



Universidade Federal de
Pernambuco

Centro de Informática

PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO



"UMA ABORDAGEM DIRECIONADA A CASOS DE USO PARA O DESENVOLVIMENTO DE SOFTWARE ORIENTADO A ASPECTOS"

Por

Geórgia Maria Carvalho de Sousa

Dissertação de Mestrado



Universidade Federal de Pernambuco

posgraduacao@cin.ufpe.br

www.cin.ufpe.br/~posgraduacao

Recife, maio de 2004

Geórgia Maria Carvalho de Sousa

**“UMA ABORDAGEM DIRECIONADA A CASOS DE USO PARA O
DESENVOLVIMENTO DE SOFTWARE ORIENTADO A ASPECTOS”**

DISSERTAÇÃO APRESENTADA AO PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO DO CENTRO DE INFORMÁTICA DA UNIVERSIDADE FEDERAL DE PERNAMBUCO, COMO REQUISITO PARCIAL PARA OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIA DA COMPUTAÇÃO.

ORIENTADOR: JELSON BRELAZ DE CASTRO

Recife, maio de 2004

Aos que são minha referência, minha felicidade, minha vida:
Deus; meus pais: Socorro e Valdecy; meus irmãos: Valdiléia e
Jorge Washington; meu namorado, amigo e cúmplice: Adjamir.

AGRADECIMENTOS

“It seems to me shallow and arrogant for any person in these times to claim they are completely self-made, that they owe all of their success to their own unaided efforts. Many hands and hearts and minds generally contribute to anyone’s notable achievements.”

Walt Disney

Força é toda causa capaz de produzir ou acelerar movimentos. Todas as conquistas são repletas de dificuldades e, para vencê-las, é necessário muita **força**.

Cada uma das pessoas mencionadas aqui, em algum momento da elaboração deste trabalho, me deu algum tipo de **força**, seja através de conversas, orações ou simplesmente pelo fato de estarem presentes em minha vida.

Essa **força** foi determinante para que eu chegasse até o final desta batalha. Por isso, esta conquista não é apenas minha, mas também de cada um que me impulsionou a chegar até aqui. Sendo assim, gostaria de dedicar meus sinceros agradecimentos:

- *A Santíssima Trindade — Pai, Filho e Espírito Santo — e a Nossa Senhora, pela **força espiritual** que inspira, protege e ilumina constantemente a minha vida;*
- *Aos meus amados pais, Socorro e Valdecy, por me tornarem quem eu sou, pelo amor, pela dedicação, por todos os sacrifícios para que eu chegasse até aqui e por terem sido **fortes**, mesmo quando a minha distância os fazia sofrer;*
- *Aos meus queridos irmãos, Valdiléia e Jorge Washington, pela torcida e apoio, por ajudarem a me conhecer melhor através das nossas diferenças e pela compreensão nos momentos em que não pude estar presente;*
- *A Adjamir, por ser meu lar, minha alegria, meu anjinho da guarda; por me fazer vivenciar um amor cúmplice e verdadeiro; e por despertar o que há de melhor em mim na*

vida profissional e pessoal. Queria agradecê-lo especialmente pelo apoio incondicional e pela paciência, tendo abdicado muitas vezes de todo seu tempo livre para estar comigo e me ajudar a vencer esta batalha.

- A minha avó Brígida, a minha tia Teresa e a minha sogrinha Ana Maria pelas orações em minha intercessão nos momentos mais difíceis do mestrado;
- As minhas irmãs de coração, Ayla e Ellen, por serem minha família em Recife, pela maravilhosa convivência diária e por todos os momentos inesquecíveis que vivemos juntas;
- Aos meus amigos, especialmente a Giselle, por assim continuarem, apesar da ausência constante em suas vidas. E aos novos amigos que conheci durante o mestrado: Tiago (Baiano), Cledja, Rohit, João, Enoque, Fábio Rilston, Hélio, Cléber, Vivianne e toda galera do INOCOOP;
- Ao meu orientador, Jaelson, pelo incentivo, confiança e pela parceria nas idéias e discussões;
- Ao professor Paulo Borba e ao doutorando Sérgio Soares, por terem me propiciado o primeiro contato com a orientação a aspectos e pelas importantes contribuições para o enriquecimento desta dissertação;
- Ao professor Júlio Leite, por seus comentários gentis e enriquecedores a respeito desta dissertação e, especialmente, pela preciosa motivação em dar continuidade a este trabalho;
- A todos os participantes das listas de discussão aosd, aosd-br e re-online, pela troca de conhecimentos determinantes na elaboração deste trabalho;
- Aos pesquisadores com os quais me correspondi, especialmente a Christina Von Flach, Ana Moreira e João Araújo;

