FUNDAÇÃO DE ENSINO "EURÍPIDES SOARES DA ROCHA"
CENTRO UNIVERSITÁRIO EURÍPIDES DE MARÍLIA – UNIVEM
FACULDADE DE CIÊNCIA DA COMPUTAÇÃO

MARCOS DE MORAES BRASIL

UMA ANÁLISE DE ALGUMAS ABORDAGENS DE MODELAGEM ORIENTADAS A ASPECTOS

MARÍLIA
2007

MARCOS DE MORAES BRASIL

UMA ANÁLISE DE ALGUMAS ABORDAGENS DE MODELAGEM ORIENTADAS A ASPECTOS.

"Monografia apresentada como requisito do Trabalho de Conclusão de Curso, orientada pelo professor Dr. Valter Vieira de Camargo."

MARÍLIA
2007

MARCOS DE MORAES BRASIL

UMA ANÁLISE DE ALGUMAS ABORDAGENS DE MODELAGEM ORIENTADAS A ASPECTOS

Banca Examinadora da dissertação apresentada ao Programa de Bacharelado da Univem, para obtenção do Título de Bacharel em Ciência da Computação.

Resultado:_____

ORIENTADOR: Prof. Dr _____

1º EXAMINADOR:_____

2 EXAMINADOR:_____

MARÍLIA,.....de .... de 2007

AGRADECIMENTOS

Dedico principalmente a minha avó que sempre torceu por mim com os olhos cheio de brilho, mesmo não estando mais entre nós ainda esta feliz seja onde for.

Agradeço primeiramente aos meus pais e família que a todo momento me deram o apoio e a possibilidade de concretizar esse sonho, sempre acreditando e persistindo em manter os esforços interruptamente durante esses longos anos longe de casa.

A minha namorada que me suportou em época de prova, mesmo eu sendo insuportável, ela agüentou e colaborou para eu atravessar essas turbulentas semanas.

Aos professores Valter Vieira, Edmundo Sérgio Spoto, Fátima de Lourdes dos Santos N. Marques, que sempre me deram não só o proposto pelo curso, mas também a nunca desistir de um sonho, sem eles já teria desistido no meio do caminho.

Sem esquecer dos amigos de sala que não ficam somente nisso tais como, Tadeu Friol, Marco Aurélio Pioto, Evandro Pontelli Navarro, Rodrigo Espinosa, Fabio Tebalde, Bebe e Jun, que além de colegas de sala são como irmãos prontos para toda hora, seja na hora da descontração ou na hora de pegar firme em estudos.

E claro a Deus, por me iluminar e me triar para esse desfecho, que para mim é mais do que eu merecia. Agradeço de coração a todos vocês!

MORAES BRASIL, Marcos. **Uma Comparação entre Abordagens de Modelagem Orientada a Aspectos.** 2007 Monografia (Bacharelado em Ciências da Computação) – Centro Universitário Eurípedes de Marília, Fundação de Ensino "Eurípedes Soares da Rocha", Marília, 2007.

RESUMO

Nesta monografia é apresentada uma comparação entre algumas abordagens de modelagem para sistemas no paradigma orientado a aspectos. O objetivo é complementar um trabalho anterior que já realizou esse estudo incluindo três novas abordagens para esse paradigma. Embora haja várias linguagens para a implementação de sistemas orientados a aspectos e, inclusive uma que se destaca como a principal a AspectJ, o mesmo não ocorre com abordagens de modelagem para esse paradigma. Assim, a comparação mostrada nesta monografia visa a auxiliar um engenheiro de software a optar por uma determinada abordagem de modelagem durante o desenvolvimento de um sistema orientado a aspectos. A comparação foi realizada mediante critérios já definidos em um trabalho anterior. Para a realização do estudo, foi utilizado um sistema hipotético de pedidos como estudo de caso para auxiliar nas comparações efetuadas.

Palavras-chave: Modelagem Orientada a Aspectos, UML , AspectJ.

MORAES BRASIL, Marcos. **Uma Comparação entre Abordagens de Modelagem Orientada a Aspectos.** 2007 Monografia (Bacharelado em Ciências da Computação) – Centro Universitário Eurípedes de Marília, Fundação de Ensino "Eurípedes Soares da Rocha", Marília, 2007.

ABSTRACT

In this work a comparison is presented among some modeling approaches for aspect-oriented systems. The aim is to improve a previous work including three new approaches for that new paradigm. Although there are several languages for implementing aspect-oriented systems and, besides one that stands out as to main, the same doe not happen with modeling approaches for that paradigm. Thus, the comparison shown in this work seeks to aid a software engineer in choosing a specific modeling approach during the development of an aspect-oriented system. The comparison was made by means of some criteria defined in a previous work. In this study, a hypothetical order system was used as case study to aid in the comparisons.

Keywords: Aspect Oriented Programming, UML, AspectJ

SUMÁRIO

INTRODUÇÃO

Contexto

A POA (Programação Orientada a Aspectos) foi criada por Gregor Kiczales na década de 90, mais especificamente no ano de 1997 com o intuito de prover uma programação que modularizasse os trechos de códigos espalhados e ou entrelaçados na aplicação em uma nova construção denominada Aspecto (KICZALES *et al*., 1997).

Identificar, separar e compor características transversais de alto nível de abstração provê melhor entendimento do problema e da solução, rastreabilidade entre os diversos modelos construídos durante o desenvolvimento e entre as diversas características e reusabilidade entre as fases de desenvolvimento (Isabel Brito *et al.,* 2004).

AspectJ é a linguagem mais utilizada para a Programação Orientada a Aspectos (KICZALES *et al*., 2001), porém existem outras linguagens tais como, o AspectC e o HyperJ, mas este trabalho é utilizado o AspectJ por ser mais difundido. Essa padronização não ocorre com as abordagens de modelagem; embora o número de propostas seja grande, não há uma determinada abordagem que se destaque.

Motivação

A motivação para a realização desse trabalho é a inexistência de uma abordagem de modelagem padrão para programação orientada a aspectos. Aliado a isso, também não é encontrado na literatura um trabalho que realize comparação entre propostas de modelagem.

Objetivos

Este trabalho tem como objetivo principal, fornecer subsídios para que o desenvolvedor possa escolher uma determinada abordagem de modelagem dependendo do contexto do projeto que está sendo conduzido. Outro objetivo está no referencial teórico que o trabalho irá deixar que pode ser utilizado para o desenvolvimento de outros trabalhos.

Organização do Texto

Esta monografia está estruturada da seguinte forma: No Capítulo 1 são apresentados alguns conceitos fundamentais sobre POA, AspectJ e UML (Linguagem de Modelagem Unificada); no Capítulo 2 são apresentadas as abordagens de modelagem estudadas; no Capítulo 3 são apresentados os critérios de comparação utilizados e no Capítulo 4 é feita a comparação entre essas abordagens e em seguida é feita a conclusão do trabalho realizado.

CAPÍTULO 1

1.1 Considerações Iniciais

Neste capítulo o objetivo é fornecer conceitos básicos de Programação Orientada a aspectos, da linguagem AspectJ e da linguagem de modelagem UML.

1.2 Programação Orientada a Aspectos

A programação orientada a aspectos é uma técnica de programação que visa a separar os interesses base dos interesses transversais de um sistema. Apesar de existir outras linguagens, a mais difundida é a AspectJ, que é uma extensão da Java.

