3
Desenvolvimento de Software Orientado a Aspectos

O desenvolvimento de sistemas orientado a aspectos (DSOA) caracteriza uma nova sub-área da engenharia de software preocupada em desenvolver métodos, técnicas e ferramentas que dêem apoio a todas as fases do desenvolvimento. Ela leva em consideração a nova geração de tecnologias usadas para promover melhor separação de características, tais como programação orientada a aspectos e programação orientada a assuntos (*subjects*). Assim como quando surgiu o desenvolvimento orientado a objetos, DSOA surgiu depois que linguagens de programação atingiram um nível de maturidade suficiente para que haja a preocupação em propagar os novos conceitos e fundamentos para as etapas inicias do desenvolvimento.

PUC-Rio - Certificação Digital Nº 0210666/CA

Na Figura 14, resumimos como os trabalhos em DSOA (utilizados nesta tese) estão relacionados entre si: o DSOA envolve trabalhos em *Early-aspects*, programação orientada a aspectos e programação orientada a assuntos; trabalhos em *Early-aspects* consideram trabalhos em engenharia de requisitos orientada a aspectos e trabalhos em desenho orientado a aspectos; AspectJ e Hyper/J são exemplos de linguagens/ferramentas que possibilitam a programação orientada a aspectos (POA). Em (Brichau, 2005), encontramos um extenso relatório sobre linguagens de POA existentes.

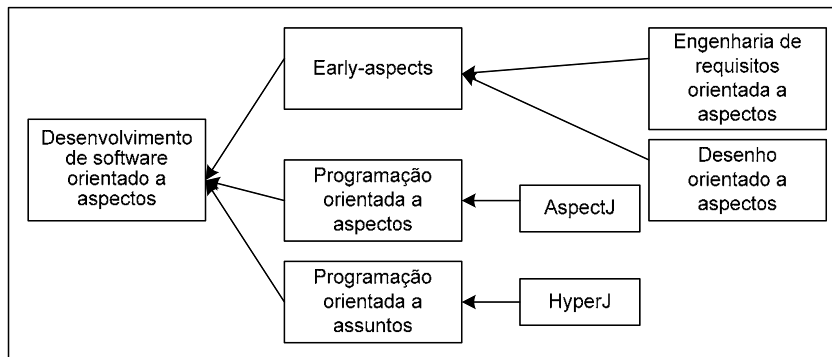

Figura 14. Visão geral dos trabalhos sobre desenvolvimento orientado a aspectos

Neste Capítulo apresentamos os principais conceitos sobre desenvolvimento orientado a aspectos que embasam esta tese. Na Seção 3.1, descrevemos as

propriedades da separação avançada de características. Nas Seções 3.2 e 3.3, apresentamos as linguagens AspectJ e Hyper/J. Na Seção 3.4, apresentamos algumas abordagens em engenharia de requisitos influenciadas pelo crescente interesse neste novo paradigma. Na Seção 3.5, resumimos como estas abordagens nos influenciaram.

3.1.
Propriedades da Separação Avançada de Características

Abordagens para separação avançada de características, tais como, programação adaptativa (Lieberherr, 1994; Lieberherr, 1996), filtros de composição (Aksit, 1992) e programação orientada a assuntos (Harrison, 1993) foram responsáveis pela consolidação da programação orientada a aspectos, num sentido mais amplo. Em (Chavez, 2003, 2004), um modelo de aspectos é definido para esclarecer e unificar os conceitos que estas abordagens utilizam. Neste modelo de aspectos, Chavez (2003, 2004) define os seguintes elementos como necessários para que uma linguagem seja considerada orientada a aspectos, veja Figura 15[7].

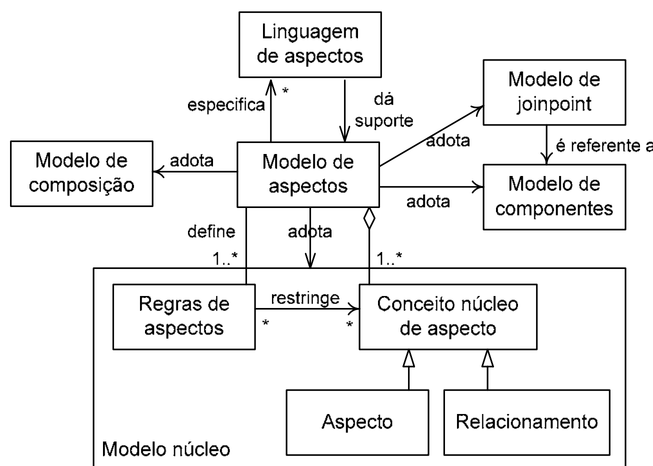

Figura 15. Modelo de aspectos

- Modelo de componentes – representa o modelo conceitual que define os tipos de componentes, regras e princípios gerais que ajudam a pensar sobre o problema e decompô-lo. Por exemplo, o modelo de objetos;
- Modelo de *joinpoint* – representa o modelo conceitual que define os elementos do modelo de componentes que podem ser afetados por um

[7] Esta figura é uma tradução da figura original definida em (Chavez, 2003, 2004).

PUC-Rio - Certificação Digital Nº 0210666/CA

"aspecto". O modelo de *joinpoint* é, assim, dependente do modelo de componentes adotado. Ele pode denotar um elemento relacionado à estrutura estática (contexto léxico) ou dinâmica (contexto dinâmico), por exemplo, métodos e atributos do modelo de objetos;

- Modelo núcleo (core) – representa o modelo conceitual que define: "aspectos" como uma abstração para modularizar características transversais e "relacionamento" (crosscutting) como um mecanismo de composição de aspectos e componentes. "Aspectos" especificam um conjunto de *joinpoints* (interfaces transversais – onde corta) e modificações (*features* transversais – o que é transversal) para serem aplicadas neles. "Relacionamento" é o relacionamento que liga um aspecto a um ou mais componentes, afetando sua estrutura e comportamento.
- Modelo de composição (weaving) – representa o modelo conceitual que define os tipos de mecanismos de composição. A composição é um processo que relaciona "aspectos" e componentes especificados nos *joinpoints*;
- Suporte à dicotomia – dicotomia é a clara distinção e separação do que são componentes e aspectos;
- Suporte a algum tipo de quantificação sobre a estrutura ou comportamento dos componentes (quantification) – definida como a habilidade de escrever declarações unitárias e separadas que atingem muitos lugares não-locais em um código;
- Suporte à transparência (obliviousness) – assume-se que os aspectos são invisíveis para os componentes, i.e., é a idéia de que os componentes não precisam conhecer nem ser preparados para receber as mudanças providas pelos aspectos.