- *Aos professores José Tadeu e a Daniela Coelho, pela orientação e incentivo durante a iniciação científica na UFPB, e por tudo que aprendi nesse período;*
- *Aos meus professores do curso técnico de Processamento de Dados na Escola Técnica Federal da Paraíba: Leônidas Francisco, por ter me mostrado que a humildade engrandece ainda mais um profissional competente; e Frederico Costa, por ter acreditado no meu potencial e aberto as portas da iniciação científica na UFPB;*
- *Ao Conselho Nacional de Pesquisa (Cnpq), pelo suporte financeiro, essencial para realização deste trabalho;*
- *Ao Centro de Informática, pelo suporte técnico e acadêmico durante esses dois anos;*
- *A todos aqueles que torceram e acreditaram em mim e neste trabalho.*

*“Há duas formas para viver sua vida:
uma é acreditar que não existe milagre;
a outra é acreditar que todas as coisas são um milagre”.*

Albert Einstein

“Porque para Deus nada é impossível”.

Lucas 1,37

RESUMO

SOUSA, Geórgia Maria Carvalho de. **Uma Abordagem Direcionada a Casos de Uso para o Desenvolvimento de Software Orientado a Aspectos**. 2004. 195f. Dissertação (Mestrado em Ciência da Computação) – Programa de Pós-Graduação em Ciência da Computação, UFPE, Recife.

Sistemas grandes e complexos geralmente possuem muitas preocupações inter-relacionadas. No entanto, algumas dessas preocupações (ex: preocupações não-funcionais e regras de negócio), denominadas de preocupações transversais (ou aspectos), não conseguem ser separadas adequadamente utilizando mecanismos da orientação a objetos. Esse tipo de preocupação tende a ficar espalhada em várias unidades do sistema e/ou misturada com preocupações específicas de cada uma dessas unidades. Isso aumenta o acoplamento entre artefatos e, conseqüentemente, prejudica a compreensão, a manutenção e o reuso dos artefatos envolvidos.

O paradigma orientado a aspectos surge nesse contexto oferecendo mecanismos que possibilitam a separação de preocupações transversais em artefatos de software. Esse é um paradigma emergente e, a maioria dos esforços nessa área concentrou-se, inicialmente, em atividades voltadas para implementação. A tendência atual é incorporar fundamentos e práticas desse paradigma em atividades iniciais do ciclo de desenvolvimento para que seja possível obter os benefícios de sua utilização também em artefatos de requisitos, análise e projeto. Nos últimos anos, algumas abordagens vêm sendo propostas nesse sentido. Contudo, as propostas pesquisadas geralmente focam num fluxo de desenvolvimento particular (requisitos, projeto ou implementação), não promovendo a integração com os demais.

Partindo dessa motivação, o objetivo deste trabalho é estabelecer uma abordagem que possibilite a separação de preocupações transversais desde as atividades iniciais do ciclo de desenvolvimento utilizando fundamentos do paradigma orientado a aspectos. Para alcançar esse objetivo, decidimos adaptar os fluxos de requisitos, análise e projeto do Processo Unificado. Entretanto, como esses fluxos abrangem um grande número de atividades, optamos por limitar o escopo desse trabalho ao subconjunto dessas atividades que representa a perspectiva de desenvolvimento guiado por casos de uso desse processo.