Czarnecki (2001) comenta que a necessidade de manipular um requisito importante de cada vez, durante o desenvolvimento de um sistema, é chamado de princípio da separação de interesses. A diferença da POA para os demais paradigmas mesmo os que oferecem unidades modulares e procedimentos para a organização do sistema é que a POA oferece mecanismos quer permitem a modularização de interesses que abrangem mais de um componente funcional, interesses esses separados por fragmentos de códigos denominados aspectos.

Elrad *et al* (2001) afirmam que a programação orientada a aspectos consiste na separação dos interesses de um sistema em unidades modulares e posterior composição (*weaving*) desses módulos em um sistema completo.

A Programação Orientada a Aspectos provê mecanismos de linguagem que explicitamente capturam a estrutura de entrecorte (*crosscutting)*, alcançando, assim, os benefícios de melhor modularidade, tais como: código mais simples, mais fácil de manter e alterar e conseqüentemente, aumento da reusabilidade do código.

Pontos de Junção(*Join Points)* são os pontos na execução do sistema, onde serão feitos os entrecortes para a utilização dos aspectos. Vários pontos de junção podem ser agrupados para formar um conjunto de junção.

Um adendo (*advice)* é executado em um conjunto de junção. Um adendo possui três tipos básicos: *Before, After e Around*. No *before*, o trecho de código é executado antes do ponto de junção; no after o trecho de código é executado depois do ponto de junção e no *around* o trecho de código é executado simultaneamente ao ponto de junção.

Um conjunto de junção (*Point Cuts)* é formado por vários pontos de junção. Quando a execução de um programa chegar em um desses pontos de junção declarados no conjunto de junção um adendo é executado. Isso permite ao programador descrever aonde e quando um trecho de código deve ser executado durante a execução do programa.

As declarações Inter-Tipo permitem ao programador adicionar métodos, campos ou interfaces nas classes existentes utilizando o aspecto.

A unidade modular "aspecto" encapsula conjuntos de junção, adendos e declarações inter-tipos em uma unidade modular de implementação, que pode alterar a estrutura estática ou dinâmica de um programa. A estrutura estática pode ser alterada usando as declarações inter-tipos citadas acima. A estrutura dinâmica ocorre em tempo de execução por meio dos pontos de junção que são selecionados pelos conjuntos de junção e por meio da edição dos adendos.

1.3 A Linguagem AspectJ

A linguagem AspectJ, que é uma extensão da linguagem Java, é a linguagem mais difundida atualmente para implementar programas orientados a aspectos(KICZALES *et al*., 2001). Com esta linguagem é possível implementar separadamente os interesses transversais e a parte base do sistema. A linguagem AspectJ possui alguns elementos básicos que são os pontos de junção, os conjuntos de junção, os adendos, as declarações inter-tipos e os aspectos.

Com essa linguagem é possível definir vários pontos de junção dentro de um programa, que por sua vez, serão agrupados em um conjunto de junção e assim, os pontos de junção poderão ser afetados pelo aspecto. Pontos de junção podem ser uma execução de métodos e construtores, recebimento de chamada a construtores, acesso a campos e execução de manipuladores de exceção.

Na Figura 1.1 é mostrada a estrutura de um programa na AspectJ com a composição, onde temos as classes como componentes e os aspectos então é feita a composição através dos pontos de junção para então serem executados.

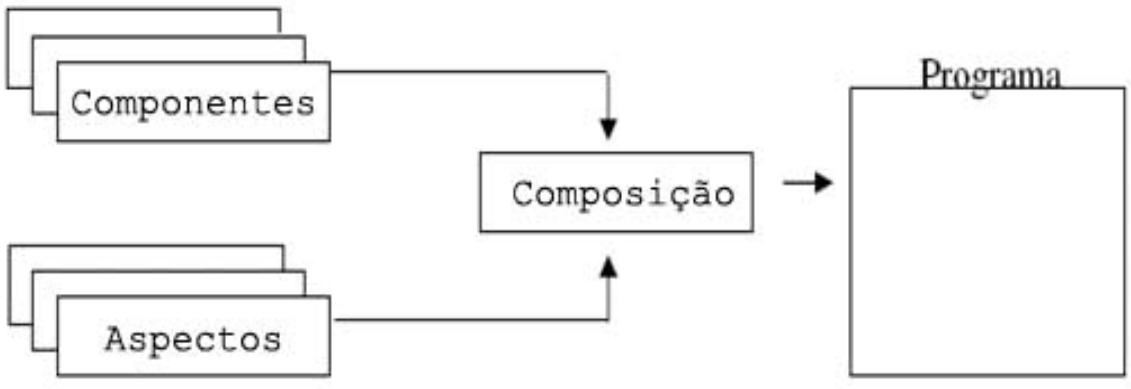


Figura 1.1 - Estrutura de programa no AspectJ (Monteiro,2004)

Nas Figura 1.2 e 1.3 ilustra-se um trecho de código em AspectJ que implementa um interesse transversal entre uma classe e um aspecto, esse interesse faz com que o aspecto

Trace entrecorte toda vez que o método *set* for chamado. Na Figura 1.2 é possível observar a existência de uma classe chamada Pedido que por sua vez possui um método *main* e dentro desse métodos *set*, esses métodos serão entrecortados pelo aspecto e o mesmo executará seus adendos. A Figura 1.3 ilustra-se o código do aspecto *Trace*, que entrecorta a chamada/execução dos métodos *set* da classe Cliente. Assim quando esses métodos são chamados o aspecto *Trace* através dos adendos *before* e *after* são executados. Os adendos tem como finalidade exibir mensagens antes e depois que o método é executado.

```
public class Pedido {
            public static void main(String args[]) {
             Cliente c = new Cliente();
            c.setNome("Marcos");
            System.out.println("Nome: "+ c.getNome());
            c.setCodigo(1);
            System.out.println("Código: "+c.getCodigo());
            c.setCidade("Pompéia");
            System.out.println("Cidade: "+c.getCidade());
        }
    }
```

Figura 1.2 Trecho de código da classe Pedido.

```
public aspect Trace {
     public pointcut pt1(): call( void **.set*(..));

    before(): pt1() {
      System.out.println("Ponto 1 Before (antes) do método set");
    }
    after(): pt1() {
        System.out.println("Ponto 1 After( depois ) do método set");
    }
}
```

Figura 1.3 Trecho de código da classe Trace.

1.4 Linguagem de Modelagem UML

A UML (Unified Modeling Language) é uma linguagem de modelagem para especificar, visualizar, construir e documentar os artefatos de um processo de software(OMG, 1999).

Com UML podemos fazer uma modelagem visual de maneira que os relacionamentos entre os componentes do sistema sejam mais bem visualizados e compreendidos e documentados.

A UML é a junção de três das mais conceituadas linguagens de modelagem orientadas a objetos (Booch de Grady, OOSE de Jacobson e o OMT de Rumbaugh).

Um detalhe muito interessante da Uml é que possui uma forma de estendê-la para domínios específicos. Isso pode ser feito por meio de Perfis(*Profiles*). Um Perfil em UML é um nome dado a um grupo predefinido de estereótipos (s*tereotypes)*, etiquetas com valor(*tagged values)* e restrições (*constraints)* que conjuntamente especializam e configuram o UML para um determinado domínio de aplicação ou para um determinado processo de desenvolvimento (OMG, 1999). Estereótipos são mecanismos que estendem o vocabulário UML para suprir necessidades que não encontram-se em outros meta-elementos disponíveis, esses estereótipos podem ser aplicados em classes, relacionamentos e etc.(exemplo: `<<aspect>>, <<crosscut>>`), (BOOCH *et al*., 1999)

A abstração em UML é o principio de ignorar os assuntos não relevantes, tornando assim uma concentração maior nos assuntos principais. Abstração de procedimentos é o princípio de que qualquer operação com um efeito bem definido pode ser tratada por seus usuários como uma entidade única, mesmo que a operação seja realmente conseguida através de alguma seqüência de operações de nível mais baixo.