Em (Chavez, 2003, 2004) é apresentada uma avaliação de algumas linguagens em relação a este modelo de aspectos. Na Tabela 1, ilustramos um resumo desta avaliação para as linguagens AspectJ e Hyper/J, e nas seções seguintes descrevemos estas duas linguagens.

PUC-Rio - Certificação Digital Nº 0210666/CA

Tabela 1. Avaliação de AspectJ e Hyper/J utilizando o modelo de aspecto

	AspectJ	**Hyper/J**
Modelo de componentes	modelo de objetos	modelo de objetos
Componente	Objeto	objeto
Linguagem de componentes	Java	Java
Modelo de ***joinpoint***		
Joinponits dinâmicos	pontos em execução	--
Joinponits estáticos	classes	classes, campos, métodos, intefaces
Modelo núcleo		
Aspecto	aspect	*hyperslice*?
Interface transversal	*Pointcuts*	Hypermodule
Feature **transversal**	introduction, *advice*	campos, métodos
Modelo de composição	estático e dinâmico	estático
Dicotomia	sim	não
Quantificação	sim	sim
Transparência	sim	sim

3.2. AspectJ

O termo programação orientada a aspectos foi inicialmente utilizado entre 1995 e 1996 em um projeto da *Xerox Palo Alto Research Center* supervisionado por Gregor Kiczales (Lopes, 2004). Este grupo estava desenvolvendo algumas ferramentas de propósito específico (AML, RG, DJ) que influenciaram o surgimento da linguagem AspectJ como uma linguagem de **propósito geral** (Lopes, 2004).

AspectJ é uma extensão à linguagem Java (AspectJ, 2006). Nela, a separação é realizada através da inserção de uma nova abstração denominada "aspecto" à programação orientada a objetos. A composição é realizada através de um combinador denominado *weaver*, que é responsável por pré-processar o código orientado a objetos, inserindo ou modificando os objetos com o comportamento dos aspectos.

Além dos métodos e atributos, um aspecto é composto de *pointcuts*, *advice* e *intertype declarations*. Eles podem alterar a estrutura estática ou dinâmica de um programa. A estrutura estática é alterada adicionando, por meio das *intertype declarations*, membros (atributos, métodos ou construtores) a uma classe, modificando, assim, a hierarquia do sistema. A alteração na estrutura dinâmica de

PUC-Rio - Certificação Digital Nº 0210666/CA

um programa ocorre por meio dos *joinpoints*, que são selecionados por *pointcuts*, e pela adição de comportamentos (*advice*) antes ou depois dos *joinpoints*.

Um *pointcut* indica os pontos onde um determinado comportamento será inserido. Ele é descrito por um nome e um corpo. O corpo indica os *joinpoints* onde serão aplicados o *advice* associado, podendo haver vários *joinpoints* aninhados através dos operadores *or*, *and* e *not*. Cada *pointcut* está associado a um ou mais *advice*. *Advice* descreve o comportamento (trecho de código) a ser inserido nos *joinpoints*. Existem três formas principais de *advice*, são elas: *before*, *around* e *after*. Como seus nomes sugerem, *before* executa antes do *joinpoint*, *after* executa depois e *around* executa antes e depois, alternativamente, substituindo o comportamento do *joinpoint*.

Na Figura 16 (Brichau, 2005), exemplificamos os elementos de um aspecto na linguagem AspectJ. No aspecto abstrato FFDC são definidos três *pointcuts* (1) e um *advice* (2), que especifica a emissão de uma exceção (3) antes dos pontos afetados..

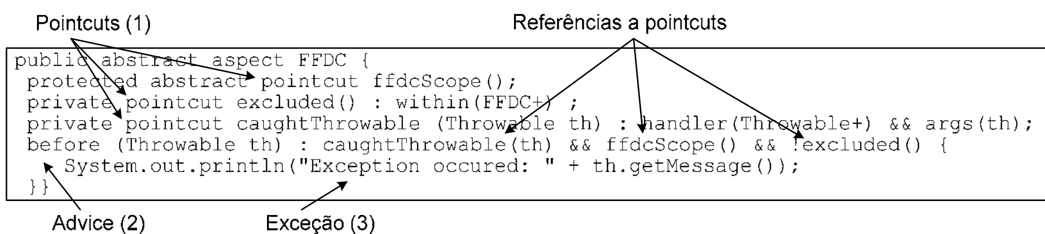

Figura 16. Exemplo de aspecto em AspectJ

Nossa abordagem utiliza os conceitos de *pointcut*, *advice* e *intertype declaration* para representar, em modelos de requisitos, como características transversais afetam umas as outras. No Capítulo 4, apresentamos nossa abordagem.

3.3. Hyper/J

Hyper/J originou-se da abordagem de separação multidimensional de características (Tarr, 1999), uma evolução da programação orientada a assuntos (Harrison, 1993). Com o objetivo de prover uma flexível e incremental separação, modularização e integração de artefatos de software baseados em diferentes características (Ossher, 2001; Hyper/J, 2006).

PUC-Rio - Certificação Digital Nº 0210666/CA

A programação orientada a assuntos foi proposta em 1993 como uma extensão do paradigma de orientação a objetos para abordar o problema de separar visões alternativas de uma classe ou um conjunto de classes. Harrison e Ossher (1993) definem a noção de especificar, separadamente, hierarquias diferentes de classes. Cada hierarquia implementaria uma característica do sistema e a composição delas seria realizada para construir o sistema, ou sistemas diferentes.

Assuntos são conjuntos de classes ou partes de classes que formam uma hierarquia para modelar uma característica. A combinação de hierarquias, escrita através de regras de composição, produz novos assuntos e é realizada por uma ferramenta de composição (Kaplan, 1996). Desta forma, o programador pode criar vários modelos, cada um encapsulando uma única característica, e assim, gerenciar os problemas de espalhamento e entrelaçamento de características.

