aspectos

A figura 4-6 representa um exemplo de elementos utilizando a notação proposta por [PAW 02]. A representação de um aspecto é realizada pelo estereótipo <<*aspect*>>. A relação unidirecional do aspecto para a classe C, que indica que este aspecto intercepta e interrompe a execução de um método da classe em algum ponto relevante da execução e executa algum código antes ou depois da execução da classe é representado pelo estereótipo <<*pointcut*>>. O nome do método do aspecto definido no aspecto A que afeta algum ponto dos componentes definido pelo papel *"r2"* é representado pelo papel *"r1"*. Um conjunto de pontos de junção no formato T[expressão], onde T pertence ao conjunto descrito na tabela 4-2 e a expressão representa algum

ponto no programa como, por exemplo, a assinatura de algum método, o nome de uma classe ou mesmo alguns elementos da tabela 4-3 é representado pelo rótulo *"r2"*. O número de instâncias, tanto dos aspectos quanto das classes é representado pelas cardinalidades *"card1"* e *"card2"*. Para estender a semântica do relacionamento é utilizada uma etiqueta valorada *"tag"*.

Elemento	Exemplo	Descrição
!¹	!(ClassName.m())	Denota a invocação de um método chamado m() da classe ClassName.
?	?(ClassName.m())	Denota a execução de um método chamado m() da classe ClassName.
<N>	<N> (ClassName)	Denota a criação de todas as instâncias da classe ClassName.
<U>	<U> (ClassName)	Denota todos os pontos do programa onde uma instância da classe ClassName é usada pela primeira vez.
<C>	<C> (ClassName)	Denota todos os pontos do programa em que alguma instância da classe ClassName é clonada.
<R>	<R> (ClassName)	Denota os pontos do programa em que uma instância da classe ClassName foi remotamente criada.
<E>	<E> (ClassNameException)	Denota todos os pontos do programa em que a exceção ClassNameException foi lançada.

Tabela 4-2: Elementos Notacionais para Pontos de Junção [PAW 02]

ALL	Todos os métodos de C
CONSTRUCTORS	Todos os construtores de C
SETTERS	Todos os métodos "set" de C
GETTERS	Todos os métodos "get" de C
SETTER (att)	O método "set" de algum atributo
GETTER (att)	----
ACCESSOR	Todos os métodos de C que lê o estado de algumas instâncias.
ACCESSOR ({att})	Todos os métodos de C que lêem os atributos contidos na lista "att"
MODIFIERS	Todos os métodos de C que modificam o estado de instâncias
MODIFIERS ({att})	Todos os métodos de C que modificam os atributos contidos na lista "att"

Tabela 4-3: Palavras Chave para a Notação [PAW 02]

Os estereótipos, suas devidas classes e suas etiquetas criados por [CAM e MAS 04] são apresentados divididos por tipo na tabela 4-5.

Estereótipo	Classes base da UML	Etiquetas
<<aspect>>	Classifier	Não tem
<<introduction>>	Association	attribute method
<<includes_extends>>	Association	PCDescriptor joinPoint
<<includes_implements>>	Association	interface
<<crosscutting>>	Association	parent
<<pointcut>>	Operation	Não tem
<<before>>	Operation	Não tem
<<after>>	Operation	Não tem
<<after_returning>>	Operation	Não tem
<<after_throwing>>	Operation	Não tem
<<around>>	Operation	Não tem

Tabela 4-4: Estereótipos do Perfil UML-AOD [CAM e MAS 04]

Na tabela 4-4 os estereótipos do perfil UML-AOD são mostrados divididos por tipo. Há um estereótipo de classe, quatro de associação e seis de operação.

Abaixo seguem os estereótipos do perfil UML da UML-AOF proposta por [CAM 04] para a aplicação em frameworks orientados a aspectos.

Estereótipo	Restrições	Semântica
<<aspect-application>> ou <<aspect-app>>	Este tipo de estereótipo só pode ser aplicado a elementos do tipo Classe Classes da aplicação abstratas devem ter seu nome representado em itálico.	Denota que o elemento rotulado é um aspecto e pertence à aplicação e não ao framework. Portanto, provavelmente foi criado pelo desenvolvedor da aplicação.
<<aspect-framework>> ou <<aspect-f>>	Este estereótipo só pode ser aplicado a elementos do tipo Classe.	Denota que o elemento rotulado é um aspecto pertencente ao framework. Sendo assim, ele não foi criado pelo desenvolvedor da aplicação.
<<class-application>> ou <<class-app>>	Este estereótipo só pode ser aplicado a elementos do tipo Classe.	Denota que o elemento rotulado é uma classe e pertence à aplicação e não ao framework. Sendo assim, provavelmente foi criada pelo desenvolvedor da aplicação.
<<class-framework>> ou <<class-f>>	Este estereótipo só pode ser aplicado a elementos do tipo Classe.	Denota que o elemento rotulado pertence ao framework. Sendo assim, ele foi criado pelo desenvolvedor do framework.

<<pointcut-hook>>	Este estereótipo só pode ser utilizado em aspectos. Este estereótipo só pode ser aplicado a operações. Este estereótipo é usado na grande maioria das vezes em aspectos do framework.	Denota um ponto de instanciação do framework. O elemento rotulado é um ponto de corte que pode ser redefinido por um aspecto especializado. Geralmente esse aspecto especializado foi criado pelo desenvolvedor da aplicação.
<<template>>	Este estereótipo só pode ser aplicado a operações/métodos.	Denota que o padrão de projeto *Template Method* foi utilizado. Significa que esse método possui uma parte fixa e várias partes variáveis, que são fornecidas por métodos abstratos. Esses métodos abstratos possuem o estereótipo <<hook>>
<<hook>>	Este estereótipo só pode ser aplicado a métodos. Pode acontecer que os métodos rotulados com este estereótipo sejam ganchos de uma sugestão e não de um método normal.	Significa que esse método é um ponto de instanciação do framework e pode ser modificado pelo desenvolvedor da aplicação. Sua existência não implica na aplicação do padrão *Template Method*, pois podem existir vários métodos gancho independentes.
<<hook: nome da operação gabarito>>	Este estereótipo só pode ser aplicado a métodos. Pode acontecer que os métodos rotulados com este estereótipo sejam ganchos de uma sugestão e não de um método normal.	Possui a mesma semântica de um método gancho normal, porém este método gancho pertence a um determinado método gabarito, cujo nome é discriminado em “nome da operação gabarito”

Tabela 4-5: Estereótipo <<aspect-application>> [CAM 04]

4.5. Agrupar Preocupações e Implementar Advices

Para a utilização dos modelos propostos no capítulo 3 do presente trabalho, o próximo passo a ser realizado é a separação dos hot spots em grupos de preocupações ou interesses, respeitando o agrupamento realizado pelo Projetista de FAOA/FCOA. Este passo é de extrema importância, uma vez que para cada grupo de preocupação será criado um aspecto com um nome semelhante ao dado ao grupo.

Cada grupo irá se transformar em um aspecto, onde se deve implementar os *pointcuts* e *advices* relativos aos hot spots a serem implementados.

É nesta etapa que se decide quais hot spots ou *joinpoints* devem ser implementados ou não, para a aplicação a ser gerada pelo framework de componente orientado a aspectos. Quando se quer implementar um hot spot, deve-se criar um *pointcut* para o mesmo e caso não queira a implementação, basta não relacionar ou criar nenhum *pointcut* para o hot spot.

Deve-se sempre criar os *advices* com o tempo de interceptação *around*, para que seja possível fazer a composição do código implementado no *advice* relativo ao *pointcut* interceptado.

4.6. Utilizar Componentes e Gerar Aplicação

Para se gerar uma aplicação é necessária à instanciação do FOO desenvolvido, que deve ser realizado via configuração do componente de configuração com a utilização de uma Ferramenta de Geração de Aplicações que tenha acesso ao arquivo de configuração de componentes.

No momento em que se instancia o FOO, é realizada a consulta ao componente de configuração e conforme as informações nele contida, é realizada a composição dos FCOA´s que serão utilizados como componentes

pelo FOO através da composição dos FCOA´s nos hot spots do FOO. Gerando assim diferentes aplicações de uma mesma família de software.

4.7. Ferramenta de Configuração

A ferramenta de configuração (editor) é uma ferramenta que manipula o componente de configuração do framework orientado a objeto através da configuração dos componentes.

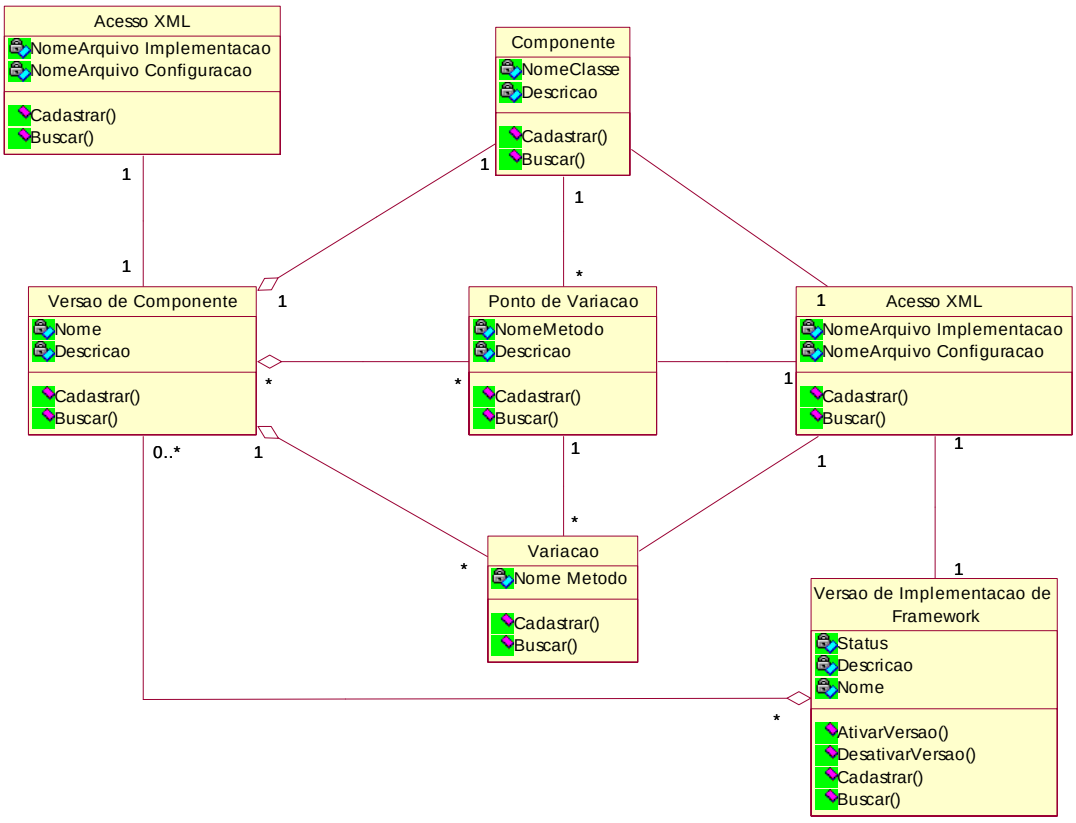


Figura 4-7: Diagrama de Classes do Pacote de Configuração

Conforme apresentado no diagrama de classes da figura 4-7 o pacote de configuração do framework manipula arquivos XML de configuração de

Versões de Componentes e de *Versões de Implementações de Produtos de Software* a partir da escolha dos *Pontos de Variação* e suas respectivas *Variações* a serem utilizadas pelo Framework de Aplicação Orientado a Objetos de uma família de software. Esta configuração é realizada através de uma ferramenta (editor) que importa o pacote de configuração proposto nos modelos II e III.