Encapsulamento é para omitir as informações de uma determinada entidade que diz respeito somente a ela, é fundamental que o objeto proteja seus dados, não permitindo que os usuários do objeto os acesse diretamente, mas sim através de métodos.

1.5 Considerações Finais

Este trabalho foi desenvolvido para prover uma comparação entre abordagens de modelagem em UML para o paradigma POA que ainda não possui uma modelagem padrão. Esse paradigma prevê a modularização de interesses transversais, para que através de seus pontos de junção com o código base do sistema possa ser executados os adendos estruturados dentro de uma unidade chamada Aspecto..

Para fazer a modelagem utilizando a UML foi preciso o estudo de Perfis para que pudesse modelar as particularidades do paradigma.

CAPÍTULO 2

2.1 Considerações Iniciais

Neste capítulo são apresentadas as extensões de modelagem em UML para a linguagem orientada a aspectos. Modelagens essas propostas por, Stein *et al.*(2002), Zakaria *et al.*(2002) e Suzuki *et al.* (1999) para que no capítulo 4 possa ser feita a comparação entre elas.

2.2 Abordagens Estudadas

2.2.1 Abordagem de Stein e outros

Stein *et al.* (2002) apresentam uma extensão da UML para os conceitos orientados a aspectos de AspectJ baseando-se nos conceitos orientados a objetos existentes na UML, ressaltando suas semelhanças e diferenças. Em sua abordagem, os relacionamentos associação da UML são identificados como representações adequadas para os pontos de junção do AspectJ. Para todos os componentes de um aspecto do AspectJ, como conjunto de junção, adendo e declarações inter-tipo assim como o próprio aspecto, a abordagem é um Perfil UML.

No exemplo utilizado por Stein(2002) ilustra-se o padrão de projeto `Observer` que é utilizado para criar um rótulo colorido (*`color label`*) que altera sua cor sempre que o botão (*`button`*) for pressionado. Na Figura 1.4 é mostrado um aspecto abstrato chamado `SubjectObserverProtocol` que possui dois conjuntos de junção abstratos, denominados "*`stateChanges`*" e um adendo chamado `advice_id01`, e duas declarações inter-tipos. Nota-se que na modelagem proposta por Stein *et al.* (2002) o aspecto é representado por uma caixa, assim como uma classe, porém na parte inferior pode-se colocar os conjuntos de junção, representados pelo estereótipo `<<pointcut>>`, os adendos que atuam sobre os conjuntos de junção definidos, representados pelo estereótipo `<<advice>>`, e também as declarações inter-tipos, que são representadas por círculos tracejados com um retângulo na parte superior direita. Dentro desse retângulo há parâmetros cujos valores devem ser informados pelo engenheiro de aplicação quando esse aspecto for concretizado. Esses valores são classes de uma aplicação específica que serão afetadas pelas declarações inter-tipos. Essa estratégia de definir parâmetros para as declarações inter-tipos é bastante utilizada

também por outras abordagens de modelagem (por exemplo, Clarke) para permitir que um aspecto seja modelado de forma genérica e depois seja acoplado a um código-base.

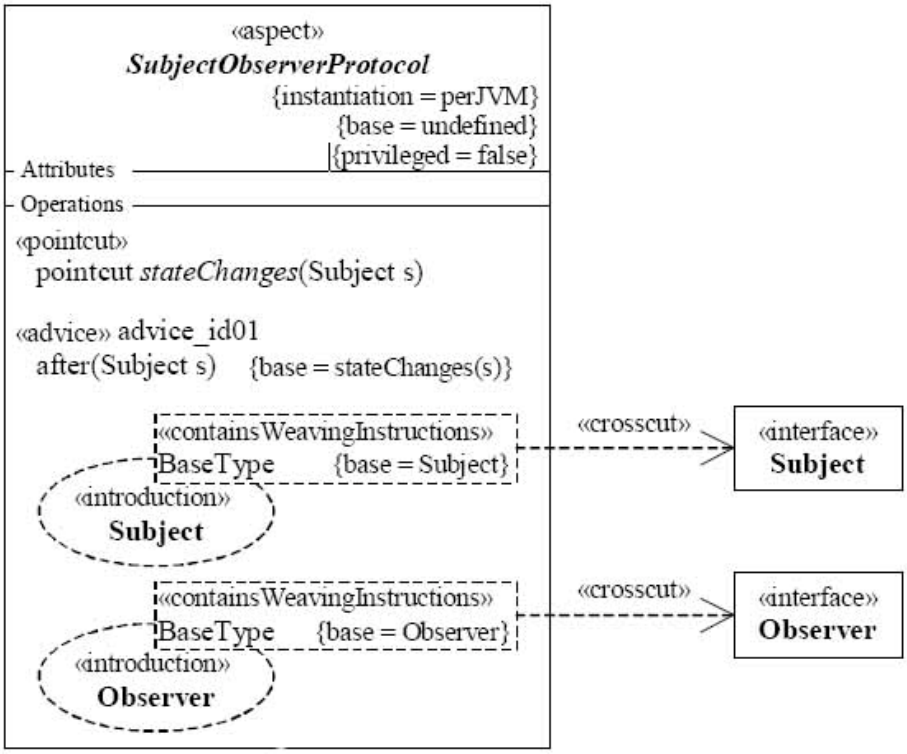


Figura1.4 – Exemplo SubjectObserverProtocol abordagem de Stein.

Na Figura 1.5 encontra-se o aspecto concreto que estende o aspecto abstrato `SubjectObserverProtocol` mostrado na Figura 1.4. Note-se que nesse aspecto o conjunto de junção abstrato definido no aspecto `SubjectObserverProtocol` foi concretizado no sentido de entrecortar todas chamadas ao método *`click`*() da classe Button. Note-se também que os parâmetros para as declarações inter-tipo também foram definidos. Na primeira declaração inter-tipo o valor informado é *button*. Isso informa que essa classe faz o papel de Subject do padrão Observer.