Por volta de 1999, a programação orientada a assuntos evoluiu para separação multidimensional de características (SMC) (Ossher, 2001). O termo "multidimensional" significa "múltiplos critérios para decomposição". SMC provê um meio para que os desenvolvedores decomponham o sistema, utilizando diferentes critérios (dimensões) ao mesmo tempo e de maneira não-invasiva. Desta forma, SMC trata da decomposição dominante.

Hyper/J é uma ferramenta desenvolvida pela IBM T.J. Watson Research Center que provê SMC para a linguagem Java (HyperJ, 2006). Em oposição a AspectJ, Hyper/J não modifica ou estende a linguagem Java. Hyper/J provê uma liguagem paralela para especificação de *hyperslice*s, sendo usada juntamente ao código para gerar um novo conjunto de unidades integradas.

PUC-Rio - Certificação Digital Nº 0210666/CA

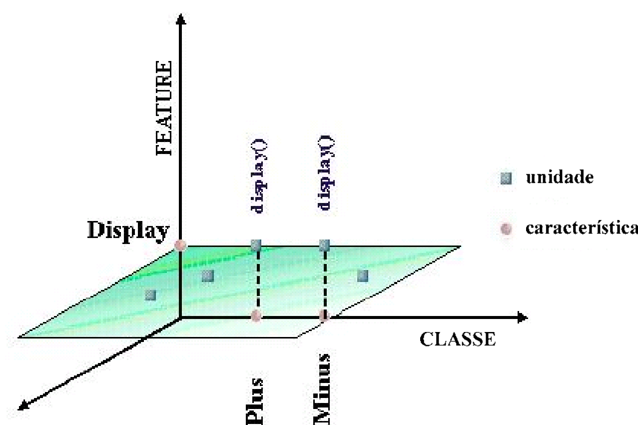

Figura 17. Exemplo de Hyperslice

Um *hyperslice* é um módulo que encapsula uma *feature*. Na Figura 17 (Chavez, 2004), o *hyperslice* denominado “display” é representado pelo plano que corta cinco unidades (representadas pelos quadrados). Destas cinco unidades, duas são os métodos “display” das classes “minus” e “plus”.

Para usar a ferramenta Hyper/J é necessário oferecer os seguintes arquivos (Chavez, 2004):

- Hiper-espaço – um arquivo que consiste de uma lista de classes Java com as quais o desenvolvedor está trabalhando e deseja aplicar Hyper/J.
- Mapeamento de características – um ou mais arquivos descrevendo quais unidades das classes Java (do hiper-espaço) endereçam cada *feature*.
- Hiper-módulo – um arquivo descrevendo quais *hyperslices* (*features*) devem ser integrados e como a integração deve ocorrer, incluindo a estratégia de composição e expressões de ordem. Um relacionamento de integração indica que unidades de correspondência nos diferentes *hyperslices* devem ser integradas umas com as outras.

Unidades de correspondência são unidades (interfaces, classes, métodos, atributos, dentre outros) relacionadas segundo algum critério e são compostas (misturadas ou sobre-escritas) dentro de uma nova unidade de software. As unidades de correspondência são identificadas utilizando uma das seguintes estratégias de composição: *mergeByName*, *nonCorrespondingMerge* e *overrideByName*. Pode-se ainda indicar a ordem em que a integração deve ocorrer através do relacionamento *Order*; a combinação de unidades, mesmo que seus nomes não sejam os mesmos, através do relacionamento *Equate*; e o relacionamento *Bracket* indica que um conjunto de métodos deve ser agrupado de maneira que quando invocado será precedido ou seguido por outros métodos.

Na Figura 18 (Brichau, 2005), exemplificamos a especificação de um hiper-módulo, que contém os *hyperslice*s: *kernel*, *check*, e *display*. Estes *hyperslice*s são integrados utilizando a estratégia *mergeByName*. O relacionamento utilizando *equate* indica uma correspondência entre as operações *process*, *check_process* e *display_process* das características *Kernel*, *Check* e *Display*, respectivamente. O resultado da composição são novos arquivos de classes, que podem ser utilizados na execução.

PUC-Rio - Certificação Digital Nº 0210666/CA

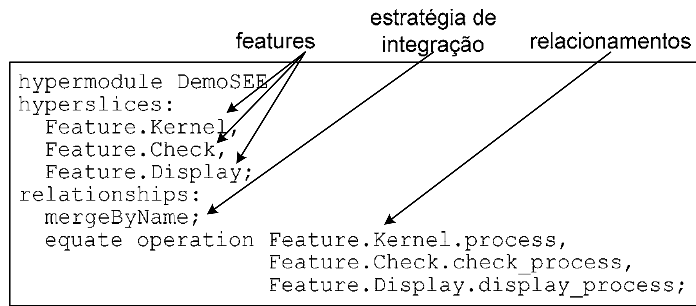

Figura 18. Exemplo de hiper-módulo em Hyper/J

Desta maneira, Hyper/J provê (HyperJ, 2006):

- Flexível separação e modularização de características – sistemas Java podem ser escritos com ou sem a separação de características multidimensional. Hyper/J possibilita a identificação, separação e manipulação de características, podendo ser utilizado apenas quando for necessário, durante a evolução.
- Composição – o mecanismo de composição sintetiza e integra tipos diferentes de características, e provê a habilidade de realizar *mix-and-match* e *plug-and-play* não invasivo, tratando alguns problemas de desenvolvimento, tais como: criação de extensões e configurações de software, sem modificar o código fonte original; customização e integração de sistemas e componentes reusáveis; facilitação do desenvolvimento em vários times, cada um trabalha em modelos independentes; manutenção e rastreabilidade entre todos os artefatos de software.
- Evolução e remodularização não-invasiva e sob demanda – por facilitar a separação utilizando diferentes critérios, utilizando uma linguagem em paralelo a Java, Hyper/J permite efetuar mudanças no código à medida que novas características e melhores formas de decompô-las surgem.

Nossa abordagem é inspirada em Hyper/J da seguinte maneira: provemos um mecanismo de separação menos intrusivo, podendo ser utilizado de acordo com a demanda; e extraímos visões segundo diferentes critérios a partir da especificação estática (descrição léxica), tal como a composição realizada em Hyper/J. No Capítulo 4, Seção 4.2.3 e Seção 4.3.3, respectivamente, apresentamos estas propriedades em nossa abordagem.