Para avaliar o uso da abordagem, aplicamos nossa proposta no contexto de um sistema de *Internet Banking*.

ABSTRACT

SOUSA, Geórgia Maria Carvalho de. **Uma Abordagem Direcionada a Casos de Uso para o Desenvolvimento de Software Orientado a Aspectos**. 2004. 195f. Dissertação (Mestrado em Ciência da Computação) – Programa de Pós-Graduação em Ciência da Computação, UFPE, Recife.

Big and complex systems generally have many interrelated concerns. However, some of these concerns (e.g. non-functional concerns and business rules), called of crosscutting concerns (or aspects), are not adequately separated using mechanisms of the object orientation. This type of concern tends to be spread in some system units and/or tangled with specific concerns of each one of these units. This increases the coupling between artifacts and, consequently, it harms the understanding, the maintenance and the reuse of the involved artifacts.

The aspect-oriented paradigm appears in this context offering mechanisms that make possible the separation of crosscutting concerns in software artifacts. This is an emergent paradigm and, the majority of efforts in this area were concentrated, initially, in activities directed toward implementation. The current trend is to incorporate the basic concepts and practices of this paradigm in early activities of the development cycle so that it is possible to also get the benefits of its use in requirements, analysis and design artifacts. In the last years, some approaches have being proposed in this direction. However, the studied proposals generally focus on a particular development workflow (requirements, design or implementation), not promoting the integration with the others.

From this motivation, the goal of this work is to establish an approach that makes possible the separation of crosscutting concerns since the early activities of the development cycle using the basic concepts of the aspect-oriented paradigm. To reach this goal, we decide to adapt the requirements, analysis and design workflow of the Unified Process. However, as these workflows enclose a great number of activities, we opt to limit the scope of this work to the subgroup of these activities that represents the use case driven development perspective of this process.

To evaluate the use of our approach, we apply it in the context of an Internet Banking system.

ÍNDICE

AGRADECIMENTOS	IV
RESUMO	VIII
ABSTRACT	IX
LISTA DE FIGURAS	XIII
LISTA DE TABELAS	XVI
1. INTRODUÇÃO	1
1.1 CONTEXTO	2
1.2 PROBLEMA	5
1.3 OBJETIVOS	6
1.4 JUSTIFICATIVA	7
1.5 ESCOPO DA DISSERTAÇÃO	8
1.6 METODOLOGIA	8
1.7 ORGANIZAÇÃO DA DISSERTAÇÃO	10
2. DESENVOLVIMENTO DE SOFTWARE GUIADO POR CASOS DE USO NO PROCESSO UNIFICADO	12
2.1 VISÃO GERAL DO PROCESSO UNIFICADO	13
2.2 FLUXO DE REQUISITOS	15
2.2.1 Identificar atores e casos de uso	16
2.2.2 Especificar casos de uso	17
2.2.3 Estruturar o modelo de casos de uso	19
2.3 FLUXO DE ANÁLISE	23
2.3.1 Analisar casos de uso	24
2.3.2 Analisar classes	27
2.4 FLUXO DE PROJETO	28
2.4.1 Projetar casos de uso	29
2.4.2 Projetar classes	31
2.5 TRATAMENTO DE REQUISITOS NÃO-FUNCIONAIS	31
2.6 CONSIDERAÇÕES FINAIS	32
3. DESENVOLVIMENTO DE SOFTWARE ORIENTADO A ASPECTOS: ESTADO DA ARTE	34
3.1 SEPARAÇÃO DE PREOCUPAÇÕES	35
3.2 ABORDAGENS PARA SEPARAÇÃO AVANÇADA DE PREOCUPAÇÕES	37
3.2.1 Programação Orientada a Aspectos	38
3.2.2 Separação Multidimensional de Preocupações	42