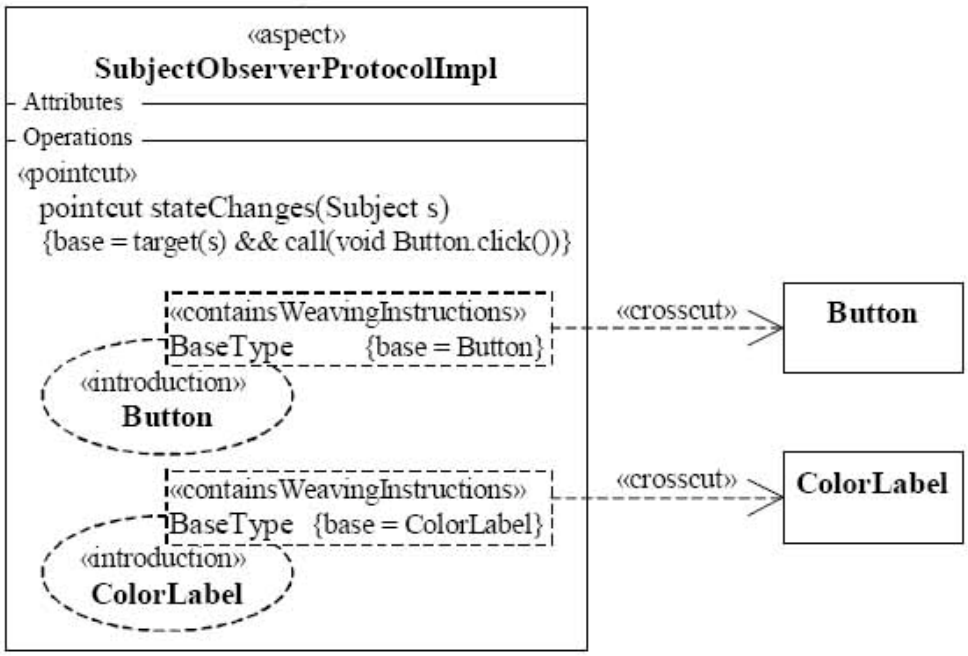


Figura1.5 – Exemplo SubjectObserverProtocolImpl abordagem de Stein.

Os aspectos são modelados por Stein *et al.* (2002) como Classes de um diagrama UML, diferenciada pelo estereótipo `<<aspect>>` .

Tendo como base a estrutura utilizada para modelagem Orientada a Objetos, Stein acrescenta o estereótipo `<<pointcut>>` dentro do aspecto. O ponto de junção fica no compartimento de operações do aspecto como uma *string* especificando seu nome.

As características transversais do aspecto são identificadas pelo estereótipo `<<advice>>` dentro da própria classe especificando o seu nome e suas aplicações no aspecto. Assim como o pontos de junção os adendos também ficam no compartimento de operação.

Os relacionamentos de entrecortes são especificados pelo estereótipo `<<crosscut>>` que indica a base onde o aspecto será entrecortado.

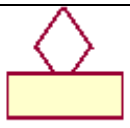
2.2.2 Abordagem de Zakaria e outros

Zakaria *et al.* (2002) utilizam uma notação para a representação de aspecto e conjuntos de junção como extensão da UML. A relação entre um aspecto e uma classe é feita por meio de relacionamento de associação, para isso é criado um conjunto de estereótipos.

A classe utiliza um estereótipo `<<aspect>>` e uma relação com o ponto de junção, representado com o estereótipo `<<has pointcut>>` e este mesmo ponto de junção tem relação com o aspecto através da estereótipo `<<has pointcut>>`.O estereotipo `<<control>>` indica a classe na qual o aspecto irá entrecortar como mostrado a Figura 1.6. Quando se tem um aspecto ligado a um outro aspecto cuja a prioridade é inferior, usa-se o estereótipo `<<dominates>>` no aspecto de maior prioridade.

Os aspectos definidos utilizando-se de elementos não existentes no modelo UML, como mostra a Tabela 1.1.

Tabela 1 Especificação do estereótipo <<aspect>>

Base UML meta-model elemento	Esterótipo	Representação	Explicação
Classe	<<aspect>>		O estereótipo <aspect>> é usado para indicar um aspecto

Os pontos de junção da modelagem são descritos com um circulo simples, com o seu nome na parte inferior, como mostra a Tabela 1.1.

Tabela 2 Especificação do estereótipo <<pointcut>>

Base UML meta-model lemento	Esterótipo	Representação	Explicação
Classe	<<Pointcut>>		Este estereótipo especifica o pointcut na classe onde um aspecto advice irá ser chamado

Como exemplo Zakaria *et al.*(2002) criam uma relação entre um aspecto chamado MoveTracking e uma classe chamada Line, entrecortarda pelo ponto de junção chamado Move.

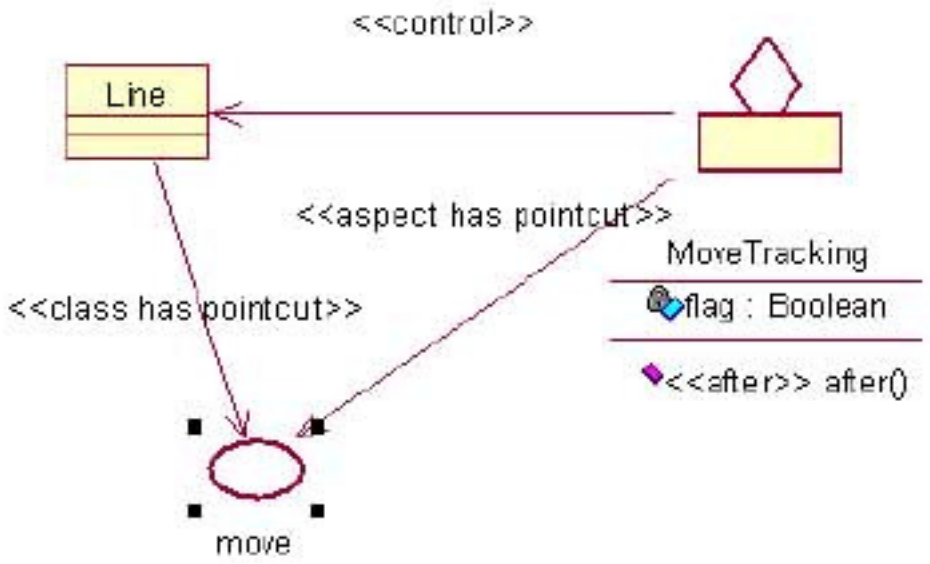


Figura 1.6 Exemplo de modelagem utilizada por Zakaria *et al* (2002).

A notação de Zakaria *et al.*(2002) ainda prevê a criação de um estereótipo para cada tipo de adendo, sendo eles `<<before>>, <<after>> e <<around>>` representando as possíveis funções a serem utilizadas no AspectJ, relacionamento entre, classe-aspecto, classe-pointcut e um aspecto-pointcut, o que poderia dificultar a modelagem de um sistema um pouco maior.

2.2.3 Abordagem de Suzuki e outros

Suzuki *et al.* (1999) utilizam novos termos para apoiar os aspectos e reutilizam um meta-modelo UML para fazer o relacionamento entre os aspectos e as classes.

Os aspectos podem conter seus atributos, métodos e relacionamentos com as classes da qual estará afetando, baseando-se na programação em AspectJ.

Para a representação de um aspecto adotou-se uma classe com o estereótipo `<<aspect>>` como mostrado a Figura 1.7. Essa classe contém uma lista de informações, contendo a classe e os métodos que o aspecto irá afetar.