PUC-Rio - Certificação Digital Nº 0210666/CA

PUC-Rio - Certificação Digital Nº 0210666/CA

3.4. Engenharia de Requisitos Orientada a Aspectos

Early-aspects é uma nova sub-área da engenharia de software que está interessada em prover técnicas e métodos para sistematicamente identificar, separar, representar e compor características transversais nas fases de desenvolvimento anteriores à implementação (Rashid, 2002). Ainda não se tem uma clara definição para "aspectos" no processo de definição de requisitos (Bakker, 2005; Nuseibeh, 2004). A maioria das abordagens tem trabalhado com o conceito de "candidato a aspectos", denotando características transversais, e em muitas destas, elas são somente RNFs.

Em (Chitchyan, 2005a) é apresentado um apanhado das abordagens para engenharia de requisitos que se preocupam com a separação e composição de características transversais. Os autores definem alguns critérios para comparar estas abordagens e avaliar novos nichos de pesquisa na área. Os critérios são, suporte à: identificação, composição, análise de interações e tomada de decisões, rastreabilidade, mapeamento, evolução e escalabilidade. Em (Chitchyan, 2005a, 2005b), a comparação entre estas abordagens é apresentada e é relatado que a identificação de características tem sido o tópico mais pesquisado, enquanto os demais critérios não têm recebido a devida atenção.

Como os trabalhos neste tópico apresentam idéias ainda em evolução, nesta seção não apresentamos conceitos consolidados, mas sim, as principais abordagens na área. Tentamos organizá-las por autor ou grupo de pesquisa, seguindo a ordem cronológica de publicação. Estes trabalhos nos inspiraram e foram decisivos para definição de nossa abordagem.

3.4.1. Rashid, Moreira e Brito

Em (Rashid, 2002), é definido um modelo genérico para engenharia de requisitos orientada a aspectos (modelo EROA). Este modelo explicita a necessidade de tratar características transversais cedo, e primeiramente define que estas características são "candidatas a aspectos". Neste trabalho, apenas RNFs em um alto nível de abstração são identificados como candidatos a aspectos.

A identificação de candidatos a aspectos é realizada por identificar em uma matriz requisitos versus características (*concerns*) quais as características que cruzam mais de um requisito. Posteriormente, cada candidato a aspecto é analisado em mais detalhe para a identificação de conflitos e estabelecimento de prioridades. Por fim, deve-se especificar o impacto de aspectos candidatos em termos de duas dimensões: i) mapeamento – indica como ele é mapeado para fases posteriores do desenvolvimento (função, decisão arquitetural ou aspecto); e ii) influência – estabelece quais as atividades do ciclo de desenvolvimento são afetadas pelo candidato a aspectos (requisitos, arquitetura, projeto ou manutenção).

Em (Rashid, 2003), o modelo EROA é melhorado por incluir as atividades de composição e tratamento de conflitos, veja Figura 19. A composição determina como candidatos a aspectos influenciam ou restringem requisitos, através de regras de composição. O tratamento de conflitos é realizado através do preenchimento de uma matriz de contribuição entre aspectos candidatos, e atribuição de pesos quando um contribui negativamente para a realização do outro.

PUC-Rio - Certificação Digital Nº 0210666/CA

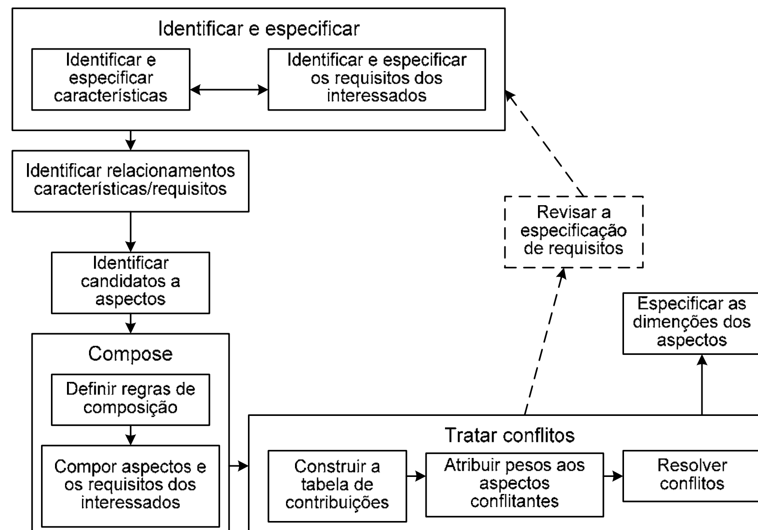

Figura 19. Modelo EROA

Neste trabalho, Rashid (2003) define uma instância do modelo EROA baseada em pontos de vista (Finkelstein, 1996). Nesta instância, a identificação de características transversais é realizada por meio de uma matriz que apresenta pontos de vista (visões como opiniões) versus requisitos. Se um requisito cruza vários pontos de vista então ele é um candidato a aspectos. Requisitos são escritos na forma de sentenças em linguagem natural não estruturada. XML é utilizado

para especificar tanto sentenças de requisitos, quanto candidatos a aspectos e regras de composição, veja um exemplo na Figura 20.

Os três formatos de descrição, ilustrados na Figura 20 representam uma maneira formal de descrever os requisitos e as composições. Entretanto:

- são utilizadas sentenças de requisitos não estruturadas, não permitindo a composição entre partes do requisito;
- duas representações diferentes são utilizadas para os requisitos de pontos de vista e requisitos que são candidatos a aspectos, dificultando o reuso;
- a composição não pode ser realizada entre candidatos a aspectos ou entre pontos de vista;
- o formato utilizado para especificar regras de composição, apesar de ser simples, torna a especificação mais extensa porque não provê uma maneira simplificada de escrever que uma característica transversal afeta vários pontos (propriedade de quantificação, definida na Seção 3.1), assim, para cada ponto tem que ser especificada uma regra.