3.2.3 Filtros de Composição	47
3.3 PARADIGMA ORIENTADO A ASPECTOS	48
3.3.1 Fundamentos	49
3.3.2 Identificação de aspectos.....	51
3.4 DESENVOLVIMENTO DE SOFTWARE ORIENTADO A ASPECTOS	52
3.4.1 Abordagens relacionadas à engenharia de requisitos orientada a aspectos	53
3.4.2 Abordagens relacionadas ao projeto orientado a aspectos	61
3.5 CONSIDERAÇÕES FINAIS.....	67
4. ADAPTANDO O PROCESSO UNIFICADO PARA O DESENVOLVIMENTO DE SOFTWARE ORIENTADO A ASPECTOS	69
4.1 VISÃO GERAL	70
4.2 ADAPTANDO O FLUXO DE REQUISITOS	73
4.2.1 Identificar e analisar requisitos não-funcionais	75
4.2.2 Especificar requisitos não-funcionais	78
4.2.3 Estruturar o modelo de casos de uso.....	78
4.3 ADAPTANDO O FLUXO DE ANÁLISE	88
4.3.1 Analisar casos de uso	88
4.3.2 Analisar classes.....	95
4.4 ADAPTANDO O FLUXO DE PROJETO	96
4.4.1 Projetar casos de uso.....	97
4.4.2 Projetar classes	103
4.5 CONSIDERAÇÕES FINAIS.....	105
5. ESTUDO DE CASO: INTERNET BANKING	108
5.1 OBJETIVOS	109
5.2 FLUXO DE REQUISITOS.....	110
5.2.1 Identificar atores e casos de uso	110
5.2.2 Especificar casos de uso	111
5.2.3 Identificar e analisar requisitos não-funcionais	113
5.2.4 Especificar requisitos não-funcionais	117
5.2.5 Estruturar o modelo de casos de uso.....	118
5.3 FLUXO DE ANÁLISE	130
5.3.1 Analisar casos de uso	131
5.3.2 Analisar classes.....	141
5.4 FLUXO DE PROJETO	143
5.4.1 Projetar casos de uso.....	143
5.4.2 Projetar classes	154

5.5 CONSIDERAÇÕES FINAIS.....	158
6. CONCLUSÃO.....	160
6.1 DISCUSSÃO	161
6.2 CONTRIBUIÇÕES.....	162
6.3 COMPARAÇÃO COM TRABALHOS RELACIONADOS.....	163
6.4 LIMITAÇÕES E TRABALHOS FUTUROS	164
6.5 CONSIDERAÇÕES FINAIS.....	166
REFERÊNCIAS BIBLIOGRÁFICAS	168
GLOSSÁRIO.....	176

LISTA DE FIGURAS

Figura 2.1 - Relação do modelo de casos de uso com os demais modelos (adaptado de [JACOBSON et al., 1992]).....	15
Figura 2.2 - Fluxo de atividade para capturar requisitos como casos de uso, incluindo os responsáveis por cada atividade (retirado de [JACOBSON et al., 1999])	16
Figura 2.3 - Modelo para especificação de caso de uso proposto por COCKBURN (2000).....	18
Figura 2.4 - Modelo para especificação de casos de uso proposto pelo RUP [RATIONAL, 2002].....	19
Figura 2.5 - Notações UML para mecanismos de estruturação de casos de uso	21
Figura 2.6 - Diagrama de caso de uso ilustrando diferença entre os relacionamentos de generalização e inclusão	23
Figura 2.7 - Atividades e seus responsáveis no fluxo de análise (reproduzido de [JACOBSON et al., 1999])	24
Figura 2.8- Representação do particionamento da funcionalidade de um caso de uso entre classes de análise (adaptado de [JACOBSON et al., 1992])	25
Figura 2.9 - Atividades e seus responsáveis no fluxo de projeto (reproduzido de [JACOBSON et al., 1999])	28
Figura 2.10 - Exemplo de substituição de subfluxos num diagrama de seqüência por subsistemas	30
Figura 3.1 - Exemplo de espalhamento e entrelaçamento de preocupação transversal na hierarquia de classes	38
Figura 3.2 - Processo de combinação de aspectos e classes para gerar o sistema	40
Figura 3.3 - Estrutura de uma entidade <i>aspect</i> em AspectJ	41
Figura 3.4 - Exemplo de código em AspectJ.....	41
Figura 3.5 - Exemplo de possíveis dimensões de um hiper-espço.....	43
Figura 3.6 - Exemplo em Hyper/J de mapeamento entre dimensões de preocupações (retirado de [OSSHER & TARR, 2000])	45
Figura 3.7 - Exemplo de hiper-espço em <i>Hyper/J</i> (adaptado de [OSSHER & TARR, 2000])	46
Figura 3.8 - Exemplo de declaração de composição entre <i>hyperslices</i> em <i>Hyper/J</i>	46
Figura 3.9 - Ilustração do funcionamento da separação e composição de aspectos	50
Figura 3.10 - Modelo de Processo AORE (adaptado de [RASHID et. al, 2003])	55
Figura 3.11 - Exemplo de composição em diagrama de caso de uso (adaptado de [BRITO & MOREIRA, 2003]).....	56
Figura 3.12 - Exemplo de declaração de combinação (retirado de [SUZUKI & YAMAMOTO, 1999]).....	61
Figura 3.13 - Representação gráfica da modelagem de aspectos segundo ZAKARIA et al. (2002)	62
Figura 3.14 - Representação gráfica da modelagem de aspectos segundo PAWLAK et al. (2002)	64
Figura 3.15 - Exemplo de <i>template</i> de pacote UML parametrizável para modelar o aspecto de rastreamento (retirado de [CLARKE & WALKER, 2001])	65
Figura 3.16 - Exemplo de padrão de comportamento para o aspecto de <i>Trace</i> (retirado de [CLARKE & WALKER, 2001])	65