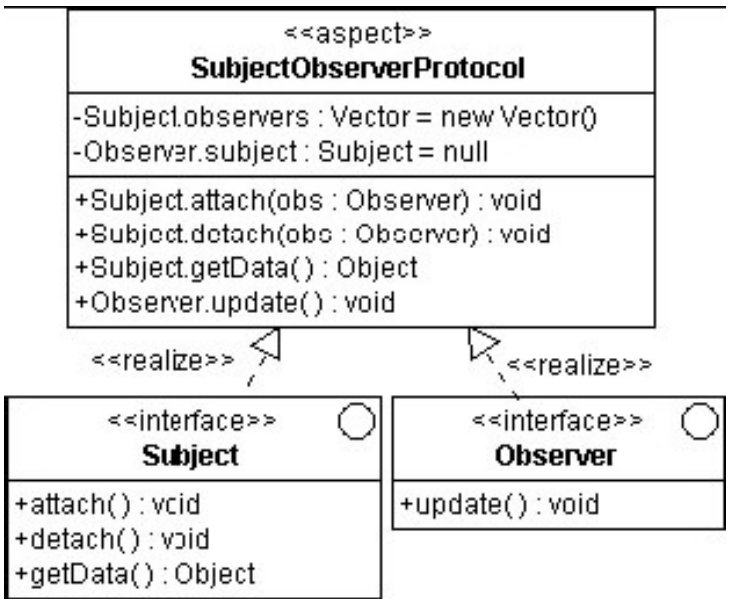


Figura 1.7 Demonstração da classe aspecto e esterótipo de relacionamento.

Os relacionamentos entre as classes e os aspectos são como relacionamentos de dependência derivados do meta-modelo de UML como mostrado na Figura 1.7. Em UML

existem três tipos de relacionamento, sendo eles, derivação, realização e refinamento, para a representação de um relacionamento entre uma classe e um aspecto, foi adotado o estereótipo <<realize>> informando que o aspecto será utilizado em uma determinada classe, como foi mostrado na Figura 1.7.

Essa extensão não possui um estereótipo próprio para os adendos, deixando assim um grande vazio e dificultando na hora de implementar a aplicação.

2.3 Considerações Finais

Cada uma das três abordagens utiliza diferentes maneiras de representação UML para POA. Stein *et al.* (2002) utilizam o estereótipo <<aspect>> na classe para representar um aspecto que afetará outras classes, o estereótipo <<pointcut>> especificando onde os adendos serão entrecortados e para a representação dos adendos utiliza-se os estereótipos <<before>> e <<after>>.

Zakaria *et al.* (2002) assim como Stein *et al.* (2002) utilizam os estereótipos <<before>> e <<after>> mas não utilizam estereótipo para especificar a classe aspecto, para relacionar um aspecto com a classe desejada utiliza um relacionamento com o ponto de junção com a legenda **<<**has pointcut**>>** e um outro relacionamento especificando qual classe esse ponto de junção afetará, esse relacionamento utiliza a legenda **<<** class has pointcut**>>** e ainda um relacionamento da classe aspecto com a classe afetada dizendo através da legenda <<control>> que a classe será afetada por determinado aspecto, Zakaria *et al*(2002). Suzuki *et al*(1999) instanciam uma classe com o estereótipo <<aspect>> relacionada com a classe afetada através de um relacionamento com a legenda <<realize>> , Suzuki *et al.*(1999) ao contrário dos anteriores não utiliza estereótipo algum para definir os adendos.

CAPÍTULO 3

3.1 Considerações Iniciais

Neste capítulo são mostrados os critérios de comparação utilizados para comparar as abordagens mostradas no capítulo anterior. Para realizar essa comparação foi feita uma modelagem de um sistema em comum para todas as abordagens.

3.2 Critérios de Comparação

Para evitar que os modelos fossem comparados de uma maneira subjetiva, foram utilizados alguns critérios de comparação. Todos os critérios de comparação utilizados são baseados nos critérios de comparação proposto no trabalho (AOSD-Europe-ULANC-9, 2005). Existem mais alguns critérios como legibilidade e semântica, entretanto eles não foram utilizados neste trabalho por serem critérios subjetivos. O critério nível de abstração irá definir se uma modelagem possui muitos detalhes (nível de abstração baixo), se possui uma quantidade média de detalhes (nível de abstração médio) ou se possui poucos detalhes (nível de abstração alto). Uma abordagem pode possuir mecanismos que representam um alto nível de abstração, mas também fornecer subsídios para analisar detalhes de mais baixo nível.

Já o critério de nível de separação de interesses determina se uma abordagem é simétrica ou assimétrica. Uma abordagem simétrica separa todos os interesses de um sistema, sem fazer distinção entre transversais e não transversais. Já uma abordagem assimétrica, só separa os interesses transversais do resto do sistema.

O critério de rastreabilidade determina se o nível de rastreabilidade da abordagem é baixo, médio ou alto. Este critério determina se a abordagem permite que o modelo seja

rastreado durante as suas três fases de desenvolvimento que são a análise o projeto e a implementação. Quanto maior a capacidade de manter um mapeamento entre os artefatos de desenvolvimento que são criados ao longo do ciclo de vida, maior será sua rastreabilidade (o que o caracterizaria como alto).

A evolução determina o nível de evolução fornecido pela abordagem. Consiste na possibilidade de se poder alterar, adicionar ou remover os artefatos existentes em um projeto. Artefatos é tudo o que é criado para se desenvolver o software (atores, estereótipos, classes, temas etc.). O critério de escalabilidade determina a adequabilidade da abordagem a medida com que o sistema adquire um grande número de componentes/classes. A abordagem deve ser apropriada tanto para grandes quanto para pequenos sistemas. A classificação de uma abordagem em relação a sua escalabilidade poderá ser baixa (fácil modelar um sistema pequeno e difícil para modelar sistemas grandes), média ou alta (fácil modelar sistemas pequenos e grandes).

Foi utilizado também dois critérios comparativos, que são eles, representatividade do entrecorte e a representatividade do aspecto(Uetanabara *et al*.,2007). Esses critérios são chamados de comparativos porque permitem comparar uma abordagem e obter uma indicação numérica de sua qualidade em cada critério.

O critério de representatividade do entrecorte mede o grau de clareza em que é possível identificar as características importantes do relacionamento de entrecorte. Esse critério visa a determinar se é possível: 1) visualizar/identificar que um aspecto afeta uma ou mais classes; 2) se é possível visualizar/identificar exatamente os métodos afetados por um aspecto; 3) se é possível visualizar/identificar se o método afetado pelo aspecto antes ou depois de sua execução/chamada e 4) se é possível visualizar/identificar se o método é afetado no contexto de chamada ou execução.

Critério de representatividade do aspecto mede o grau de clareza em que é possível identificar os componentes básicos de um aspecto e sua própria presença. Esse critério visa a determinar se: 1) é fácil visualizar e identificar a presença de um ou mais aspectos com a notação; 2) se é fácil visualizar e identificar a presença de um ou mais conjuntos de junção e 3) se é fácil visualizar e identificar a presença de um ou mais adendos. Esse critério pode ser classificado como baixo, médio ou alto. Se os três itens puderem ser identificados, o critério recebe valor “alto”. Se apenas dois puderem ser identificados o critério recebe valor “médio” e se apenas um for identificado o critério recebe valor “baixo”.

3.3 Modelagem das Diferentes Abordagens

Para que a comparação fosse feita foi proposto um sistema que controla a compra de produtos, relacionando os clientes que os compraram, número do pedido e quantidade de itens desse pedido. Um aspecto chamado Trace foi criado para rastrear o código base do sistema para entrecortar onde houvesse métodos *set*, assim que localizados, o aspecto executa seus adendos *before* e *after* para exibir mensagem antes e depois que o método é entrecortado.

Abaixo são demonstradas as modelagens do sistema de compras seguindo os conceitos de modelagem especificados por cada autor estudado.

A Figura 1.8 demonstra a modelagem do sistema de compras com suas classes do código base que são: Cliente, Pedido, Item e Produto e também um aspecto denominado `Trace`.