PUC-Rio - Certificação Digital Nº 0210666/CA

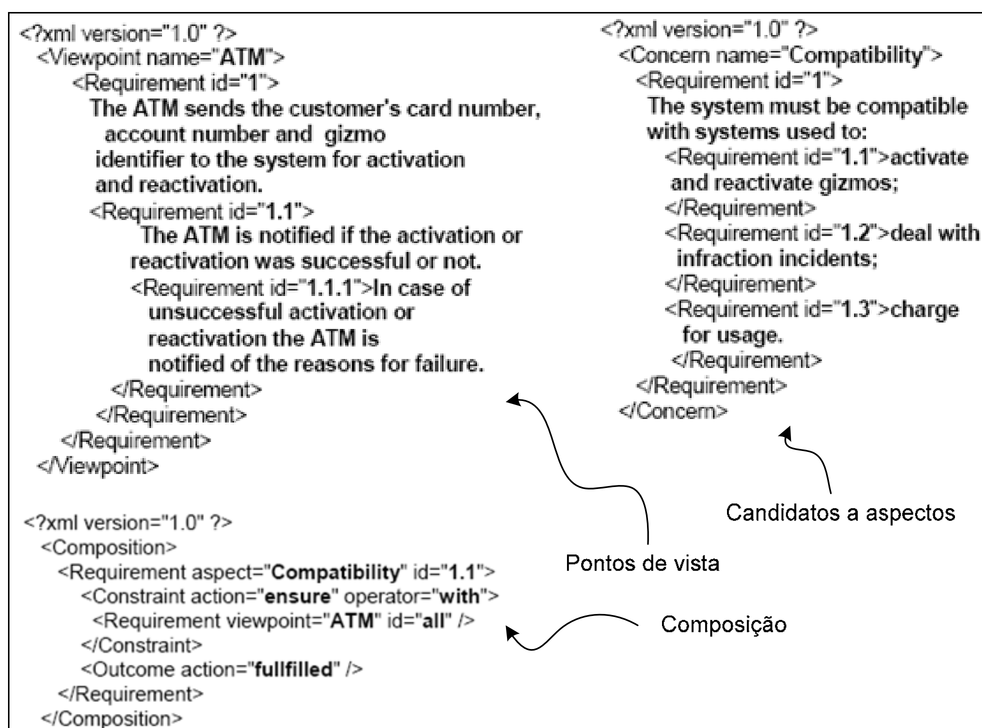

Figura 20. Exemplo de descrição de requisitos no modelo EROA

Em (Moreira, 2002) é definido um processo para identificação e especificação de RNFs que cortam RFs, bem como a integração entre eles representada por modelos de casos de uso e diagramas de sequência. Na Figura 21a, ilustramos o *template* utilizado para especificar RNFs e na Figura 21b há um

exemplo ilustrativo. As informações contidas nas linhas *Where* e *Requirements* do *template* (Figura 21) são utilizadas para identificar se o RNF corta vários casos de uso. Se sim ele é considerado um RNF transversal.

Os conceitos (operadores) *overlapping*, *overriding* e *wrapping* são utilizados para representar, respectivamente, mudança, sobreposição ou encapsulamento do requisito afetado. No diagrama de casos de uso os RNFs transversais são representados por casos de uso esteriotipados, os detalhes da composição são mostrados em diagramas de sequência.

Em (Brito, 2004), Brito amplia a abordagem definida em (Moreira, 2002), utilizando o framework NFR para identificar e especificar RNFs. Além disto, são definidos os operadores *enabling*, *disabling*, *pure interleaving* e *full synchronization*, inspirados nos operadores LOTOS, para registrar a ordem em que os RNFs transversais devem ser aplicados.

(a)
Name The name of the quality attribute
Description Executive description
Focus A quality attribute can affect the system (i.e. the end product) or the development process
Source Source of information (e.g. stakeholders, documents)
Decomposition Quality attributes can be decomposed into simpler ones. When all (sub) quality attributes are needed to achieve the quality attribute, we have an AND relationship. If not all the sub quality attributes are necessary to achieve the quality attribute, we have an OR relationship
Priority Expresses the importance of the quality attribute for the stakeholders. A priority can be MAX, HIGH, LOW and MIN
Obligation Can be optional or mandatory
Influence Activities of the software process affected by the quality attribute
Where List of the actors influenced by the quality attribute and also a list of models (e.g. use cases and sequence diagrams) requiring the quality attribute
Requirements Requirements describing the quality attribute
Contribution Represents how the quality attribute affects other quality attributes. This contribution can be positive (+) or negative (-)

(b)
Name Security
Description Restricts the access to the system and to the data within the system
Focus System
Source Stakeholders
Decomposition Integrity and Confidentiality. Both have an AND relationship with
Security
Priority MAX
Obligation Mandatory
Influence Design, system architecture and implementation
Where Actors: VehicleOwner, Bank Use cases: PayBill, RegisterVehicle
Requirements The system must: 1. protect the vehicle's owner registration data 2. guarantee integrity in the data transmitted to the bank 3. guarantee integrity on data changed/queried by the operator 4.…
Contribution (-) to response time and (+) to multiuser and compatibility.

Figura 21. a) Template para RNFs, b) exemplo utilizando o template

Assim, as abordagens em (Brito, 2004; Moreira, 2002) consideram apenas RNFs como características transversais e as informações informais contidas no *template* são utilizadas apenas como guia para o engenheiro construir manualmente os diagramas; nenhum suporte ferramental é proposto.

PUC-Rio - Certificação Digital Nº 0210666/CA

Em (Moreira, 2005a, 2005b), os autores apresentam uma abordagem denominada multi-dimensional, para explicitar a necessidade de considerar RFs tanto quanto RNFs como candidatos a aspectos. Nesta abordagem define-se:

- um espaço de meta-características (*meta-concerns*) onde características concretas, específicas de uma aplicação, podem ser derivadas baseando-se nas *features* específicas de um domínio de problema;
- a noção de interseção composicional, possibilitando a identificação de conjuntos de características como base (foco), de maneira a observar as interações (*trade-offs*) entre as outras características; e
- a utilização da análise destas interações para prover um guia de mapeamento destas estruturas para a arquitetura.