Figura 3.17 - Exemplo de composição do aspecto de <i>Trace</i> usando o relacionamento de <i>binding</i> (retirado de [CLARKE & WALKER, 2001])	66
Figura 4.1- Visão geral sobre tabela de composição utilizada na nossa proposta	73
Figura 4.2 - Atividades de requisitos da nossa proposta	74
Figura 4.3 Comparação entre composição de preocupações não-funcionais nas abordagens de AORE (a) e na nossa proposta (b).....	77
Figura 4.4 - Visão geral das atividades do <i>framework</i> NFR.....	78
Figura 4.5 - Especificação do caso de uso <i>Fazer matrícula em disciplina</i>	82
Figura 4.6 - Exemplo de relacionamento do tipo <i>crosscuts</i>	82
Figura 4.7 - Roteiro para o preenchimento da tabela de composição nível 1	83
Figura 4.8 - Exemplo de caso de uso de extensão (adaptado de [RATIONAL, 2002])	86
Figura 4.9 - Exemplo de caso de uso de extensão (adaptado de [JACOBSON et al., 1992])	86
Figura 4.10 - Representação gráfica de uma classe de análise aspectual.....	89
Figura 4.11 - Diagrama de colaboração para o caso de uso <i>Fazer matrícula em disciplina</i>	90
Figura 4.12- Roteiro para o preenchimento da tabela de composição nível 2.....	92
Figura 4.13 - Ilustração de como uma mensagem será interceptada, dependendo da regra de composição	94
Figura 4.14 - Visão do diagrama de colaboração do caso de uso <i>Fazer matrícula em disciplina</i> COM a presença do objeto aspectual <i>VerificacaoDebito</i>	95
Figura 4.15 - Diagrama de colaboração para realização da responsabilidade "verificar débito"	96
Figura 4.16 - Exemplo de diagrama de seqüência para primeira parte do caso de uso <i>Fazer matrícula em disciplina</i>	99
Figura 4.17 - Roteiro para o preenchimento da tabela de composição nível 3.....	101
Figura 4.18 - Visão do diagrama de seqüência da primeira parte do caso de uso <i>Fazer matrícula em disciplina</i> COM a presença do objeto de projeto aspectual <i>VerificacaoDebito</i>	103
Figura 4.19 - Exemplo de diagrama de seqüência para realização da operação <i>verificarDebito</i> da classe de projeto aspectual <i>VerificacaoDebito</i>	104
Figura 4.20 - Definição da estrutura da classe de projeto aspectual <i>VerificacaoDebito</i>	105
Figura 5.1 - Decomposição da preocupação segurança em outras preocupações	114
Figura 5.2 - Decomposição da preocupação de segurança em operacionalizações	115
Figura 5.3 - Diagrama de caso de uso após estruturação	121
Figura 5.4 - Diagrama de colaboração para caso de uso <i>Visualizar saldo de conta</i>	133
Figura 5.5 - Diagrama de colaboração para caso de uso <i>Visualizar extrato de conta</i>	133
Figura 5.6 - Diagrama de colaboração para caso de uso <i>Realizar Transferência</i>	134
Figura 5.7 - Visão do diagrama de colaboração do caso de uso <i>Visualizar saldo de conta</i> COM a presença dos objetos de análise aspectuais	139
Figura 5.8 - Visão do diagrama de colaboração do caso de uso <i>Visualizar extrato de conta</i> COM a presença dos objetos de análise aspectuais.....	140
Figura 5.9 - Visão do diagrama de colaboração do caso de uso <i>Realizar transferência</i> COM a presença dos objetos de análise aspectuais	140
Figura 5.10 - Realização da responsabilidade "cobrar tarifa" da classe de análise aspectual <i>CobrancaTransacaoExtra</i>	141