As *Versões de Componentes* são as possíveis implementações de um componente a serem utilizadas nos produtos de software.

A partir do conjunto de *Versões de Componentes* contendo as diferentes versões dos componentes que poderão compor o framework de aplicação, é possível definir como será composto o produto de software a ser gerado pela instanciação do framework de aplicação. Uma *versão do framework* contém quais são os componentes e as suas variações devem ser utilizadas para cada ponto de variabilidade que deverão compor o produto de software através da composição dos frameworks de componentes orientados a aspectos a ser realizada através da Programação Orientada a Aspecto.

5. Estudo de Caso

Escolheu-se um framework de sistema de assinaturas de correio, que foi apresentado por [ROB 02] uma vez que o modelo da aplicação e suas possíveis variações já foram definidas e validadas no trabalho acima citado, para se aplicar e validar os modelos aqui propostos.

5.1. Projeto do FCOA/FAOA

O modelo da aplicação é representado pela figura 5-1 abaixo, que apresenta um diagrama das características de um sistema de assinatura de correio eletrônico.

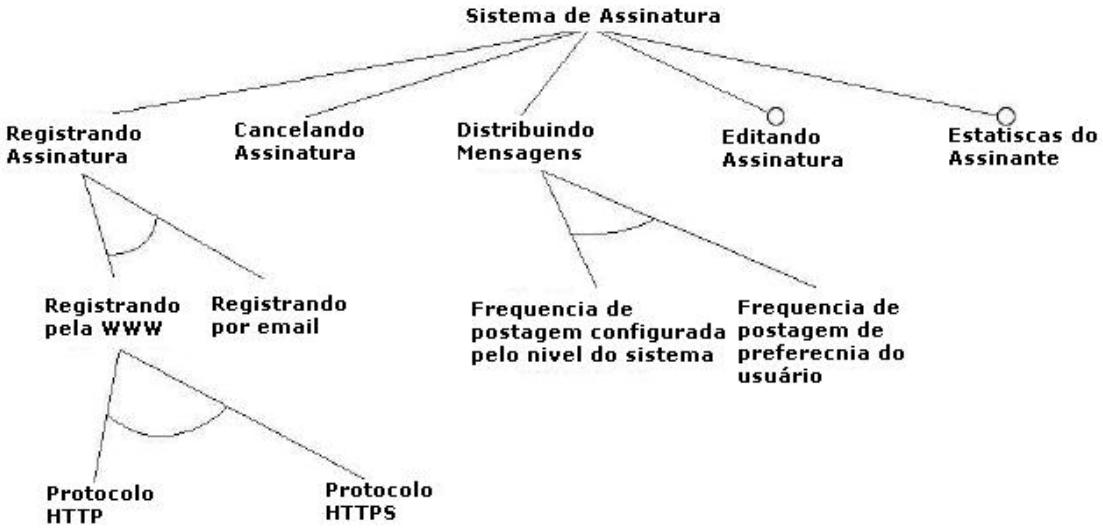


Figura 5-1: Diagrama de característica de sistema de assinatura de e-mail [ROB 02]

O diagrama de característica mostrado acima gera segundo [ROB 02] três diferentes sistemas que são representados na tabela 5-1 e descritos a seguir:

- **Sistema 1:** sistema comum baseado em assinatura por correio eletrônico: emite-se um E-mail com a palavra “assinatura” como assunto e torna-se registrado: o sistema não fornece uma possibilidade para editar dados e pode-se somente cancelar a assinatura.

- **Sistema 2:** este sistema é orientado para web: preenche-se um formulário (que neste caso é enviado pelo protocolo http) e torna-se registrado; o usuário é convidado ao mesmo tempo à responder algumas perguntas e o envio de mensagens depende destas respostas; o sistema permite editar dados do usuário.

- **Sistema 3:** é uma composição do correio e do sistema orientado a web; existe uma possibilidade adicional de emitir dados de registro em formulário criptografado (SSL/HTTPS); comprado com o sistema 2; não existe edição e a presença de módulos de estatística;

Sistema	Ponto de Variação	Característica	Projeto do Componente
Todos	Não Disponível.	Cancelando Assinatura	cancelSubscription()
S1	Registrando Assinatura	Registro por E-mail	registerMailSubsciption()
S1	Distribuindo Mensagens	Freqüência de postagem configuradas a nível do sistema.	postNews(); readConfigFiles();
S1	Editando Assinatura	Não Disponível.	Não Disponível.
S1	Estatísticas do Assinante	Não Disponível.	Não Disponível.
S2	Registrando Assinatura	Registro por Protocolo http.	registerWWWSubsciption();
S2	Distribuindo Mensagens	Freqüência de postagem configuradas a nível do usuário.	postNews(); readSubscriptionData();
S2	Editando Assinatura	Edição de Assinatura.	uptadeSubsciption()
S2	Estatísticas do Assinante	Estatísticas do Assinante.	registerPollAnswera(); evalStatistics();
S3	Registrando Assinatura	Registro por E-mail	registerMailSubsciption()
		Registro por Protocolo http.	registerWWWSubsciption();
		Registro por Protocolo https.	
S3	Distribuindo Mensagens	Freqüência de postagem configuradas a nível do usuário.	postNews(); readSubscriptionData();
S3	Editando Assinatura	Não Disponível.	Não Disponível.
S3	Estatísticas do Assinante	Não Disponível.	Não Disponível.

Tabela 5-1: Representação do Sistemas Gerados [ROB 02]

Dada a documentação acima, deve-se cumprir as etapas de Modelagem do FAOA/FCOA e o agrupamento de preocupações como definidos no capítulo 4.

5.1.1. Modelagem do FAOA/FCOA

Nesta etapa deve-se identificar os componentes a serem implementados para o sistema de assinatura de e-mail utilizado como exemplo e nomeá-los como FCOA´s conforme relacionados na tabela 5-2.

Componente	Função	Nome FCOA
Registro de Assinatura	Registrar Assinatura dos usuários	FCOARegistro
Distribuição de Mensagens	Distribuir Mensagens e ou Noticias	FCOADistribuição
Edição de Assinatura	Editar os dados de assinatura dos usuários.	FCOAEdição
Estatísticas do Assinante	Gerar as estatísticas dos usuários do sistema	FCOAEstatística

Tabela 5-2: FCOA´s encontrados no Sistema de Assinatura

No exemplo de sistema de assinatura de e-mail utilizado como exemplo os quatro componentes apresentados na tabela 5-2 foram identificados a partir do diagrama de características apresentado por [ROB 02].

5.1.2. Agrupamento das Preocupações

Nesta etapa deve-se realizar a separação das preocupações encontradas no sistema de assinatura de e-mail utilizado como exemplo conforme citado e relacionados na tabela 5-3.

Nome Preocupação	Descrição Preocupação
Registro	Atividades relacionadas ao registro de assinatura dos usuários do sistema
Distribuição	Atividades relacionadas a distribuir de mensagens e ou noticias
Edição	Atividades relacionadas à edição dos dados de assinatura dos usuário.
Estatísticas	Atividades relacionadas à geração e exibição das estatísticas dos usuário do sistema

Tabela 5-3: Agrupamento de Preocupações encontrados no Sistema de Assinatura

As preocupações apresentadas na tabela 5-3 foram separadas a partir da identificação de cada uma das características apresentadas para os componentes identificados na tabela 5-2.

Os pontos de variabilidade e suas variações para as preocupações representadas na tabela 5-3 acima estão representadas na tabela 5-4 abaixo.

Agrupamento por Preocupação	Ponto de Variabilidade	Variações	Implementação
Registro	Registro de Assinatura	Registro por E-mail.	registerWWWSubsciption
		Registro por Web	registerMailSubsciption
	Protocolo do Registro	Protocolo de E-mail	ProtocolMail
		Protocolo http	Protocolhttp
		Protocolo https	Protocolhttps
Distribuição	Envio de Mensagens	Enviar Mensagem	PostNews
	Freqüência de Envio	Configurada pelo Sistema	ReadConfigFile
		Configurada pelo Usuário	readSubscriberData
Edição	Edição da Assinatura	Editar Assinatura	updateSubscription
Estatística	Geração de dados estatísticas dos usuários	Gerar Dados Estatísticos	registerPollAnswers
	Exibição das estatísticas dos usuários	Exibir Estatística	evalStatistics

Tabela 5-4: Pontos de Variabilidade e Variações por Preocupações encontrados no Sistema de Assinatura

As variações apresentadas para cada ponto de variabilidade dos componentes para o Sistema de Assinatura de E-mail foram identificadas através da análise do diagrama de características apresentado na figura 5-1 e mostrados na tabela 5-4.

5.1.3. Classificação dos Pontos de Variação na Linha de Produtos de Software

A classificação dos pontos de variação dentro de cada um dos produtos da linha de produtos, conforme se demonstra a tabela 5-5.

Sistema	Componente	Ponto de Variação	Tipo de Variação
S1	FCOARegistro	Tipo de Registro	Uma variação entre várias alternativas
		Protocolo de Registro	Uma variação entre várias alternativas
	FCOADistribuição	Configuração da Freqüência de Postagem	Uma variação entre várias alternativas
		Envio de Mensagem	Variação Opcional
	FCOAEdição	Edição da Assinatura	Variação Opcional
	FCOAEstatísticas	Respostas Estatísticas	Variação Opcional
		Resultados Estatísticos	Variação Opcional
S2	FCOARegistro	Tipo de Registro	Uma variação entre várias alternativas
		Protocolo de Registro	Uma variação entre várias alternativas
	FCOADistribuição	Configuração da Freqüência de Postagem	Uma variação entre várias alternativas
		Envio de Mensagem	Variação Opcional
	FCOAEdição	Edição da Assinatura	Variação Opcional
	FCOAEstatísticas	Respostas Estatísticas	Variação Opcional
		Resultados Estatísticos	Variação Opcional

S3	**FCOARegistro**	Tipo de Registro	Uma combinação de variações entre várias alternativas
		Protocolo de Registro	Uma combinação de variações entre várias alternativas
	FCOADistribuição	Configuração da Freqüência de Postagem	Uma variação entre várias alternativas
		Envio de Mensagem	Uma variação entre várias alternativas
	FCOAEdição	Edição da Assinatura	Variação Opcional
	FCOAEstatísticas	Respostas Estatísticas	Variação Opcional
		Resultados Estatísticos	Variação Opcional

Tabela 5-5: Classificação dos Pontos de Variabilidade dentro da Linha de Produtos de Softwares de uma Familia de Software

A classificação dos Pontos de variabilidade dentro da Linha de Produtos de Software realizado na tabela 5-5 se mostra necessária para a verificação de viabilidade de implementação dos diferentes sistemas através do modelo proposto.

Tendo-se classificado os pontos de variação dentro de uma linha de produtos de uma família de softwares, deve-se verificar quais são as limitações do modelo aplicado para identificar quais são os produtos da família de software serão viáveis de serem implementados com o modelo proposto.

O exemplo da família de software em questão (sistema de assinatura de e-mail) apenas o sistema 3 que não é viável de implementação, uma vez que ele possui pontos de variação que são do tipo “uma combinação de variações entre várias alternativas” e esta é exatamente a limitação do modelo proposto. Por este motivo só será possível implementar com o framework apresentado neste caso de uso os sistemas 1 e 2.