Esta abordagem utiliza os formatos de descrição para regras de composição e para características definidos em (Rashid, 2003), mas substitui a descrição de pontos de vista, por uma descrição de meta-características, veja um exemplo na Figura 22. As regras de composição definidas em (Rashid, 2003) são utilizadas. A diferença é que em (Moreira, 2005a, 2005b) uma característica pode restringir qualquer outra característica e não apenas pontos de vista como em (Rashid, 2003).

```
<?xml version="1.0" ?>
<MetaConcern name="InformationRetrieval">
  <Description>The operation of accessing information from a computer system</Description>
  <Examples>Database retrieval, Multimedia retrieval</Examples>
  <Relationships>Availability, Mobility, InformationUpdate</Relationships>      ← Meta-concern
</MetaConcern>

<?xml version="1.0" ?>
<Concern name="InformationRetrieval">
  <Requirement id="1">It should be possible to retrieve information from the system.
    <Requirement id="1.1">It should be possible to access information about the attractions</Requirement>
    <Requirement id="1.2">It should be possible to access information about the current location</Requirement>
  </Requirement>
  <Requirement id="1.3">It should be possible to obtain a list of available preset tours</Requirement>
</Concern>                                                                      ← Concern

<?xml version="1.0" ?>
<Composition>
 <Requirement concern="InformationRetrieval" id="all" />
  <Constraint action="provide" operator="during">                              ← Composição
    <Requirement concern="Customisability" id="all"/>
    <Requirement concern="Navigation" id="1"/>
    <Requirement concern="Mobility" id="1" children="include"/>
    <Requirement concern="InformationUpdate" id="all"/>
  </Constraint>
  <Outcome action="fulfilled"/>
 </Requirement>
</Composition>
```

Figura 22. Exemplo de descrição de requisitos na abordagem multidimensional

PUC-Rio - Certificação Digital Nº 0210666/CA

3.4.2. Baniassad

Em (Baniassad, 2004) é definida uma abordagem para análise (Theme/Doc) e desenho (Theme/UML) baseados no conceito de “temas” (*themes*). Um tema é um conjunto de estruturas e comportamentos que representam uma *feature*. O modelo de temas define dois tipos de temas, base e transversais. Os temas transversais são aqueles que compartilham estruturas e/ou comportamentos com outros temas, e são, assim, considerados “aspectos”.

Theme/Doc provê visões e suporte ferramental para identificação e representação de características transversais. As visões de ações (*action view*), de ações agrupadas (*clipped action view*) e a visão de temas (*theme view*) são geradas semi-automaticamente através da análise léxica de um documento de requisitos. Na Figura 23 e Figura 24, os losangos representam os verbos das sentenças de requisitos e as elipses representam as sentenças onde eles aparecem. Por meio das interações entre as ações visualizadas na Figura 23, e análise das sentenças de requisitos, o engenheiro percebe que a ação “logged” corta as ações “register”, “unregister” e “give”, retratando este relacionamento transversal com um elo cinza na Figura 24.

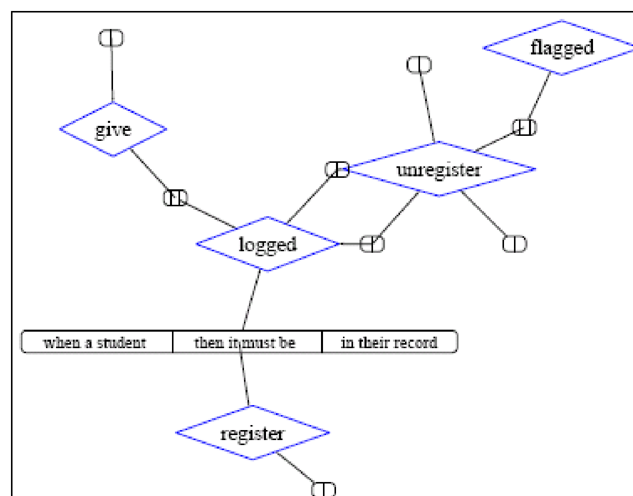

Figura 23. Exemplo da visão de ações de Theme/Doc

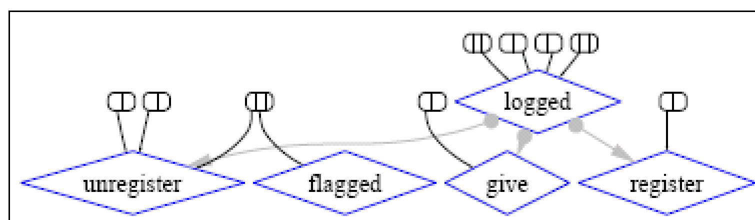

Figura 24. Exemplo da visão de ações agrupadas de Theme/Doc

PUC-Rio - Certificação Digital Nº 0210666/CA

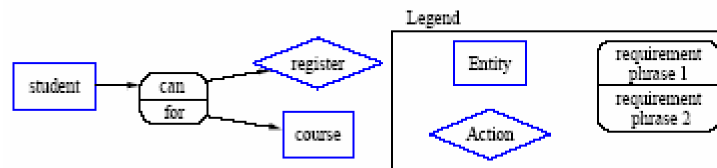

Figura 25. Exemplo da visão de temas de Theme/Doc

Estas visões ajudam o engenheiro a analisar as interações entre requisitos, e assim, separá-los em dois grupos. Um grupo onde os requisitos são auto-contidos, i.e, nenhum requisito de um conjunto interage com requisitos de outro conjunto, e o segundo grupo onde os conjuntos possuem requisitos que interagem com os de outros conjuntos, i.e, são transversais. A visão de temas (Figura 25) é utilizada para guiar o engenheiro na modelagem da arquitetura usando Theme/UML, que consiste de algumas extensões para os diagramas UML.

A geração destas visões é semi-automática e o engenheiro precisa fornecer outras informações além das sentenças de requisitos, desta forma mudanças em requisitos podem demandar algum re-trabalho ao engenheiro. Além disto, nenhum mecanismo para composição de características transversais é proposto.

3.4.3. Sousa

Em (Sousa, 2003), propõe-se uma adaptação do framework NFR (Mylopoulos, 1992). Esta adaptação modifica algumas atividades do framework, para incluir a composição e o mapeamento de RNFs transversais, veja a Figura 26. Os operadores de composição definidos em (Moreira, 2002), representando os conceitos *overlapping*, *overriding* e *wrapping*, são utilizados para adicionar a semântica dos candidatos a aspectos aos relacionamentos dos grafo de RNFs no nível de operacionalização, e não no nível abstrato como em (Moreira, 2002). Veja o exemplo ilustrado na Figura 27 (Sousa, 2003).