Figura 5.11 - Realização da responsabilidade "cobrar CPMF" da classe de análise aspectual <i>CobrancaCPMF</i>	142
Figura 5.12 - Realização da responsabilidade "obter saldo disponível CPMF" da classe de análise aspectual <i>CobrancaCPMF</i>	142
Figura 5.13 - Realização da responsabilidade "registrar" da classe de análise aspectual <i>GravacaoInfoTransacao</i>	142
Figura 5.14 - Realização da responsabilidade "cadastrar" da classe de análise aspectual <i>Bonificacao</i>	142
Figura 5.15 - Realização da responsabilidade "cheçar" da classe de análise aspectual <i>ChecagemSenhalB</i>	143
Figura 5.16 - Realização da responsabilidade "cheçar" da classe de análise aspectual <i>ChecagemDadosPessoais</i>	143
Figura 5.17 - Diagrama de seqüência do caso de uso <i>Visualizar saldo de conta</i>	147
Figura 5.18 - Diagrama de seqüência do caso de uso <i>Visualizar extrato de conta</i>	147
Figura 5.19 - Diagrama de seqüência para a primeira parte do caso de uso <i>Realizar transferência</i>	148
Figura 5.20 - Diagrama de seqüência para a segunda parte do caso de uso <i>Realizar transferência</i>	148
Figura 5.21 - Visão do diagrama de seqüência do caso de uso <i>Visualizar saldo de conta</i> COM a presença dos objeto de projeto aspectuais que afetam a interação	154
Figura 5.22 - Diagrama de seqüência para realização da operação <i>saldoDisponivelCPMF</i> da classe de projeto aspectual <i>CobrancaCPMF</i>	155
Figura 5.23 - Estrutura da classe de projeto aspectual <i>CobrancaCPMF</i>	157
Figura 5.24 - Diagrama de classe enfatizando relacionamento da classe de projeto aspectual <i>CobrancaCPMF</i> e as clases afetadas por ela	158