5.2. Desenvolvimento do FCOA/FAOA

O desenvolvimento do FCOA deve respeitar as regras sugeridas no capítulo 3 do presente trabalho.

Os passos a serem seguidos são:

1) A transformação dos pontos de variabilidade em Hot Spots e pointcuts que devem interceptar o hot spot correspondente;

2) Os agrupamentos de preocupações devem ser transformados em aspectos;

3) As implementações de variabilidade devem ser transformadas em classes criadoras de objetos, conforme o padrão abstract factory, que serão instanciadas pelos advices correspondentes.

A figura 5-2 apresenta o diagrama de classes do componente de distribuição modelado utilizando a notação sugerida por [FON 99] antes da criação e utilização dos aspectos e do projeto de interceptação dos hotspots.

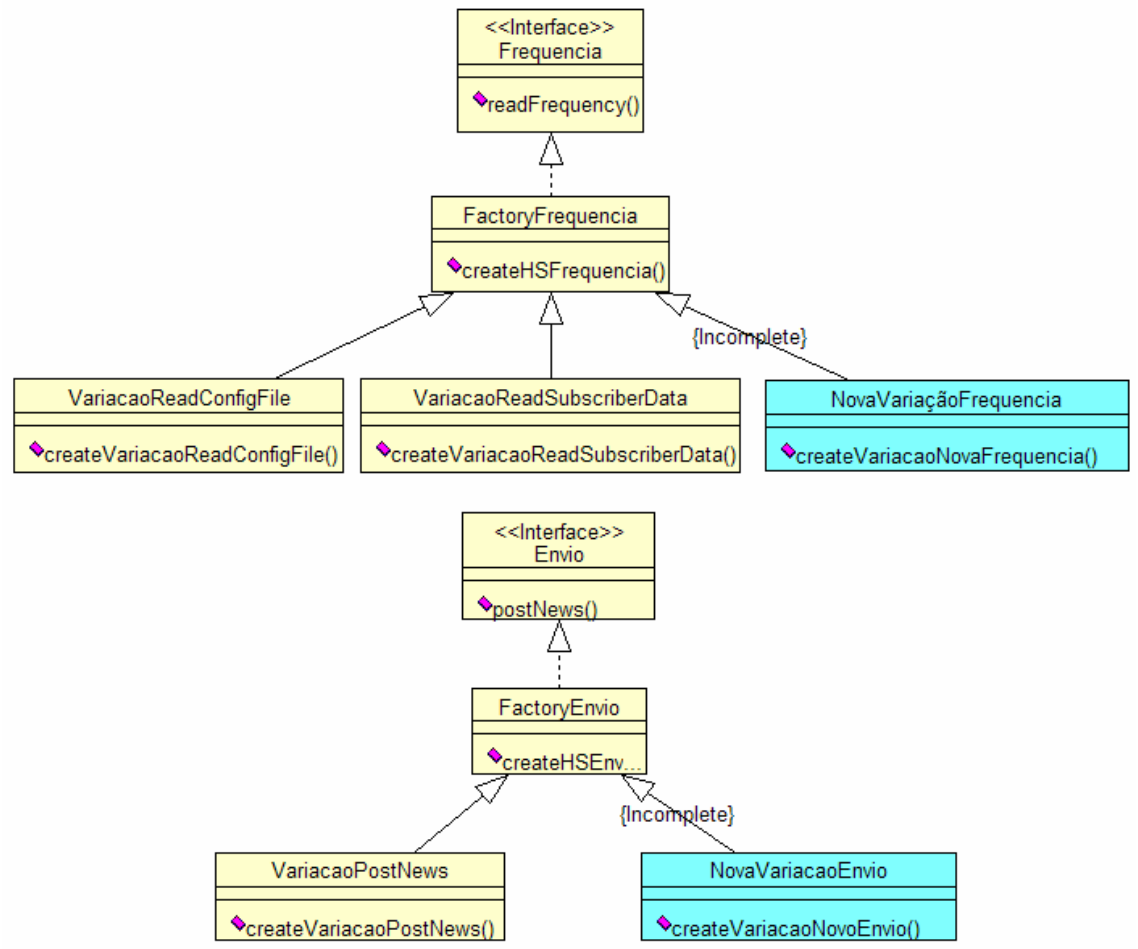


Figura 5-2: Diagrama de Classes do Componente de Distribuição

A figura 5-3 apresenta o diagrama de classes do framework de componentes orientado a aspectos de distribuição (FCOADistribuição), após a adaptação do framework orientado a objetos representado pelo diagrama de classes da figura 5-2.

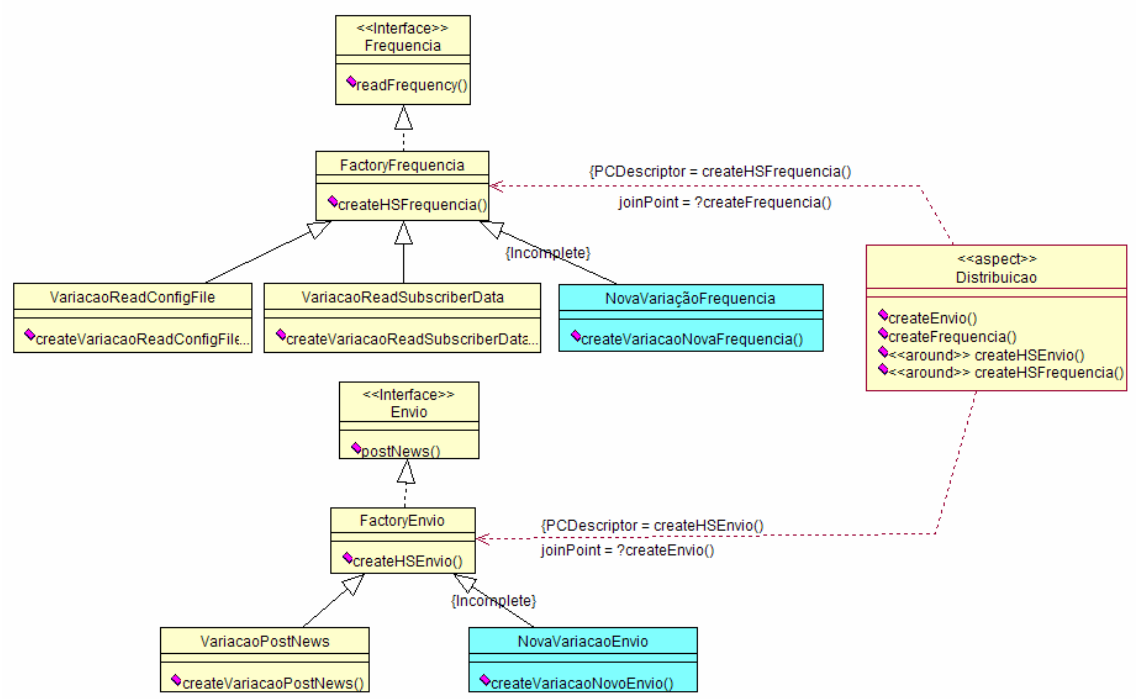


Figura 5-3: Diagrama de Classes do FCOADistribuição

A figura 5-3 apresenta o diagrama de classes do FCOADistribuição onde tem-se o aspecto distribuição que é responsável pela interceptação dos métodos de criação das fábricas abstratas de criação de Envio de Mensagem e Freqüência de Postagem de Envio.O aspecto de distribuição é responsável pela configuração de qual fábrica concreta deve ser executada para a composição dos pontos de variabilidade por uma variação configurada no pacote de configuração.

5.3. Projeto e Desenvolvimento do FOO

O FOO deve ser projetado e desenvolvido conforme as técnicas de projeto e desenvolvimento sugeridas, utilizando os componentes gerados na seção 5.2 da presente dissertação de mestrado.

A figura 5-4 apresenta o diagrama de componentes do framework orientada a objetos de sistema de assinatura de correio eletrônico.

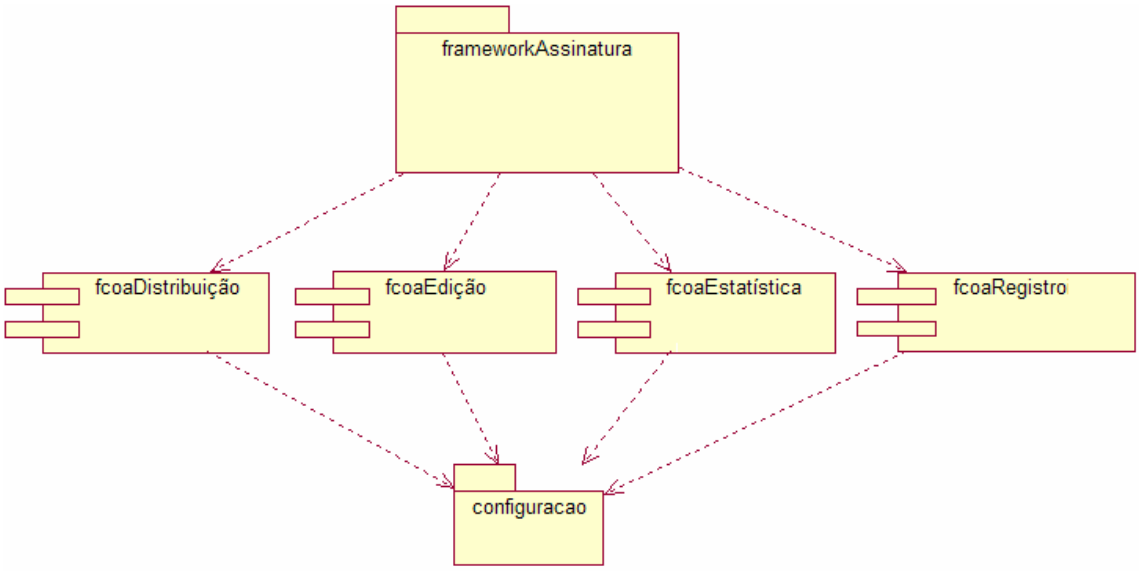


Figura 5-4: Diagrama de Componentes do Framework de Assinatura

A figura 5-4 apresenta o diagrama de componentes do framework orientada a objetos de sistema de assinatura de correio eletrônico, analisando-se o diagrama, percebe-se que o Framework de Assinatura é composto pelos frameworks de componentes orientado a aspectos criados no item anterior.

5.4. Desenvolvimento de Componentes e Aplicação.

O desenvolvedor da aplicação deve utilizar a ferramenta de configuração apresentada na sessão 4.7 do presente trabalho para realizar a configuração de quais variações devem ser utilizadas na composição de cada um dos componentes a ser utilizado pelo framework de assinatura, gerando assim as duas primeiras aplicações sugeridas por [ROB 02]. A terceira aplicação

proposta não é viável de ser gerada, uma vez que entra na limitação encontrada do mecanismo proposto, conforme já apresentado anteriormente.

Para se gerar uma aplicação diferente das aplicações sugeridas por [ROB 02] é necessário apenas fazer uma combinação de escolha das variações para cada ponto de variabilidade dos componentes diferente das combinações sugeridas.

6. Conclusões

Esta dissertação apresentou uma proposta de utilização de programação orientada a aspectos para instanciar pontos de variabilidade em um framework que descreve um componente de software através de um mecanismo de reuso para componentes de negócios orientados a aspectos.