PUC-Rio - Certificação Digital Nº 0210666/CA

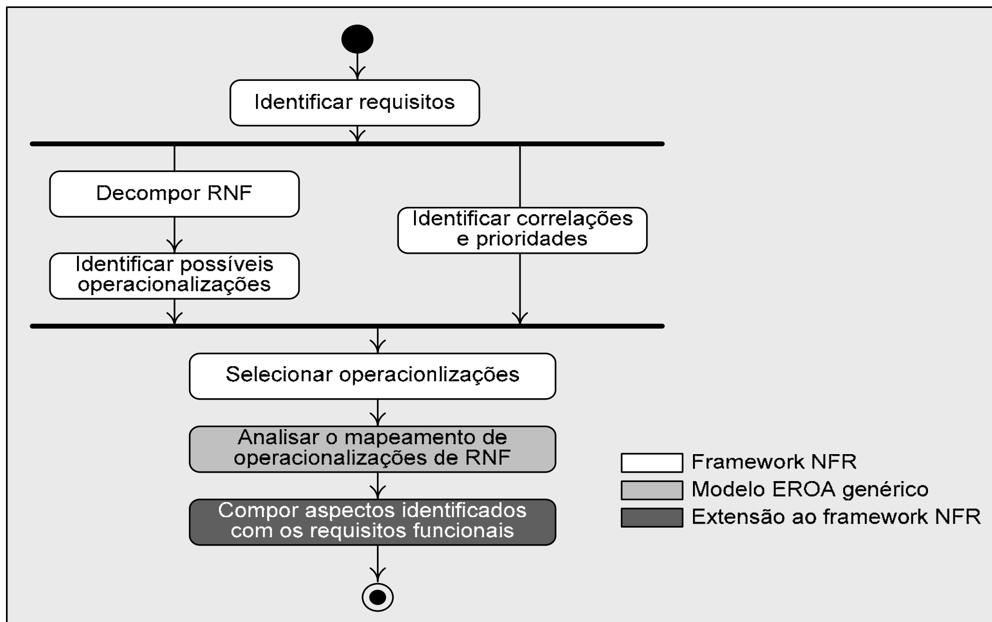

Figura 26. Adaptação do framework NFR

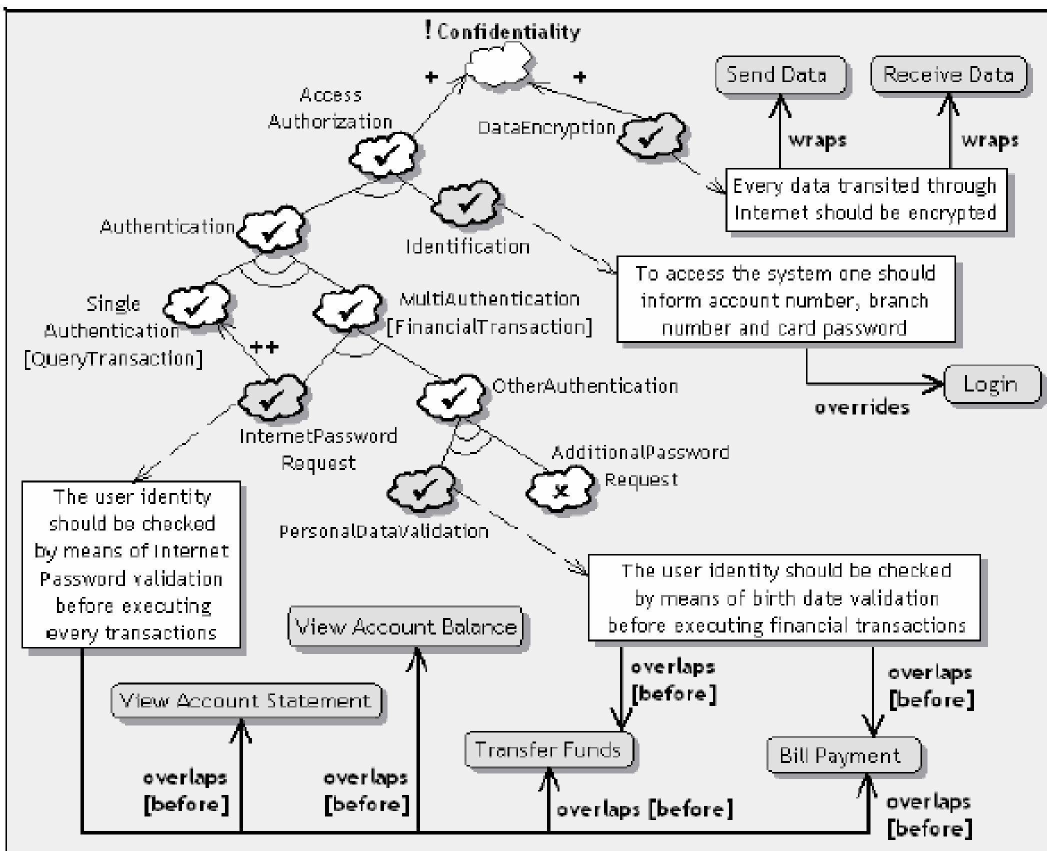

Figura 27. Exemplo da modelagem de candidato a aspectos no grafo de RNFs

Em (Sousa, 2004a, 2004b) é proposta a integração da abordagem de desenvolvimento dirigida por caso de usos (Jacobson, 1999) e o framework NFR (Chung, 2000), para dar suporte à separação de características transversais. O processo criado consiste em modelar RNFs e casos de uso, separadamente, e

PUC-Rio - Certificação Digital Nº 0210666/CA

então, especificar em tabelas como os RNFs afetam os casos de uso. Com base nesta especificação, o engenheiro pode incluir as operacionalizações de RNFs nos diagramas de casos de uso, relacionando-os com "*extend*". Além disto, os autores propõem: mudar o rótulo "*extend*" dos casos de uso para "*crosscut*" quando ele relaciona uma característica transversal; e usar um *template*, similar ao *template* dos casos de uso, para descrever as operacionalizações dos RNFs. Veja o exemplo ilustrado na Figura 29.