O aspecto `Trace` possui um estereótipo denominado `<<Aspect>>` em uma classe. Esse aspecto também inclui um conjunto de etiquetas valoradas que auxiliam a visualizar determinadas características de implementação do aspecto, por exemplo: `Instantiation, base, privileged.` Essas etiquetas recebem valores de acordo com a

necessidade do sistema. A etiqueta `Instantiation` indica o tipo de instanciação do aspecto. Há dois tipos de instanciação, o primeiro ocorre quando um único objeto é criado para o aspecto, e esse objeto entrecorta todas as operações do sistema. O segundo caso ocorre quando vários objetos de aspectos são criados, sendo que cada um deles é responsável por entrecortar um determinado ponto de junção. No caso do sistema de compras, a etiqueta `Instantiation` receberá o valor *perJVM* devido ao fato de que o aspecto irá ser iniciado uma vez para todo o sistema. A etiqueta `base` indica se existe uma classe, na faze de execução do sistema, na qual o aspecto irá entrecortar, no caso do sistema de compras não foi definida como *undefined* uma vez que não é possível identificar qual a classe o aspecto entrecortará. A etiqueta `privileged` identifica se o aspecto tem algum privilégio quando comparado a outro aspecto, por se tratar de um sistema que só tem um aspecto essa etiqueta recebe o valor *false.*

Ainda é possível ver que o aspecto contém um conjunto de operações denominados `Operations` que contém informações sobre as operações que o aspecto irá executar nas classes entrecortadas. Possui um estereótipo denominando o nome do ponto de junção e o modo que é feita a chamada no código base, denominado `<<PointCut>>`. Após especificado o ponto de junção e a chamada, o aspecto apresenta os adendos especificando seus nomes e se será executado, antes, depois ou no ato da execução do método.

Após apresentada toda a parte de implementação, foi modelado um relacionamento com o estereótipo <<CrossCut>> indicando em qual classe o aspecto entrecortará.

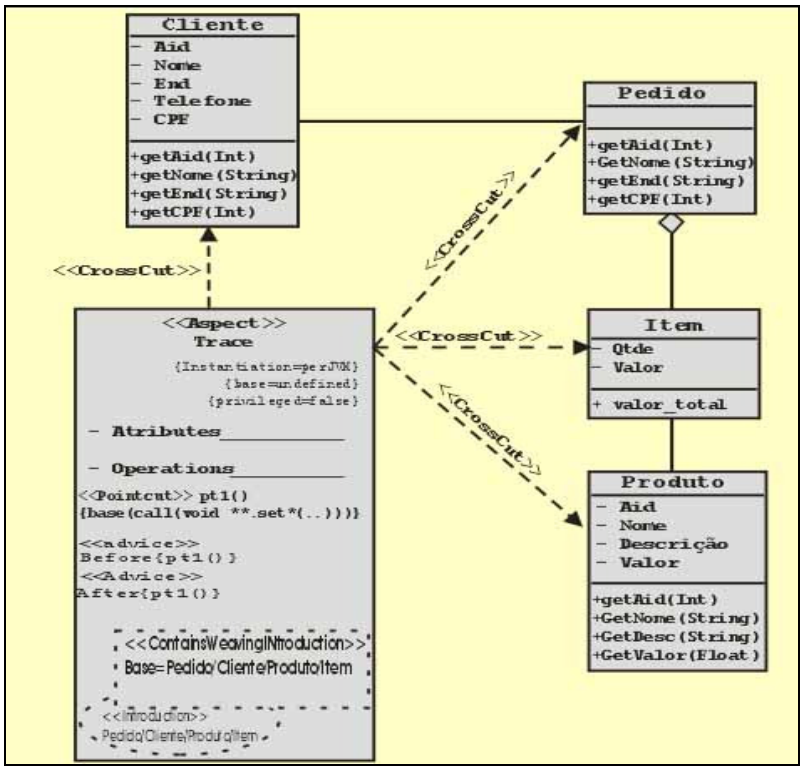


Figura 1.8 Demonstração de um sistema modelado seguindo a abordagem de Stein.

A Figura 1.9 demonstra a modelagem do sistema de compras seguindo os conceitos de Zakaria *et al.(*2002) contendo as classes do código base que são: Cliente, Pedido, Item e Produto e também um aspecto denominado `Trace`.

Para modelar um ponto de junção, foi inserido uma interface chamada PT1 com o tipo de chamada e onde o aspecto irá entrecortar.

Foi criado um relacionamento de todas as classes do código base com o ponto de junção PT1 com o estereótipo `<<Class hás PointCut>>`. Um relacionamento do aspecto Trace com o ponto de junção também foi especificado com um relacionamento demarcado com o estereótipo `<< Aspect hás PointCut >>`.

Zakaria prevê uma estensão da UML para a especificação do aspecto, no caso chamado Trace, essa notação contem além do nome do aspecto, um conjunto de informações dos adendos do aspecto.

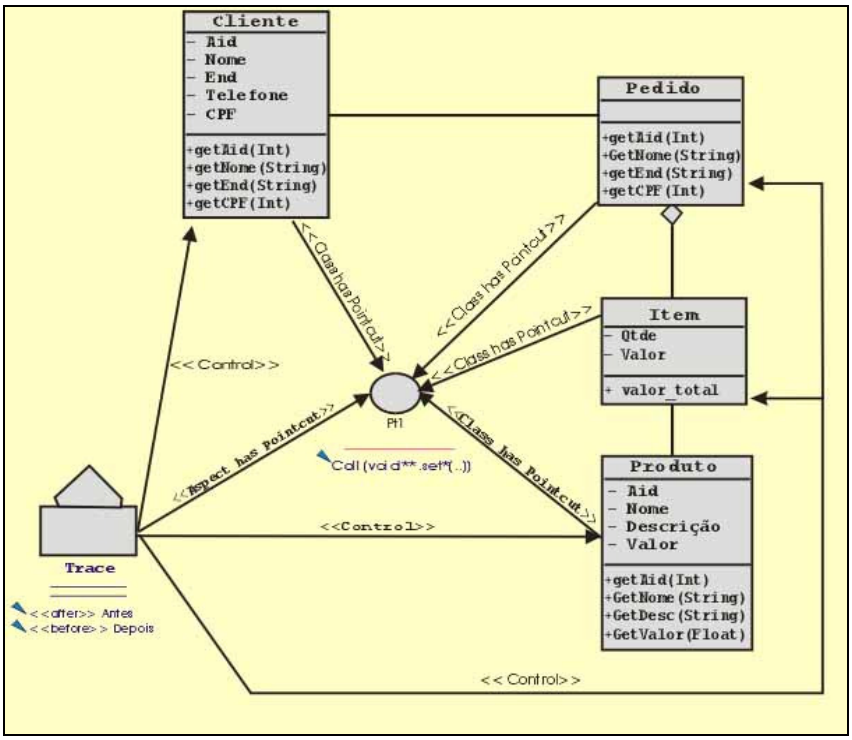


Figura 1.9 Demonstração de um sistema modelado seguindo a abordagem de Zakaria.

A Figura 2.0 demonstra a modelagem do sistema de compras seguindo os conceitos de Suzuki *et al.*(1999) contendo as classes do código base que são: Cliente, Pedido, Item e Produto e também um aspecto denominado `Trace`.

Após modelado o código base do sistema, foi introduzido uma classe com o estereótipo <<Aspect>> especificando o aspecto Trace. Foi criado um relacionamento com o estereótipo <<Realize>> indicando em quais classes o aspecto entrecortará.