Para se conseguir integrar estes conceitos (aplicação, componentes reutilizáveis, frameworks e aspectos) foi proposto um modelo de implementação de variantes em componentes de software.

Com a criação deste modelo lógico criou-se uma especialização dos frameworks de aplicação orientados a aspectos (FAOA) que foi nomeado como framework de componentes orientado a aspectos (FCOA).

A utilização do modelo, do mecanismo e da ferramenta aqui propostos é mais um passo para a área de engenharia de software para a construção das fábricas de softwares, podendo auxiliar no aumento da produtividade nos meios de projeto e desenvolvimento de software.

6.1. Contribuições da dissertação

A primeira contribuição é ter conseguido dar mais um passo para que a utilização da programação orientada a aspectos não fique limitada apenas na

utilização de problemas relativos a requisitos não funcionais de uma aplicação mostrando que é possível a sua utilização no auxilio do reuso de componentes que implementam regras de negócios. Tendo proposto um mecanismo para implementar variabilidade de software aplicado a componentes de sistema.

Outra contribuição importante foi a extensão dos Frameworks de Aplicação Orientado a Aspectos (FAOA), propondo os Frameworks de Componentes Orientado a Aspectos (FCOA), que pode ser utilizado como ferramenta para o desenvolvimento de componentes de negócios que visam o reuso.

Criação de um mecanismo de reuso que fornece os passos para a utilização de aspectos na implementação de variabilidade em um nível alto (conceitual ou lógico) de modelagem de componentes e sistemas aplicativos, apresentando uma maneira de se utilizar Programação Orientada a Aspectos em nível de regras de negócios (features).

Desenvolvimento de uma ferramenta que auxilia no processo de desenvolvimento de softwares e facilita a utilização de frameworks de software que são desenvolvidos especialmente para uma família de software.

6.2. Limitações e Dificuldades

O mecanismo proposto apresenta a limitação de implementação de variantes relativa ao tipo de variabilidade "*uma combinação de variações entre*

várias alternativas". Apesar de o problema poder ser contornado através da transformação de uma variabilidade do tipo "*uma combinação de variações entre várias alternativas*" em uma variabilidade do tipo "*uma alternativa de variação entre várias alternativas*".

6.3. Trabalhos futuros

A partir do framework orientado a objetos e dos frameworks de componentes orientados a aspectos gerados a partir do mecanismo proposto pelo presente trabalho é possível o desenvolvimento de uma ferramenta de geração de aplicações que consiga ter acesso ao FOO e aos FCOA´s criados com o objetivo de se gerar automaticamente o arquivo de configuração XML a ser utilizado pela ferramenta de configuração aqui proposta.

É necessário a realização de estudos para tornar possível a eliminação da limitação de implementação existente no modelo aqui proposto relativa ao tipo de variabilidade "*uma combinação de variações entre várias alternativas*".

7. Referências Bibliográficas

[AHM 02] AHMED, Khawar Zamam, UMRYSG, Cary E. **Desenvolvendo aplicações comerciais em Java com J2EE e UML**. Rio de Janeiro: Editora Ciência Moderna Ltda., 2002.

[AOS 05] Aspect-Oriented Software Develoment Community & Conference, http://www.aosd.net/ (04/07/2005)

[BAD 99] BADER, Atef, CONSTANTINIDES, Constantinos A., ELRAF, Tzilla. **An Aspect- Oriented Design Framework for Concurrent Systems.** Proceedings of the Aspect-Oriented Programming Workshop at ECOOP'99, Finlândia: Springer-Verlag,
1999.

[BAS 98] BASS, L.; CLEMENTS, P.; KAZMAN, R.. **Software Architecture in Practice**. Addison Wesley Longman, 1998.

[BAT 03] BATORY, D.; SMARAGDAKIS, Y. **Application Generators.** Mar. 2003. Generators. Department of Computer Sciences. The University of Texas at Austin. Disponível em: <http://www.cc.gatech.edu/~yanni>.

[BOS 99] BOSCH, J., MOLIN, O.; MATTSSON, M.; BENGTSSON, P.; FAYAD, M. **Framework problem and experiences im M. Fayad, R. Johnson, D. Schmidt.** Building Application Frameworks: Object-Oriented Foundations of Framework Design, John Willey and Sons, p. 55-82, 1999.

[BUS 96] BUSCHMANN, F.; MEUNIER, R.; ROHNERT, H.; SOMMERLAND, P.; STAL, M. **Patterns-oriented software archutecture – a system of patterns.** Wiley & Sons, 1996.

[CAM 04] CAMARGO, Valter. **UML-AOF – Um Perfil UML para o Projeto de Frameworks Orientados a Aspectos**. Relatório Técnico. ICMC-USP, São Paulo, 2004.

[CAM e MAS 04] CAMARGO, Valter. MASIERO, Paulo César. **UML-AOD – Um Perfil UML para o Projeto de Sistemas Orientados a Aspectos**. ICMC-USP, São Paulo, 2004.

[CAM 05] CAMARGO, Valter, Masiero, Paulo César. **Frameworks Orientados a Aspectos**. XIX SBES-2005 – Simpósio Brasileiro de Engenharia de Software, Uberlândia, Brasil, 2005.

[CIB 03] Cibran, M., D'Hondt, M. and Jonckers, V. **Aspect-Oriented Programming for Connecting Business Rules.** In Proceedings of the 6th International Conference on Business Information Systems. Colorado Springs, USA, June 2003.

[CLE 01] CLEMENTS, P. C.; NORTHROP, L. **Software Product Lines: Practice and Patterns.** [S.l.]: Addison-Wesley, 2001. SEI Series in Software Engineering.

[COC 94] COCKBURN, A. **Writing Effective Use Cases (Draft n.3)**, Addison-Wesley Longman, 2000.

[CON 00] CONSTANTINIDES, C. A.; BADER, A.; ELRAD, T.H., FAYAD, M. F. **Designing an Aspect-Oriented Framework in an Object-Oriented Environment**. ACM Computing Surveys, v. 32, 2000.

[CRN 02] CRNKOVIC, I.; HNICH B.; JOHSSON, T.; KIZILTAN, Z. **Specification, implementation, and deployment of components.** Communications of the ACM, ACM Press, v. 45, n.10, p. 35-40, out. 2002.

[DSO 99] D´SOUZA, D. F.; WILLS A. W. **Objects, components and frameworks with UML – the catalysis approach.** Addison-Wesley, 1999.

[FAB 95] FABRE, J., NICOMETTE, V., et Al. **Implementing Fault Tolerant Applications Using Reflective Object-Oriented Programming**. Proceedings of the 25th IEEE International Symposium on Fault-Tolerant Computing, Pasadena, CA, EUA, Jun. 1995.

[FAY 97] FAYAD, M.; SCHIMIDT. D. C. **Object-oriented application framework**. Comum. ACM, ACM Press, v.40. n.10. p. 32-38, 1997.

[FAY 99] FAYAD, M.; SCHIMIDT. D. C.; JOHNSON, R. **Building application frameworks: Object-oriented foundations of framework design.** John Wiley & Sons, 1999.

[FON 99] FONTOURA, M. **A Systematic Approach to Framework Development.** Tese de doutorado da Pontifícia Universidade Católica do Rio de Janeiro.Julho de 1999.

[FON 02] FONTOURA, M.; PREE, W.; RUMP, B. **The UML Profile for Framework Architectures.** Addison Wesley, 2002.

[FOO 88] FOOTE, B.; JOHNSON, R. E. **Designing reusable classes.** Journal of Object Oriented Programming, v. 1, n. 2, p. 22-35, 1988

[FRA 96] FRAKES, W.; TERRT, C. **Software Reuse: metrics and models**. ACM Computing Surveys (CSRU), ACM Press, v. 28, n.2, p. 415-435, out. 1996.

[FRA 01] FRANÇA, L. P. A.; STAA, A. V. **Geradores de artefatos: Implementação e instanciação de frameworks.** In.: Anais do XV SBES-2001 – Simpósio Brasileiro de Engenharia de Software, Rio de Janeiro, Brasil, 2001. p. 302-315.

[FRA 98] FRANCH, X.; BOTELLA, P. **Putting Non-Functional Requirements into Software Architecture**. In Proceedings 9th IEEE International Workshop

on Software Specification and Design (IWSSD), Ise-shima (Japan), April 1998, p.60-67.

[FRY 01] FRYE, J; YODER, J. **The hillside group – patterns home page.** Out. 2001. Disponível em: <http://hillside.ne>.

[FUR 98] FURLAN, José Davi. **Modelagem de Objetos atraves da UML – the Unified Modeling Language**. São Paulo, Makron Books, 1998.

[GAM 93] GAMMA, Erich, HELM Richard, JOHNSON Ralph e VLISSIDES John; **Design patterns – abstraction and reuse of object-oriented design**. Lecture Notes in Computer Sciente, n. 707, p. 406-431, 1993

[GAM 95] GAMMA, Erich, HELM Richard, JOHNSON Ralph e VLISSIDES John; **Design Patterns – Elementes Of Reusable Object-Oriented Software.** Addison-Wesley, 1995.

[GAM 00] GAMMA, Erich, HELM Richard, JOHNSON Ralph e VLISSIDES John; **Padrões de Projeto: Soluções Reutilizáveis de Software Orientado a Objetos.** Porto Alegre: Bookman, 2000.

[GAR 01] GARCIA, A.; LUCENA, C.. **An Aspect-Based Object-Oriented Model for Multi-Agent Systems**. In: Advanced Separation of Concerns in Software Engineering at ICSE'2001, Toronto, May 2001.

[GRI 98] GRISS, M., FAVARO, J., ALESSANDRO, M. **Integrating Feature Modeling with RSEB**, 5th International Conference on Software Reuse (ICRS-5), ACM/IEEE, Victoria, Canadá, Junho 1998.

[GRI 00] GRISS, M., **Implementing Product-Line Features with Component Reuse**, 5th International Conference on Software Reuse, ACM/IEEE, Vienna, Austria, Junho 2000.

[HAN 01] HANENBERG, S.; UNLAND, R.: **Using and Reusing Aspects in AspectJ**. Workshop on Advanced Separation of Concerns, OOSPLA, 2001.

[HAN 03] Hanenberg, S., Schmidmeier. **Idioms for Building Software Frameworks in AspectJ**. 2nd AOSD Workshop on Aspects, Components, and Patterns for Infrastructure Software (ACP4IS), Boston, MA, March 17, 2003.

[JAC 97] JACOBSON, Ivar; GRISS, Martin; JONSSON, Patrick. **Software Reuse: Architecture, Process, and Organization for Business Success**. Addison Wesley Longman, 1997.

[JAC 99] JACOBSON, I., BOOCH, G., RUMBAUGH, J., **The Unified Software Development Process**, Addison-Wesley, 1999.

[JOH 92] JOHNSON, R. E. **Documenting frameworks using patterns**. In: Conference proceedings on Object-oriented programming systems, languages,

and aplications. Vancouver, British Columbia, Canada: ACM Press, 1992. p.63-76.

[JOH 93] JOHNSON, R. E. **How to design frameworks.** 1993. Disponível por FTP anônimo em st.cs.uiuc.edu.

[JOH 97] JOHNSON, R. E. **Componentes, frameworks, patterns.** In: Proceedings of the 1997 symposium on Software reusability. Boston, Massachusetts, USA: ACM Press, 1997. p. 10-17.