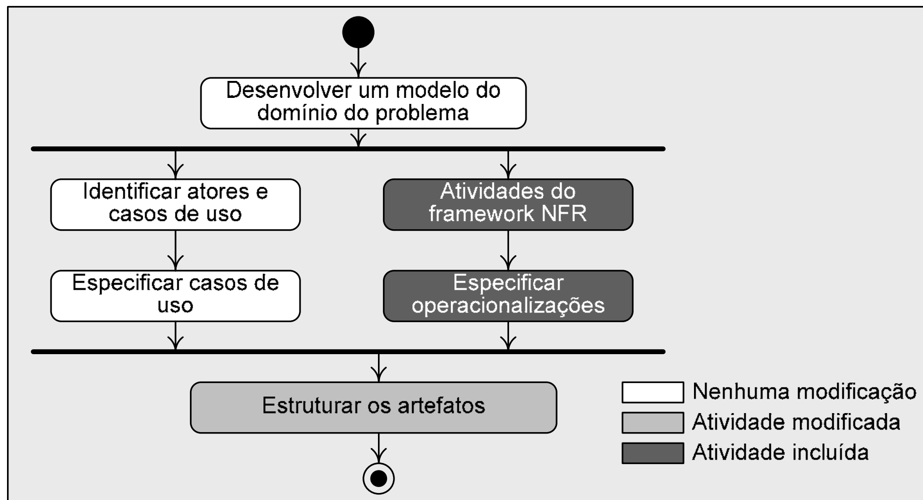

Figura 28. Adaptação do processo dirigido por caso de uso

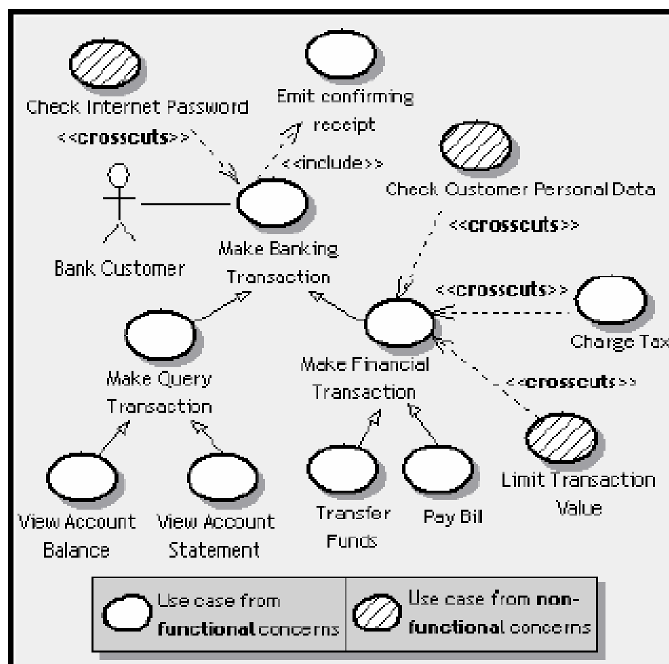

Figura 29. Exemplo de modelagem de candidato a aspectos em casos de uso

As abordagens definidas em (Sousa, 2003, 2004a, 2004b) trazem a preocupação em considerar as operacionalizações de RNFs, ao invés de RNFs abstratos. São definidas algumas diretrizes para identificar, integrar e mapear

PUC-Rio - Certificação Digital Nº 0210666/CA

características transversais para fases posteriores, mas a linguagem para retratar características transversais é apenas diagramática, criada manualmente com apoio das descrições informais de composição. Não há facilidades de ferramentas para modelagem dos artefatos propostos (diagrama de casos de uso, *template* de casos de uso, tabela de composição, diagrama de colaboração), sendo difícil manter o rastro entre eles.

3.5.
Resumo

Neste Capítulo descrevemos os conceitos principais que influenciaram o desenvolvimento de nossa abordagem para integração de características transversais durante a engenharia de requisitos. Na Seção 3.1, descrevemos as principais propriedades da separação avançada de características e o meta-modelo, definido em (Chavez, 2003, 2004), para abordagens orientadas a aspectos.

Na Seção 3.2 e 3.3, apresentamos as linguagens de programação orientada a aspectos, AspectJ e Hyper/J. AspectJ nos influenciou em definir um relacionamento transversal, como elemento da linguagem de modelagem, para registrar como algumas características se espalham e se entrelaçam a outras. Ao mesmo tempo, Hyper/J nos influenciou com a idéia de ser menos intrusivo e registrar grupos de elementos de acordo com diferentes critérios, possibilitando que a decomposição dominante da linguagem de modelagem seja utilizada, mas que outras maneiras de separação possam coexistir em um outro espaço.

Hyper/J e AspectJ são consideradas, respectivamente, abordagens simétricas e assimétricas (Harrison, 2002). Não realizamos uma avaliação aprofundada sobre a simetria de nossa abordagem, mas consideramos que nossa abordagem não pode ser classificada integralmente como simétrica nem como assimétrica. Ela apresenta simetria nos elementos porque não há distinção entre os elementos que estão transversais e aqueles que não estão, mas define um novo tipo de relacionamento que deve ser utilizado para explicitamente relacionar elementos que estão transversais. Por outro lado, nossa abordagem apresenta assimetria no relacionamento porque explicitamente define os pontos que afetam e são afetados pelas características transversais.

Na Seção 3.4, resumimos as principais abordagens em engenharia de requisitos orientada a aspectos. Todos estes trabalhos inspiraram a idéia para

PUC-Rio - Certificação Digital Nº 0210666/CA

nossa proposta. Entretanto, eles estão focados em definir métodos e técnicas que ajudam o desenvolvimento de software orientado a aspectos, tentando promover o paradigma de aspectos em todas as fases de desenvolvimento, enquanto nós estamos primeiramente interessados em ajudar a engenharia de requisitos por utilizar técnicas e ferramentas inspiradas naquelas de programação, e, só então, ajudar à engenharia de requisitos orientada a aspectos. Nosso foco é a modelagem e composição de características transversais, com o propósito de fornecer diferentes visões do modelo integrado, facilitando a inclusão e exclusão de diferentes características da modelagem de requisitos. Uma comparação detalhada entre estes trabalhos e o nosso é apresentada no Capítulo 6.

PUC-Rio - Certificação Digital Nº 0210666/CA