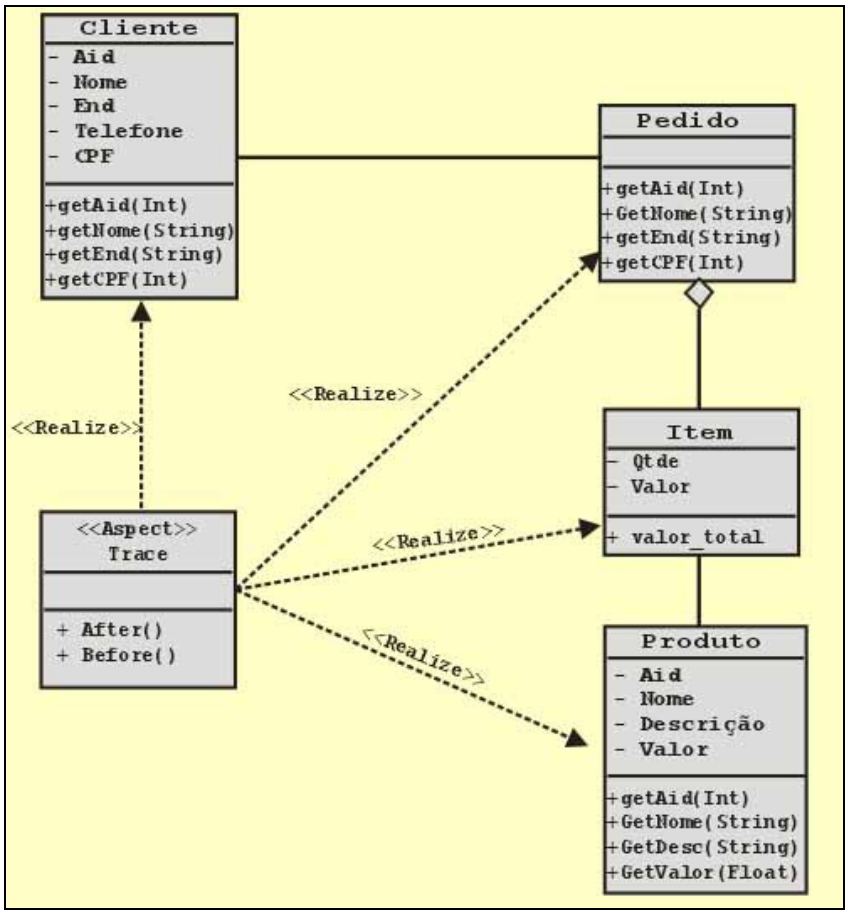


Figura 2.0 Demonstração de um sistema modelado seguindo a abordagem de Suzuki.

CAPÍTULO 4

4.1 Comparação

Nesta seção, são apresentadas as comparações entre as abordagens de modelagem mostradas no capítulo 3. As comparações foram elaboradas para que engenheiro de software possa escolher a que mais lhe convêm seguindo os critérios de análise.

A Tabela 3 exibe a comparação entre as abordagens utilizando o critério "nível de abstração".

Tabela 3 – Comparação pelo critério de nível de abstração

Abordagem	Nível de abstração
Stein *et al.* (2002)	Baixo
Zakaria *et al* (2002)	Médio
Suzuki *et al.* (1999)	Alto

A abordagem de Stein *et al*. (2002) obteve critério baixo de abstração pois possui uma quantidade baixa de detalhes. Comparada a abordagem de Zakaria *et al*. (2002) de nível médio, a abordagem de Stein *et al*. (2002) possui uma lista contendo algumas especificações do aspecto, tais como se ele é instanciado uma vez só para todo o ambiente global (PerJVM), se é um aspecto com prioridade(privileged=true/false) sobre outro, caso haja.

A abordagem de Zuzuki *et al*. (1999) é classificada como nível alto devido o fato de abstrair as funcionalidades dos aspectos.

A Tabela 3 pode ser utilizada para o programador a escolher uma abordagem por níveis de abstração de dados, ou seja o objetivo dessa tabela é detalhar as características

especificas de uma linguagem de programação, pode-se utilizar uma abordagem de baixo nível como a de Stein *et al*.(2002) para ter mais detalhes, entretanto, quando o objetivo for abstrair os dados da linguagem e optar mais para a modelagem menos detalhada, pode-se utilizar a abordagem descrita pode Suzuki (1999) que possui menos detalhes dentre as abordagens propostas.

Na Tabela 4 é mostrada uma comparação entre as abordagens utilizando o critério "nível de separação de interesses".

Tabela 4 – Comparação pelo critério de separação de interesses.

Abordagem	Separação de Interesses
Stein *et al*. (2002)	Assimétrica
Zakaria *et al* (2002)	Assimétrica
Suzuki *et al*. (1999)	Assimétrica

As modelagens são consideradas assimétricas por separarem o que é uma classe de aspecto das classes normais utilizando o estereótipo `<<aspect>>`.

Esse critério também pode ser utilizado para guiar a escolha de uma determinada abordagem. Abordagens simétricas devem ser escolhidas quando as linguagens de programação utilizadas são simétricas, como, por exemplo, HyperJ. Já quando linguagens assimétricas serão empregadas na implementação, as abordagens assimétricas é que devem ser escolhidas. Um exemplo de um linguagem assimétrica é a AspectJ.

.

Na Tabela 5 é mostrada a comparação entre as abordagens utilizando o critério "Rastreabilidade".

Tabela 5 – Comparação pelo critério de rastreabilidade.

Abordagem	Nível de Rastreabilidade
Stein *et al*. (2002)	Alto
Zakaria *et al* (2002)	Médio
Suzuki *et al*. (1999)	Baixo

A modelagem de Stein *et al*. (2002) é considerada de nível alta quando comparada com as demais pelo fato de que a rastreabilidade fica mais visível em sua modelagem, uma vez contendo bastante informações tanto para a implementação do sistema como para o projeto apenas.

Zakaria *et al*.(2002) por sua vez obteve uma modelagem de nível médio, visto que sua modelagem esta focada entre a implementação e o projeto. As informações para a implementação podem ser confusas no rodapé da modelagem, fazendo com que o engenheiro de software se perca na hora de implementar.

Ao contrario das demais abordagens, Suzuki *et al*. (1999) utiliza uma modelagem praticamente para projeto e não pra implementação do sistema, visto que oculta muitas informações importantes do sistema.

Na Tabela 6 é mostrada a comparação entre as abordagens utilizando como critério a Evolução.

Tabela 6 - Comparação pelo critério de evolução.

Abordagem	Evolução	
	Mudança	Adição - Remoção
Stein *et al*. (2002)	Uma mudança no núcleo	Temas podem ser

	pode exigir mudanças nas classes dependentes.	adicionados ou removidos.
Zakaria *et al*. (2002)	Uma mudança no núcleo pode exigir mudanças nas classes dependentes.	Aspectos podem ser adicionados e removidos sem necessidade de alteração no núcleo.
Suzuki *et al*. (2002)	Uma mudança no núcleo pode exigir mudanças nas classes dependentes.	Aspectos podem ser adicionados e removidos sem necessidade de alteração no núcleo.

Na Tabela 7 é mostrada a comparação entre as abordagens utilizando critério de representatividade do aspecto.

Tabela 7 – Comparação pelo critério de representatividade do aspecto.