[KAN 90] KANG, K. et al. **Feature-Oriented Domain Analysis (FODA) Feasibility Study**, (CMU/SEI-90TR-21, ADA 235785), Pittsburgh, PA:SEI CMU, 1990.

[KIC 96] Kiczales G., Lamping J., Mendhekar A, Maeda C, Lopes C., Loingtier J.-M., and Irwin J. **Aspect-Oriented Programming**. ACM Computing Surveys, 1996 - portal.acm.org
ACM Computing Surveys 28(4es), Dezembro1996.

[KIC 97] KICZALES, G.; LAMPING, J.; MENDHEKAR, A.; MAEDA, C.; LOPES, C.; LOINGTIER, J. ; IRWIN, J.. **Aspect-Oriented Programming**. In: Aksit, M.; Matsuoka, S., editors, 11th European Conference on Object-oriented Programming, volumen 1241 de LNCS, p. 220-242. Springer-Verlag, 1997.

[LEE 01] LEE, Richard; TEPFENHART, William M.. **UML and C++ - A Practical Guide to Object-Oriented Development**. Prentice Hall, Inc. 2001.

[LIP 00] LIPPERT, M.; LOPES, C. **A Study on Exception Detection and Handling Using Aspect-Oriented Programming**. In Proceedings of the 22nd Internacional Coference of Software Engineering (ICSE'2000), Limmerick, Ireland. IEEE Computer Society. June 2000.

[NEL 93] NELSON, N. M.; HAMILTON, G.; KHALIDI, Y. A. **Caching in an Object-Oriented System.** In Proceedings 3rd Internacional Workshop on Object Orientation in Operating System (I-WOOOSIII), December 1993, p. 95-106.

[PAE 99] DE PAEZ, R. A.; **Un acercamiento a la reutilización en ingeniería de software**. EAFIT Magazine, n. 114. p. 45-63, 1999. Universidade de EAFIT.

[PAW 02] PAWLAK, R., DUCHIEN, L., FLORIN G., LEGONG-AUBRY, F., SEINTURIER, L, MARTELLI, L. **A UML Notation for Aspect-Oriented Software Design**. In Proceedings of Workshop of Aspect Oriented Modeling with UML of Proceedings of Aspect Oriented Software Development Conference (AOSD) 2002.

[PIN 02] Pinto, M.; Fuentes, L.; Fayad, M.E. e Troya, J.M.. **Separation of Coordination in a Dynamic Aspect Oriented Framework**. April 2002.

[PIV 01] PIVETA, Eduardo Kessler. **Um Modelo de Suporte a Programação Orientada a Aspectos**. Universidade Federal de Santa Catarina, Junho de 2001. Dissertação de Mestrado.

[PRE 99] PREE, W. **Hot-spot-driven development.** In: FAYAD, M.; JOHNSON, R.; SCHIMIDT, D. (Ed.). **Building Application Frameworks: Object-Oriented Foundations os Framework Design.** Wiley & Sons, 1999. p. 379-393.

[RAS 03] RASHID, A.; Chitchyan, R. **Persistence as an Aspect**. In: Proceedings of 2nd
International Conference on Aspect Oriented Software Development – AOSD. Boston – USA, março, 2003.

[RES 05] RESENDE, Antônio Maria Pereira de, SILVA, Claudiney Calixto da. **Programação Orientada a Aspectos em Java.** Rio de Janeiro: Brasport, 2005.

[ROB 02] ROBAK, SILVA; FRANCZYK, BOGDAN; POLITOWICZ, KAMIL. **Extending The UML for Modelling Variability for System Families**. In Int. J. Appl. Math. Comput. Sci., 2002, Vol.12, No.2, 285–298.

[RUM 99] RUMBAUGH, J., JACOBSON, I., BOOCH, G., **The Unified Modeling Languages Reference Manual**, Addison-Wesley, 1999.

[SHA 03] SHANH, V.; HILL, F. **An Aspect-Oriented Security Framework**. In IEEE, Proceedings of the DARPA Information Survivability Conference and Exposition. 2003.

[SOA 02] SOARES S., Laureano E., Borba P., **Implementing Distribution and Persistence Aspects with AspectJ**. In: Proceedings of the 17th ACM Conference on Object-Oriented programming systems, languages, and applications, OOPSLA'02, pages 174-190, ACM Press. 4th-8th November 2002, Seattle, WA, USA. ACM SIGPLAN Notices 37(11)

[SOA 04] Soares S. , Borba P., **PaDA: A pattern for distribution aspects**. In: Second Latin American Conference on Pattern Languages Programming – SugarLoafPLoP 2002. 5th-7th August 2002, Itaipava, RJ, Brazil. *Anais do SBgames, outubro de 2004* ICMC - University of São Paulo Magazine, pages 87-99.

[SOM 01] SOMMEVILLE, I. **Software Engineering.** Addison-Wesley, 2001.

[SUV 05] Suvée, D. Fraine, B. De; and Vanderperren W.. **Fusej: An architectural escription language for unifying aspects and components**. In Software-engineering Properties of Languages and Aspect Technologies Workshop. 2005.

[TAL 94] TALIGENT. **Building object-oriented frameworks.** [S.1.]: Taligent, 1994. White paper.

[TAL 95] TALIGENT. **Leveraging object-oriented frameworks.** [S.1.]: Taligent, 1995. White paper.

[TIM 02] TIMMIS, J.; LEMOS, de R.; AYARA, M.; DUNCAN, R. **Towards immune inspired fault tolerance in embedded systems.** In Proceedings of 9th Internacional Conference on Neural Information Processing, IEEE, November, 2002, p. 1459-1463.

[WIA 91] WIRFS-BROCK, A. et al. **Designing reusable designs: Experiences designing object-oriented frameworks**. In: Object-Oriented Programming Systems, Languages and Applications Conference; European Conference on Object-Oriented Programming, 1991. Ottawa. **Addendum to the proceeddings.** Ottawa: [s.n.], 1991

[YOD 98] YODER, J. W.; JOHNSON, R. E.; WILSON, Q. D. **Connecting Business Objects to Relational Databases**. In Conference on the Pattern Languages of Programs, Monticello, EUA, Proceedings. August, 1998.

Apêndice

1. *Código Fonte do Framework de Sistema de Assinatura de E-mail.*

SistemaAssinatura.java

```
/*
 * Created on 13/04/2006
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package frameworkAssinatura;

import fcoaDistribuicao.*;
import fcoaEdicao.HSEditarAssinatura;
import fcoaEdicao.FactoryEditarAssinatura;
import fcoaEstatistica.FactoryRespostasEstatisticas;
import fcoaEstatistica.FactoryResultadosEstatisticas;
import fcoaEstatistica.HSRespostasEstatisticas;
import fcoaEstatistica.HSResultadosEstatisticas;
import fcoaRegistro.FactoryProtocoloRegistro;
import fcoaRegistro.FactoryTipoRegistro;
import fcoaRegistro.HSProtocoloRegistro;
import fcoaRegistro.HSTipoRegistro;


/**
 * @author Cristiano
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
public class SistemaAssinatura {
	SistemaAssinatura(){
		usaFramework();
	}

	private void usaFramework() {
		FSCancelamento cancelamento = new Cancelamento();
		HSTipoRegistro registro = FactoryTipoRegistro.createHSTipoRegistro();
		HSProtocoloRegistro protocolo =
FactoryProtocoloRegistro.createHSProtocoloRegistro();
		HSEnvio envio = FactoryEnvio.createHSEnvio();
		HSFrequencia frequencia = FactoryFrequencia.createHSFrequencia();
		HSEditarAssinatura edicao =
FactoryEditarAssinatura.createHSEditarAssinatura();
		HSRespostasEstatisticas respostas =
FactoryRespostasEstatisticas.createHSRespostasEstatisticas();
		HSResultadosEstatisticas resultados =
FactoryResultadosEstatisticas.createHSResultadosEstatisticas();

		registro.registrarAssinatura();
		cancelamento.cancelSubscription();
		envio.postNews();
		frequencia.readFrequency();
		protocolo.protocoloRegistro();
```

```
                edicao.editarAssinatura();
                respostas.respostasEstatisticas();
                resultados.resultadosEstatisticas();
        }

        public static void main(String[] args) {
                new SistemaAssinatura();
        }
}
```

FactoryCancelamento.java

```
/*
 * Created on 11/04/2006
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package frameworkAssinatura;

/**
 * @author Cristiano
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
public class FactoryCancelamento {
        public static FSCancelamento createFSCancelamento(){
                FSCancelamento retorno = null;
                return retorno;
        }
}
```

FSCancelamento.java

```
/*
 * Created on 11/04/2006
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package frameworkAssinatura;

/**
 * @author Cristiano
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
public interface FSCancelamento {
        public void cancelSubscription();
}
```

Cancelamento.java

```
/*
 * Created on 13/04/2006
 *
 * TODO To change the template for this generated file go to
```

```
 * Window - Preferences - Java - Code Style - Code Templates
 */
package frameworkAssinatura;

/**
 * @author Cristiano
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
public class Cancelamento implements FSCancelamento{
        public void cancelSubscription() {
                System.out.println("FrozenSpot - Aqui se implementa todo o código de
Cancelamento de Assinatura.");
        }
}
```

FSCancelamento.java

2. Código Fonte do Pacote de Configuração.

AcessoXML.java

```
package configuracao;
import org.w3c.dom.*;

import java.io.*;
import javax.xml.parsers.DocumentBuilderFactory;
import java.util.ArrayList;

/**
 *
 * @author Administrador
 */
public class AcessoXML
{

   private String nome_arquivo;
   private Document documento_principal;
   private Element elemento_principal;


   /** Creates a new instance of AcessoXML */
   public AcessoXML(String nom)
   {
      nome_arquivo = nom;

      try
      {
         DocumentBuilderFactory dbf=DocumentBuilderFactory.newInstance();
         javax.xml.parsers.DocumentBuilder db = dbf.newDocumentBuilder();
         documento_principal = db.parse(nome_arquivo);
         elemento_principal = documento_principal.getDocumentElement();
      }
      catch(Exception e) {
          System.out.println("O aquivo nao foi encontrado, ou ocorreu um erro ao interpreta-
lo...");
```

```
    }
}

/**
 * Funcao que percorre os nodos xml e retorna um string com o arquivo
 * @param elemento contendo a partir de onde será percorrido o aquivo
 * @param espaco contendo os espacos para identacao do arquivo
 * @return String ret contendo o arquivo
 */
public String PercorreArquivo(Element elemento, String espaco)
{
    espaco = espaco+"    ";
    String ret = new String();


    ret = "\r\n"+espaco+"<"+elemento.getNodeName();

    int length = 0;
    if (elemento.getAttributes() != null)
    {
        length = elemento.getAttributes().getLength();
    }

    /*Pegando os atributos do campo*/
    Attr attributes[] = new Attr[length];
    for (int i = 0; i < length; i++)
    {
        Attr attribute = (Attr)elemento.getAttributes().item(i);
        attributes[i] = attribute;
        ret += " "+attribute.getNodeName()+"=\""+attribute.getNodeValue()+"\"";
    }
    ret += ">";

    /*Pegando os nodos filhos do campo*/
    if (elemento.getFirstChild().getNodeValue() != null)
    {
       ret = ret + elemento.getFirstChild().getNodeValue().trim();
    }

    NodeList filhos = elemento.getChildNodes();
    for (int i = 1; i < filhos.getLength(); i = i+2)
    {
       ret = ret + PercorreArquivo((Element) filhos.item(i), espaco);
    }
    ret = ret + "\r\n" + espaco+"</"+elemento.getNodeName()+">";
    return ret;
}

        public String load_valor_nodo(Node pai, String nome_nodo)
        {
                NodeList temp_classe = load_filho_elemento((Element)pai, nome_nodo);
                Element classe = (Element)temp_classe.item(0);

                return classe.getFirstChild().getNodeValue();
        }

        public ArrayList load_valor_nodo_variacao(Node pai, String nome_nodo){
                ArrayList retorno = new ArrayList();
                String temp = null;
                NodeList temp_classe = load_filho_elemento((Element)pai, nome_nodo);
```

```
                        Element classe = (Element)temp_classe.item(0);
                        retorno.add(classe.getFirstChild().getNodeValue());
                        while ((temp = classe.getLastChild().getNodeValue()) != null){
                                retorno.add(temp);
                        }

                        return retorno;
                }
        /**
         * Funcao que retorna o nodo principal do documento XML
         * @return
         */
                public Element load_elemento_principal()
        {
           return elemento_principal;
        }


                /**
                 * Funcao que retorna os filhos de um nodo de acordo com o seu nome
                 * @param nome_nodo contendo o nome do nodo a ser buscado
                 * */
                public NodeList load_elemento(String nome_nodo)
        {
           Element element = load_elemento_principal();
           return element.getElementsByTagName(nome_nodo);
        }

                /**
                 * Fucnao que retorna os filhos de um nodo de acordo com o seu nome
                 * @param nome_nodo contendo o nome do nodo a ser buscado
                 * */
                public NodeList load_filho_elemento(Element pai, String nome_filho)
        {
           return pai.getElementsByTagName(nome_filho);
        }


                public Element load_elemento_atributo(String nome_nodo_pai, String
nome_nodo_procura, String nome_atributo, String valor_atributo)
                {
                        Element ret = null;

                        NodeList pais = load_elemento(nome_nodo_pai);
           for (int i=0;i < pais.getLength();i++)
           {
              Element pai = (Element) pais.item(i);

              NodeList filhos = pai.getElementsByTagName(nome_nodo_procura);

              for (int j = 0; j < filhos.getLength(); j++)
              {
               Element filho = (Element) filhos.item(j);

               int length = 0;
                 if (filho.getAttributes() != null)
                 {
                        length = filho.getAttributes().getLength();
                 }
```

```
                    /*Pegando os atributos do campo*/
                    for (int k = 0; k < length; k++)
                    {
                            Attr attribute = (Attr)filho.getAttributes().item(k);
                            if (nome_atributo.equals(attribute.getNodeName()) &&
valor_atributo.equals(attribute.getNodeValue()))
                            {
                                    ret = filho;
                            }
                    }

                }
            }

                            return ret;
                    }

                    /**
                     * Funcao que grava um arquivo XML
                     */
            public void Grava()
            {
               try{
                  FileWriter file = new FileWriter(nome_arquivo);
                  String temp = new String("<?xml version=\"1.0\"?>\r\n");
                  file.write(temp.toCharArray());
                  file.write( PercorreArquivo(elemento_principal, "").toCharArray() );
                  file.close();
               }catch(Exception e){}
            }

            /**
             * Funcao que retorna o aquivo XML em texto
             * @return String contendo o arquivo
             */
            public String Mostra()
            {
                    return PercorreArquivo(elemento_principal, "");
            }

            /**
             * Funcao que insere um nodo no XML
             * @param nome_nodo
             * @return
             */
            public Element InsereNodo(String nome_nodo)
            {
               return InsereNodo(elemento_principal.getNodeName(), nome_nodo);
            }

            /**
             * Funcao que insere um nodo no XML em outro nodo
             * @param pai contendo o nodo a partir de onde será inserido o nodo
             * @param nome_nodo contendo o nome do nodo
             * @return Element contendo o nodo criado
             */
            public Element InsereNodo(String pai, String nome_nodo)
            {
               return InsereNodo(elemento_principal.getNodeName(), nome_nodo, "");
            }
```

```
        //Ex.:  InsereNodo("ATORES", "testenodo", "valor qualquer");
           /**
            * Funcao que insere um nodo a partir de outro com o valor do nodo criado
            */
           public Element InsereNodo(String pai, String nome_nodo, String valor)
           {
              Element newElement = null;
              try{
                 NodeList temp = elemento_principal.getElementsByTagName(pai);

                 Element tagPai = (Element) temp.item(0);

                 newElement = documento_principal.createElement(nome_nodo);
                 Text textNode = documento_principal.createTextNode(valor);
                 newElement.appendChild(textNode);
                 tagPai.appendChild(newElement) ;
              }catch(Exception e){}

              return newElement;
           }

           public static void main (String Args[])
           {
              AcessoXML acesso = new AcessoXML(".\\implement\\implement.xml");
              //System.out.println(acesso.Mostra());

              Element tag = acesso.load_elemento_atributo("IMPLEMENTACAO", "VERSAO",
"flg_ativo", "1");
              if (tag != null) {
                   System.out.println(tag.getNodeName());

                        int length = 0;
                        if (tag.getAttributes() != null)
                        {
                             length = tag.getAttributes().getLength();
                        }

                        Attr attributes[] = new Attr[length];
                        for (int j = 0; j < length; j++)
                        {
                             Attr attribute = (Attr)tag.getAttributes().item(j);
                             attributes[j] = attribute;
                             System.out.println("
"+attribute.getNodeName()+"=\""+attribute.getNodeValue()+"\"");
                        }

                        acesso.Grava();
              }
              else
                   System.out.println("Nenhuma Versão Ativa.");
           }

        }
```

InterfaceXML.java

```
        package configuracao;
        import org.w3c.dom.*;
```

```
public class InterfaceXML {
	private String nom_arquivo_config = ".\\configuracao\\configuracao.xml";
	private AcessoXML acesso;
	private Element versao_ativa;
	private PontoVariacao vet_ponto_variacao[];

	public InterfaceXML()
	{
	acesso = new AcessoXML(nom_arquivo_config);
	versao_ativa = acesso.load_elemento_atributo("IMPLEMENTACAO", "VERSAO",
"flg_ativo", "1");
	if (versao_ativa!= null)
	{
	    NodeList pontos_variacao = acesso.load_filho_elemento(versao_ativa, "PV");
	    vet_ponto_variacao = new PontoVariacao[pontos_variacao.getLength()];
	    for (int i = 0; i < pontos_variacao.getLength(); i++)
	    {
			String classe = acesso.load_valor_nodo(pontos_variacao.item(i), "CLASSE");
			String metodo = acesso.load_valor_nodo(pontos_variacao.item(i),
"METODO");
			String variacao = acesso.load_valor_nodo(pontos_variacao.item(i),
"VARIACAO");

			vet_ponto_variacao[i] = new PontoVariacao(classe, metodo, variacao);
	    }
	}
	    }


	public PontoVariacao BuscaPontoVariacao(String nom_classe, String nom_metodo)
	{
		PontoVariacao ret = null;

		if (vet_ponto_variacao != null)	{
			for (int i = 0; i < vet_ponto_variacao.length; i++)	{
				PontoVariacao pt_var = vet_ponto_variacao[i];

				if (pt_var.getNomeClasse().equals(nom_classe) &&
pt_var.getNomePontoVariacao().equals(nom_metodo))		{
					ret = pt_var;
				}

			}
		}

		return ret;
	}

}
```

PontoVariacao.java

```
package configuracao;

public class PontoVariacao {
	private String nome_classe;
	private String nome_ponto_variacao;
	private String nome_variacao;

	PontoVariacao(String ent_classe, String ent_ponto_variacao, String ent_variacao)
```

```
		{
			nome_classe = ent_classe;
			nome_ponto_variacao = ent_ponto_variacao;
			nome_variacao = ent_variacao;
		}

		public String getNomeClasse()
		{
			return nome_classe;
		}

		public String getNomePontoVariacao()
		{
			return nome_ponto_variacao;
		}

		public String getNomeVariacao()
		{
			return nome_variacao;
		}
	}
```

Variacao.java

```
	/*
	 * Created on 17/04/2006
	 *
	 * TODO To change the template for this generated file go to
	 * Window - Preferences - Java - Code Style - Code Templates
	 */
	package configuracao;

	/**
	 * @author Cristiano
	 *
	 * TODO To change the template for this generated type comment go to
	 * Window - Preferences - Java - Code Style - Code Templates
	 */
	public class Variacao {
		private String nome_classe_chama;
		private String nome_metodo_chama;

		public Variacao (String ent_classe, String ent_metodo){
			nome_classe_chama = ent_classe;
			nome_metodo_chama = ent_metodo;
		}

		public String Retorno(){
			String retorno_variacao = new String();
			int cont = 0;
			InterfaceXML int_xml = new InterfaceXML();

			PontoVariacao busca_ponto_variacao =
int_xml.BuscaPontoVariacao(nome_classe_chama, nome_metodo_chama);
			if (busca_ponto_variacao != null)
			{
				String nom_metodo_variacao =
busca_ponto_variacao.getNomeVariacao();
				retorno_variacao = nom_metodo_variacao;
```

```
                }

                return retorno_variacao;
        }
}
```