Abordagem	Representatividade do Aspecto
Stein *et al*. (2002)	Alto
Zakaria *et al*. (2002)	Médio
Suzuki *et al*. (1999)	Baixo

A modelagem de Stein *et al*. (2002) é considerada de nível alta quando comparada com as demais pelo fato de conter características que a torna mais fácil quanto aos três requisitos proposto pela comparação. Stein *et al* (2002) propõem uma abordagem limpa e clara comparada as outras abordagens.

Zakaria *et al.* (2002) obteve uma média em relação as três abordagens, é fácil de identificar os aspectos, fácil de identificar os adendos, porem os pontos de junção ficam complexos de ser modelados, pois necessitam de uma interface para cada um, fazendo com que atrapalhe na hora da leitura da modelagem.

Suzuki *et al* (1999) ficou em ultimo colocado, foi proposto uma classe que facilida a vizualização do aspecto, não descreve os pontos de junção,nem os adendos, fazendo com que caia seu nível de representatividade do aspecto.

Na Tabela 8 é mostrada a comparação entre as abordagens utilizando critério de representatividade do entrecorte

Tabela 8 – Comparação pelo critério de representatividade do entrecorte .

Abordagem	Representatividade do entrecorte
Stein *et al*. (2002)	Médio
Zakaria *et al*. (2002)	Baixo
Suzuki *et al*. (1999)	Baixo

A abordagem de Stein *et al* (2002) obteve o nível Médio por possibilitar a visibilidade que um aspecto afeta uma classe, é possível identificar quais métodos são afetados nessa classe e por ser possível identificar se o método é afetado antes ou depois da chamada ou execução

Na abordagem de Zakaria *et al*. (2002) o nível de representatividade de entrecorte é classificado como Baixo por só conter a possibilidade de visualizar onde um aspecto afeta uma ou mais classes.

Suzkuki *et al*. (1999) assim como Zakaria at al (2002) obteve a classificação como baixa, pelo fato de não cumprir com os 4 fatores proposto pelo critério, Suzuki *et al* (2002) propõe uma modelagem com apenas visualização das classes afetadas pelo aspecto, mas não é possível visualizar quais são exatamente os métodos afetados, se é possível visualizar se o método afetado pelo aspecto antes ou depois de sua execução/chamada e nem se é possível visualizar exatamente os métodos afetados por um aspecto.

Na Tabela 9 são mostradas todas as abordagens com todos os critérios de comparação que foram utilizados neste trabalho. Para diminuir o tamanho da figura, para cada critério foi atribuído uma sigla, como pode ser observado na legenda posicionada na parte inferior da tabela.

Abordagem	Abst.	Sep Int.	Rast.	Evolução		Rep. Do Aspecto	Rep. Do Entrecorte
				Mudança	Adição - Remoção		
Stein *et al.*	Baixo	Assimétrica	Alto	Uma mudança no núcleo pode exigir mudanças nas classes dependentes	Aspectos podem ser adicionados ou removidos	Alto	Médio
Zakaria *et al.*	Médio	Assimétrica	Médio	Uma mudança no núcleo pode exigir mudanças nas classes dependentes	Aspectos podem ser adicionados e removidos sem necessidade de alteração no núcleo	Médio	Baixo
Suzuki *et al.*	Alto	Assimétrica	Baixo	Uma mudança no núcleo pode exigir mudanças nas classes dependentes	Aspectos podem ser adicionados e removidos sem necessidade de alteração no núcleo	Baixo	Baixo

Legenda: Abst.: Nivél de Abstração; Sep. Int.: Separação de Interesses; Rast.: Rastreabilidade; Rep. Do Aspecto: Representatividade do Aspecto; Rep. Do Entrecorte: Representatividade do Entrecorte

4.2 Considerações Finais

Foi mostrada uma comparação entre cada abordagem utilizando os critérios de comparação previamente definidos no capítulo 3 e feita uma breve conclusão em cada um dos critérios de comparação sobre as abordagens estudadas. Foi mostrada também uma tabela completa contendo todas as abordagens com todos os critérios analisados que representa o produto deste trabalho.

CONCLUSÃO

Este trabalho foi elaborado com o intuito de ajudar o projetista de software a escolher uma determinada abordagem de modelagem para um sistema que utiliza Aspectos. Para isto, foram apresentados alguns cenários utilizando a abordagem de modelagem de cada autor para que assim fosse possível obter um breve entendimento sobre o funcionamento da mesma.

Uma das limitações deste trabalho é a quantidade de abordagens analisadas. Existem muitas outras abordagens de modelagem para POA, entretanto, apenas três foram apresentadas pelo fato de que algumas delas são de difícil entendimento e o tempo também é escasso. Sendo assim, seria necessário um maior estudo das mesmas para que fosse possível apresentá-las e compará-las neste trabalho. A carência de cenários mais elaborados também é uma limitação deste trabalho visto que os cenários apresentados para o estudo são triviais.

Outra deficiência deste trabalho está no fato das abordagens terem sido apenas estudadas e comparadas na fase de projeto. Um maior estudo e comparação entre as abordagens durante as três fases de desenvolvimento (análise, projeto e implementação) com cenários mais elaborados deverá ser feito em trabalhos futuros.

A contribuição deste trabalho está no referencial deixado, que pode auxiliar os desenvolvedores na escolha por uma determinada abordagem de modelagem. Outra contribuição é o levantamento bibliográfico que pode auxiliar o desenvolvimento de outros projetos semelhantes.

REFERÊNCIAS BIBLIOGRÁFICAS

UETANABARA, J., Camargo, V.V. **Uma Comparação entre Abordagens de Modelagem Orientadas a Aspectos.** In: 1o. LA. WASP (*Workshop Latino-Americano de Desenvolvimento de Software Orientado a Aspectos*),2007, João Pessoa, Brasil.

GUEDES , T.A. Gilleanes **UML Guia de Consulta Rápida**. São Paulo. Editora Novatec 2005.

STEIN, Dominik, HANEMBERG, Stefan, UNLAND, Rainer **An UML-based Aspect-Oriented Design Notation for Aspectj.** Institute for Computer Science, University of Essen, Germany, 2002 .

ZAKARIA, A. Aida, HOSNY, Hoda, ZEID, Amir **A UML Extension for Modeling Aspect-Oriented Systems.** The American University in Cairo, 2002.

SUZUKI, Junichi, YAMAMOTO, Yoshikazu **Extending UML with Aspects: Aspect Support in the Design Phase.** Department of Computer Science, Yokohama City – Japan, 2002.

KICZALES, Gregor et al. **Aspect–oriented programming with AspectJ**. In OOPSLA Object-Oriented Programming, Systems, Languages and Applications)'01, Tutorial, Tampa FL.

MONTEIRO, Elaine. Monografia: **Um Estudo Sobre Modelagem Orientada a Aspectos Baseada em AspectJ e UML.** Centro Universitário Luterano de Palmas, 2004.

RAMOS, A. Ricardo, PENTEADO, Rosângela, MASIERO, C. Paulo **Um Processo de Reestruturação de Código Baseado em Aspectos.** Universidade de Federal de São Carlos - Departamento de Computação –DC, São Carlos-SP

BRITO Isabel, MOREIRA Ana, **Workshop on Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design, In 3rd International Conference on Aspect-Oriented Software Development (AOSD 2004),** Inglaterra, 2004.

RUMBAUGH, J., JACOBSON, I., BOOCH, G; **The Unified Modeling Language Reference Manual**, Addison- Wesley, 1999