Configuração.xml

```
<?xml version="1.0"?>
    <DOCUMENTO>
        <IMPLEMENTACAO cont_id="3">
            <VERSAO cont_id="5" flg_ativo="0" id="1">
                <NOME>Sistema 1</NOME>
                <DESCRICAO>
                  Sistema 1 - Lista de Discursao
                </DESCRICAO>
                <PV id="1">
                  <CLASSE>FactoryTipoRegistro</CLASSE>
                  <METODO>createHSTipoRegistro()</METODO>
                  <VARIACAO>registerMailSubscribtion()</VARIACAO>
                </PV>
                <PV id="3">
                  <CLASSE>FactoryFrequencia</CLASSE>
                  <METODO>createHSFrequencia()</METODO>
                  <VARIACAO>readConfigFile()</VARIACAO>
                </PV>
                <PV id="4">
                  <CLASSE>FactoryEnvio</CLASSE>
                  <METODO>createHSEnvio()</METODO>
                  <VARIACAO>postNews()</VARIACAO>
                </PV>
            </VERSAO>

            <VERSAO cont_id="9" flg_ativo="0" id="2">
                <NOME>Sistema 2</NOME>
                <DESCRICAO>
                  Sistema 2 - Grupo com Edicao e Estatística
                </DESCRICAO>
                <PV id="1">
                  <CLASSE>FactoryTipoRegistro</CLASSE>
                  <METODO>createHSTipoRegistro()</METODO>
                  <VARIACAO>registerWWWSubscription()</VARIACAO>
                </PV>
                <PV id="2">
                  <CLASSE>FactoryProtocoloRegistro</CLASSE>
                  <METODO>createHSProtocoloRegistro()</METODO>
                  <VARIACAO>protocoloHttp()</VARIACAO>
                </PV>
                <PV id="3">
                  <CLASSE>FactoryFrequencia</CLASSE>
                  <METODO>createHSFrequencia()</METODO>
                  <VARIACAO>readSubscriberData()</VARIACAO>
                </PV>
                <PV id="4">
                  <CLASSE>FactoryEnvio</CLASSE>
                  <METODO>createHSEnvio()</METODO>
                  <VARIACAO>postNews()</VARIACAO>
                </PV>
                <PV id="5">
                  <CLASSE>FactoryEditarAssinatura</CLASSE>
                  <METODO>createHSEditarAssinatura()</METODO>
```

```
                <VARIACAO>updateSubscription()</VARIACAO>
              </PV>
              <PV id="6">
                <CLASSE>FactoryRespostasEstatisticas</CLASSE>
                <METODO>createHSRespostasEstatisticas()
                </METODO>
                <VARIACAO>registerPollAnswers()</VARIACAO>
              </PV>
              <PV id="7">
                <CLASSE>FactoryResultadosEstatisticas</CLASSE>
                <METODO>createHSResultadosEstatisticas()
                </METODO>
                <VARIACAO>evalStatistics()</VARIACAO>
              </PV>
          </VERSAO>
      </IMPLEMENTACAO>

      <CONFIGURACAO cont_id="7">
          <PV id="1">
              <CLASSE>FactoryTipoRegistro</CLASSE>
              <METODO>createHSTipoRegistro()</METODO>
              <VARIACAO>registerMailSubscribtion()</VARIACAO>
              <VARIACAO>registerWWWSubscription()</VARIACAO>
          </PV>
          <PV id="2">
              <CLASSE>FactoryProtocoloRegistro</CLASSE>
              <METODO>createHSProtocoloRegistro()</METODO>
              <VARIACAO>protocoloHttp()</VARIACAO>
              <VARIACAO>protocoloHttps()</VARIACAO>
          </PV>
          <PV id="3">
              <CLASSE>FactoryFrequencia</CLASSE>
              <METODO>createHSFrequencia()</METODO>
              <VARIACAO>readConfigFile()</VARIACAO>
              <VARIACAO>readSubscriberData()</VARIACAO>
          </PV>
          <PV id="4">
              <CLASSE>FactoryEnvio</CLASSE>
              <METODO>createHSEnvio()</METODO>
              <VARIACAO>postNews()</VARIACAO>
          </PV>
          <PV id="5">
              <CLASSE>FactoryEditarAssinatura</CLASSE>
              <METODO>createHSEditarAssinatura()</METODO>
              <VARIACAO>updateSubscription()</VARIACAO>
          </PV>
          <PV id="6">
              <CLASSE>FactoryRespostasEstatisticas</CLASSE>
              <METODO>createHSRespostasEstatisticas()</METODO>
              <VARIACAO>registerPollAnswers()</VARIACAO>
          </PV>
          <PV id="7">
              <CLASSE>FactoryResultadosEstatisticas</CLASSE>
              METODO>createHSResultadosEstatisticas()</METODO>
              <VARIACAO>evalStatistics()</VARIACAO>
          </PV>
      </CONFIGURACAO>
  </DOCUMENTO>
```

3. Código Fonte do Framework de Componente de Distribuição de Mensagens de E-mail (FCOADistribuicao).

Distribuicao.aj

```
/*
 * Created on 17/04/2006
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package fcoaDistribuicao;

import configuracao.*;

/**
 * @author Cristiano
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
public aspect Distribuicao {
	pointcut createEnvio(): execution(HSEnvio FactoryEnvio.createHSEnvio());
	pointcut createFrequencia(): execution(HSFrequencia
FactoryFrequencia.createHSFrequencia());

	HSEnvio around() : createEnvio() {
		Variacao variacao_retorno = new Variacao("FactoryEnvio",
"createHSEnvio()");
		String variacoes = variacao_retorno.Retorno();
		HSEnvio envio = proceed();
		if (variacoes != null){
			if (variacoes.equals("postNews()")){
				envio = VariacaoPostNews.createVariacaoPostNews();
			}
			else{
				envio =
VariacaoPostNewsNaoDisponivel.createVariacaoPostNewsNaoDisponivel();
			}
		}
		else{
			envio =
VariacaoPostNewsNaoDisponivel.createVariacaoPostNewsNaoDisponivel();
		}

	   return envio;
	}


	HSFrequencia around() : createFrequencia() {
		Variacao variacao_retorno = new Variacao("FactoryFrequencia",
"createHSFrequencia()");
		String variacoes = variacao_retorno.Retorno();
		HSFrequencia frequencia = proceed();
		if (variacoes != null){
			if (variacoes.equals("readConfigFile()")){
```

```
                                        frequencia =
VariacaoReadConfigFile.createVariacaoReadConfigFile();
                                }
                                else if(variacoes.equals("readSubscriberData()")){
                                        frequencia =
VariacaoReadSubscriberData.createVariacaoReadSubscriberData();
                                }
                                else{
                                frequencia =
VariacaoReadFrequencyNaoDisponivel.createVariacaoReadFrequencyNaoDisponivel();
                                }
                        }
                else{
                        frequencia =
VariacaoReadFrequencyNaoDisponivel.createVariacaoReadFrequencyNaoDisponivel();
                }

                return frequencia;
        }
}
```

FactoryEnvio.java

```
/*
 * Created on 13/04/2006
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package fcoaDistribuicao;

/**
 * @author Cristiano
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
public class FactoryEnvio {
        public static HSEnvio createHSEnvio(){
                HSEnvio retorno = null;
                return retorno;
        }
}
```

HSEnvio.java

```
/*
 * Created on 13/04/2006
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package fcoaDistribuicao;

/**
 * @author Cristiano
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
```

```
 */
public interface HSEnvio {
	public void postNews();
}
```

VariacaoPostNews.java

```
/*
 * Created on 17/04/2006
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package fcoaDistribuicao;

/**
 * @author Cristiano
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
public class VariacaoPostNews {
	public static HSEnvio createVariacaoPostNews(){
		return new HSEnvio() {
			public void postNews(){
				System.out.println("Enviando Mensagens!");
			}
		};
	}

}
```

VariacaoPostNewsNaoDisponivel.java

```
/*
 * Created on 17/04/2006
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package fcoaDistribuicao;

/**
 * @author Cristiano
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
public class VariacaoPostNewsNaoDisponivel {
	public static HSEnvio createVariacaoPostNewsNaoDisponivel(){
		return new HSEnvio() {
			public void postNews(){
			}
		};
	}
}
```

FactoryFrequencia.java

```
/*
 * Created on 13/04/2006
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package fcoaDistribuicao;

/**
 * @author Cristiano
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
public class FactoryFrequencia {
	public static HSFrequencia createHSFrequencia(){
		HSFrequencia retorno = null;
		return retorno;
	}
}
```

HSFrequencia.java

```
/*
 * Created on 13/04/2006
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package fcoaDistribuicao;

/**
 * @author Cristiano
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
public interface HSFrequencia {

	public void readFrequency();
}
```

VariacaoReadConfigFile.java

```
/*
 * Created on 17/04/2006
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package fcoaDistribuicao;

/**
 * @author Cristiano
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
public class VariacaoReadConfigFile {
	 public static HSFrequencia createVariacaoReadConfigFile(){
```

```
                        return new HSFrequencia() {
                                public void readFrequency(){
                                        System.out.println("Frequencia de Postagem configurada a
nível de sistema");
                                }
                        };
                }

        }
```

VariacaoReadSubscriberData.java

```
        /*
         * Created on 17/04/2006
         *
         * TODO To change the template for this generated file go to
         * Window - Preferences - Java - Code Style - Code Templates
         */
        package fcoaDistribuicao;

        /**
         * @author Cristiano
         *
         * TODO To change the template for this generated type comment go to
         * Window - Preferences - Java - Code Style - Code Templates
         */
        public class VariacaoReadSubscriberData {
                public static HSFrequencia createVariacaoReadSubscriberData(){
                        return new HSFrequencia() {
                                public void readFrequency(){
                                        System.out.println("Frequencia de Postagem configurada a
nível de usuário");
                                }
                        };
                }
        }
```

VariacaoReadFrequencyNaoDisponivel.java

```
        /*
         * Created on 17/04/2006
         *
         * TODO To change the template for this generated file go to
         * Window - Preferences - Java - Code Style - Code Templates
         */
        package fcoaDistribuicao;

        /**
         * @author Cristiano
         *
         * TODO To change the template for this generated type comment go to
         * Window - Preferences - Java - Code Style - Code Templates
         */
        public class VariacaoReadFrequencyNaoDisponivel {
                public static HSFrequencia createVariacaoReadFrequencyNaoDisponivel(){
                        return new HSFrequencia() {
                                public void readFrequency(){
```

```
                    }
                };
            }
        }
```

4. *Código Fonte do Framework de Componente de Edição de Assinaturas (FCOAEdicao).*

Edicao.aj

```
/*
 * Created on 02/05/2006
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package fcoaEdicao;
import configuracao.*;

/**
 * @author Cristiano
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
public aspect Edicao {
        pointcut createEditarAssinatura(): execution(HSEditarAssinatura
FactoryEditarAssinatura.createHSEditarAssinatura());

        HSEditarAssinatura around() : createEditarAssinatura() {
                Variacao variacao_retorno = new Variacao("FactoryEditarAssinatura",
"createHSEditarAssinatura()");
                String variacoes = variacao_retorno.Retorno();
                HSEditarAssinatura edicao = proceed();
                if (variacoes != null){
                        if (variacoes.equals("updateSubscription()")){
                                edicao =
VariacaoEditarAssinatura.createVariacaoEditarAssinatura();
                        }
                        else{
                                edicao =
VariacaoEditarAssinaturaNaoDisponivel.createVariacaoEditarAssinaturaNaoDisponivel();
                        }
                }
                else{
                        edicao =
VariacaoEditarAssinaturaNaoDisponivel.createVariacaoEditarAssinaturaNaoDisponivel();
                }

           return edicao;
        }

}
```

FactoryEditarAssinatura.java

```
/*
 * Created on 13/04/2006
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package fcoaEdicao;

/**
 * @author Cristiano
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
public class FactoryEditarAssinatura {
	public static HSEditarAssinatura createHSEditarAssinatura(){
		HSEditarAssinatura retorno = null;
		return retorno;
	}
}
```

HSEditarAssinatura.java

```
/*
 * Created on 13/04/2006
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package fcoaEdicao;

/**
 * @author Cristiano
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
public interface HSEditarAssinatura {
	public void editarAssinatura();
}
```

VariacaoEditarAssinatura.java

```
/*
 * Created on 17/04/2006
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package fcoaEdicao;

/**
 * @author Cristiano
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
```

```
 */
public class VariacaoEditarAssinatura {
	public static HSEditarAssinatura createVariacaoEditarAssinatura(){
		return new HSEditarAssinatura() {
			public void editarAssinatura(){
				System.out.println("Edição de Assinatura ativo.");
			}
		};
	}

}
```

VariacaoEditarAssinaturaNaoDisponivel.java

```
/*
 * Created on 17/04/2006
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package fcoaEdicao;

/**
 * @author Cristiano
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
public class VariacaoEditarAssinaturaNaoDisponivel {
	public static HSEditarAssinatura createVariacaoEditarAssinaturaNaoDisponivel(){
		return new HSEditarAssinatura() {
			public void editarAssinatura(){

			}
		};
	}

}
```

5. Código Fonte do Framework de Componente de Estatística de Assinaturas (FCOAEstatistica).

Estatistica.aj

```
/*
 * Created on 02/05/2006
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package fcoaEstatistica;
import configuracao.*;
```

```
	/**
	 * @author Cristiano
	 *
	 * TODO To change the template for this generated type comment go to
	 * Window - Preferences - Java - Code Style - Code Templates
	 */
	public aspect Estatisticas {
		pointcut createRespostasEstatisticas(): execution(HSRespostasEstatisticas
FactoryRespostasEstatisticas.createHSRespostasEstatisticas());
		pointcut createResultadosEstatisticas(): execution(HSResultadosEstatisticas
FactoryResultadosEstatisticas.createHSResultadosEstatisticas());

		HSRespostasEstatisticas around() : createRespostasEstatisticas() {
			Variacao variacao_retorno = new Variacao("FactoryRespostasEstatisticas",
"createHSRespostasEstatisticas()");
			String variacoes = variacao_retorno.Retorno();
			HSRespostasEstatisticas respostas = proceed();
			if (variacoes != null){
				if (variacoes.equals("registerPollAnswers()")){
					respostas =
VariacaoRespostasEstatisticas.createVariacaoRespostasEstatisticas();
				}
				else{
					respostas =
VariacaoRespostasEstatisticasNaoDisponivel.createVariacaoRespostasEstatisticasNaoDisponivel();
				}
			}
			else{
				respostas =
VariacaoRespostasEstatisticasNaoDisponivel.createVariacaoRespostasEstatisticasNaoDisponivel();
			}

		   return respostas;
		}

		HSResultadosEstatisticas around() : createResultadosEstatisticas() {
			Variacao variacao_retorno = new Variacao("FactoryResultadosEstatisticas",
"createHSResultadosEstatisticas()");
			String variacoes = variacao_retorno.Retorno();
			HSResultadosEstatisticas resultados = proceed();
			if (variacoes != null){
				if (variacoes.equals("evalStatistics()")){
					resultados =
VariacaoResultadosEstatisticas.createVariacaoResultadosEstatisticas();
				}
				else{
					resultados =
VariacaoResultadosEstatisticasNaoDisponivel.createVariacaoResultadosEstatisticasNaoDisponivel();
				}
			}
			else{
				resultados =
VariacaoResultadosEstatisticasNaoDisponivel.createVariacaoResultadosEstatisticasNaoDisponivel();
			}

		   return resultados;
		}

	}
```

FactoryRespostasEstatisticas.java

```
/*
 * Created on 13/04/2006
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package fcoaEstatistica;

/**
 * @author Cristiano
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
public class FactoryRespostasEstatisticas {
	public static HSRespostasEstatisticas createHSRespostasEstatisticas(){
		HSRespostasEstatisticas retorno = null;
		return retorno;
	}
}
```

HSRespostasEstatisticas.java

```
/*
 * Created on 13/04/2006
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package fcoaEstatistica;

/**
 * @author Cristiano
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
public interface HSRespostasEstatisticas {
	public void respostasEstatisticas();
}
```

VariacaoRespostasEstatisticas.java

```
/*
 * Created on 17/04/2006
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package fcoaEstatistica;

/**
 * @author Cristiano
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
```

```
 */
public class VariacaoRespostasEstatisticas {
	public static HSRespostasEstatisticas createVariacaoRespostasEstatisticas(){
		return new HSRespostasEstatisticas() {
			public void respostasEstatisticas(){
				System.out.println("Poll de Respostas Estatisticas ativo.");
			}
		};
	}

}
```

VariacaoRespostasEstatisticasNaoDisponivel.java

```
/*
 * Created on 17/04/2006
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package fcoaEstatistica;

/**
 * @author Cristiano
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
public class VariacaoRespostasEstatisticasNaoDisponivel {
	public static HSRespostasEstatisticas
createVariacaoRespostasEstatisticasNaoDisponivel(){
		return new HSRespostasEstatisticas() {
			public void respostasEstatisticas(){

			}
		};
	}

}
```

FactoryResultadosEstatisticas.java

```
/*
 * Created on 13/04/2006
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package fcoaEstatistica;

/**
 * @author Cristiano
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
public class FactoryResultadosEstatisticas {
	public static HSResultadosEstatisticas createHSResultadosEstatisticas(){
		HSResultadosEstatisticas retorno = null;
		return retorno;
```

```
	}
}
```

HSResultadosEstatisticas.java

```
/*
 * Created on 13/04/2006
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package fcoaEstatistica;

/**
 * @author Cristiano
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
public interface HSResultadosEstatisticas {
	public void resultadosEstatisticas();
}
```

VariacaoResultadosEstatisticas.java

```
/*
 * Created on 17/04/2006
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package fcoaEstatistica;

/**
 * @author Cristiano
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
public class VariacaoResultadosEstatisticas {
	public static HSResultadosEstatisticas createVariacaoResultadosEstatisticas(){
		return new HSResultadosEstatisticas() {
			public void resultadosEstatisticas(){
				System.out.println("Resultados Estatisticas ativo.");
			}
		};
	}

}
```

VariacaoResultadosEstatisticasNaoDisponivel.java

```
/*
 * Created on 17/04/2006
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package fcoaEstatistica;
```

```
/**
 * @author Cristiano
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
public class VariacaoResultadosEstatisticasNaoDisponivel {
	public static HSResultadosEstatisticas
createVariacaoResultadosEstatisticasNaoDisponivel(){
		return new HSResultadosEstatisticas() {
			public void resultadosEstatisticas(){

			}
		};
	}

}
```

6. Código Fonte do Framework de Componente de Registro de Assinaturas (FCOARegistro).

Registro.aj

```
/*
 * Created on 02/05/2006
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package fcoaRegistro;
import configuracao.*;

/**
 * @author Cristiano
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
public aspect Registro {
	pointcut createTipoRegistro(): execution(HSTipoRegistro
FactoryTipoRegistro.createHSTipoRegistro());
	pointcut createProtocoloRegistro(): execution(HSProtocoloRegistro
FactoryProtocoloRegistro.createHSProtocoloRegistro());

	HSTipoRegistro around() : createTipoRegistro() {
		Variacao variacao_retorno = new Variacao("FactoryTipoRegistro",
"createHSTipoRegistro()");
		String variacoes = variacao_retorno.Retorno();
		HSTipoRegistro registro = proceed();
		if (variacoes != null){
			if (variacoes.equals("registerMailSubscribtion()")){
				registro =
VariacaoTipoRegistroEmail.createVariacaoTipoRegistroEmail();
			}
```

```
                                        else if(variacoes.equals("registerWWWSubscription()")){
                                                registro =
VariacaoTipoRegistroWWW.createVariacaoTipoRegistroWWW();
                                        }
                                        else{
                                                registro =
VariacaoTipoRegistroNaoDisponivel.createVariacaoTipoRegistroNaoDisponivel();
                                        }
                                }
                                else{
                                        registro =
VariacaoTipoRegistroNaoDisponivel.createVariacaoTipoRegistroNaoDisponivel();
                                }

                        return registro;
                }

                HSProtocoloRegistro around() : createProtocoloRegistro() {
                        Variacao variacao_retorno = new Variacao("FactoryProtocoloRegistro",
"createHSProtocoloRegistro()");
                        String variacoes = variacao_retorno.Retorno();
                        HSProtocoloRegistro protocolo = proceed();
                        if (variacoes != null){
                                if (variacoes.equals("protocoloHttp()")){
                                        protocolo =
VariacaoProtocoloRegistroHttp.createVariacaoProtocoloRegistroHttp();
                                }
                                else if(variacoes.equals("protocoloHttps()")){
                                        protocolo =
VariacaoProtocoloRegistroHttps.createVariacaoProtocoloRegistroHttps();
                                }
                                else{
                                        protocolo =
VariacaoProtocoloRegistroNaoDisponivel.createVariacaoProtocoloRegistroNaoDisponivel();
                                }
                        }
                        else{
                                protocolo =
VariacaoProtocoloRegistroNaoDisponivel.createVariacaoProtocoloRegistroNaoDisponivel();
                        }

                        return protocolo;
                }
        }
```

FactoryProtocoloRegistro.java

```
/*
 * Created on 13/04/2006
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package fcoaRegistro;

/**
 * @author Cristiano
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
```

```
 */
public class FactoryProtocoloRegistro {
	public static HSProtocoloRegistro createHSProtocoloRegistro(){
		HSProtocoloRegistro retorno = null;
		return retorno;
	}
}
```

HSProtocoloRegistro.java

```
/*
 * Created on 13/04/2006
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package fcoaRegistro;

/**
 * @author Cristiano
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
public interface HSProtocoloRegistro {
	public void protocoloRegistro();
}
```

VariacaoProtocoloRegistroHttp.java

```
/*
 * Created on 17/04/2006
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package fcoaRegistro;

/**
 * @author Cristiano
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
public class VariacaoProtocoloRegistroHttp {
	 public static HSProtocoloRegistro createVariacaoProtocoloRegistroHttp(){
		return new HSProtocoloRegistro() {
			public void protocoloRegistro(){
				System.out.println("Utilizando protocolo http.");
			}
		};
	}

}
```

VariacaoProtocoloRegistroHttps.java

```
/*
 * Created on 17/04/2006
```

```
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package fcoaRegistro;

/**
 * @author Cristiano
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
public class VariacaoProtocoloRegistroHttps {
	public static HSProtocoloRegistro createVariacaoProtocoloRegistroHttps(){
		return new HSProtocoloRegistro() {
			public void protocoloRegistro(){
				System.out.println("Utilizando protocolo https.");
			}
		};
	}

}
```

VariacaoProtocoloRegistroNaoDisponivel.java

```
/*
 * Created on 17/04/2006
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package fcoaRegistro;

/**
 * @author Cristiano
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
public class VariacaoProtocoloRegistroNaoDisponivel {
	public static HSProtocoloRegistro createVariacaoProtocoloRegistroNaoDisponivel(){
		return new HSProtocoloRegistro() {
			public void protocoloRegistro(){

			}
		};
	}

}
```

FactoryTipoRegistro.java

```
/*
 * Created on 13/04/2006
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package fcoaRegistro;
```

```
/**
 * @author Cristiano
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
public class FactoryTipoRegistro {
	public static HSTipoRegistro createHSTipoRegistro(){
		HSTipoRegistro retorno = null;
		return retorno;
	}
}
```

HSTipoRegistro.java

```
/*
 * Created on 13/04/2006
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package fcoaRegistro;

/**
 * @author Cristiano
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
public interface HSTipoRegistro {
	public void registrarAssinatura();
}
```

VariacaoTipoRegistroEmail.java

```
/*
 * Created on 17/04/2006
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package fcoaRegistro;

/**
 * @author Cristiano
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
public class VariacaoTipoRegistroEmail {
	 public static HSTipoRegistro createVariacaoTipoRegistroEmail(){
		return new HSTipoRegistro() {
			public void registrarAssinatura(){
				System.out.println("Registro de Assinatura via E-mail.");
			}
		};
	}

}
```

VariacaoTipoRegistroWWW.java

```
/*
 * Created on 17/04/2006
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package fcoaRegistro;

/**
 * @author Cristiano
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
public class VariacaoTipoRegistroWWW {
	public static HSTipoRegistro createVariacaoTipoRegistroWWW(){
		return new HSTipoRegistro() {
			public void registrarAssinatura(){
				System.out.println("Registro de Assinatura via www.");
			}
		};
	}

}
```

VariacaoTipoRegistroNaoDisponivel.java

```
/*
 * Created on 17/04/2006
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package fcoaRegistro;

/**
 * @author Cristiano
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
public class VariacaoTipoRegistroNaoDisponivel {
	public static HSTipoRegistro createVariacaoTipoRegistroNaoDisponivel(){
		return new HSTipoRegistro() {
			public void registrarAssinatura(){

			}
		};
	}

}
```