LISTA DE TABELAS

Tabela 3.1 - Exemplos de tipos de filtros pré-definidos e sua semântica [BERGMANS & AKSIT, 2001]....	48
Tabela 3.2 - Matriz para relacionar preocupações com requisitos funcionais (retirado de [RASHID et. al, 2003]).....	53
Tabela 3.3 - Exemplo de Saída da Atividade <i>Especificar Dimensão de Aspectos Candidatos</i> do modelo AORE (retirado de [RASHID et. al, 2002])	54
Tabela 3.4 - Equivalências entre conceitos de extensão de casos de uso e os de orientação a aspectos (extraído de [JACOBSON, 2003b])	57
Tabela 3.5 - Comparação entre abordagens pesquisadas para modelar aspectos no fluxo de requisitos..	60
Tabela 3.6 - Comparação entre algumas abordagens para modelar aspectos no nível de projeto	67
Tabela 4.1- Modelo conceitual proposto para representar os fundamentos do paradigma orientado a aspectos nos fluxos de requisitos, análise e projeto do Processo Unificado.....	71
Tabela 4.2 - Tabela de composição para um requisito aspectual (tabela de composição nível 1)	83
Tabela 4.3 - Exemplo de especificação da tabela de composição nível 1 para requisito aspectual <i>Verificação de débito</i>	84
Tabela 4.4 - Tabela de conflitos entre requisitos aspectuais	84
Tabela 4.5 - Exemplo de preenchimento da tabela de conflitos	85
Tabela 4.6 - Exemplo de especificação do caso de uso base segundo a UML	87
Tabela 4.7 - Exemplo de especificação do caso de uso de extensão segundo a UML	87
Tabela 4.8 - Tabela de composição para classe de análise aspectual (tabela de composição nível 2) ...	91
Tabela 4.9 - Tabela de composição para a classe de análise aspectual <i>VerificacaoDebito</i>	93
Tabela 4.10 - Mapeamento entre classes de análise necessárias para realizar o caso de uso <i>Fazer matrícula em disciplina</i>	97
Tabela 4.11 - Mapeamento entre classes de análise necessárias para realizar o requisito aspectual <i>VerificacaoDebito</i>	98
Tabela 4.12 -Tabela de composição para classe de projeto aspectual ou tabela de composição nível 3.	99
Tabela 4.13 - Tabela de composição nível 3 para classe de projeto aspectual <i>VerificacaoDebito</i>	102
Tabela 4.14 - Tabela de composição nível 1 (perspectiva do aspecto)	106
Tabela 4.15 - Tabela de composição nível 1 com perspectiva de caso de uso	106
Tabela 5.1 - Especificação do caso de uso <i>Efetuar Login</i>	111
Tabela 5.2 - Especificação do caso de uso <i>Visualizar saldo de conta</i>	111
Tabela 5.3 - Especificação do caso de uso <i>Visualizar extrato de conta</i>	112
Tabela 5.4 - Especificação do caso de uso <i>Realizar Transferência</i>	112
Tabela 5.5 - Especificação do caso de uso <i>Pagar boleto bancário</i>	113
Tabela 5.6 - Especificação do caso de uso <i>Checar senha InternetBanking</i>	117
Tabela 5.7 - Especificação do caso de uso <i>Checar dados pessoais do cliente</i>	118
Tabela 5.8 - Especificação do caso de uso <i>Limitar valor de movimentação</i>	118
Tabela 5.9 - Especificação do cenário principal do caso de uso <i>Visualizar saldo de conta (após estruturação)</i>	122

Tabela 5.10 - Especificação do cenário principal do caso de uso <i>Visualizar extrato de conta (após estruturação)</i>	122
Tabela 5.11 - Especificação do cenário principal do caso de uso <i>Realizar Transferência</i> incluindo referências a requisitos aspectuais.....	123
Tabela 5.12 - Especificação do cenário principal do caso de uso <i>Realizar Transferência (após estruturação)</i>	124
Tabela 5.13 - Especificação do cenário principal do caso de uso <i>Pagar boleto bancário (após estruturação)</i>	124
Especificar tabe	124
Tabela 5.14 - Tabela de composição nível 1 para requisito aspectual <i>Cobrar tarifa por transação extra</i>	125
Tabela 5.15 - Tabela de composição nível 1 para requisito aspectual <i>Gravar informações da transação</i>	126
Tabela 5.16 - Tabela de composição nível 1 para o requisito aspectual <i>Cobrar CPMF</i>	126
Tabela 5.17 - Tabela de composição nível 1 para o requisito aspectual <i>Oferecer bonificação</i>	126
Tabela 5.18 - Tabela de composição nível 1 para o requisito aspectual <i>Checar senha Internet Banking</i>	127
Tabela 5.19 - Tabela de composição nível 1 para o requisito aspectual <i>Checar dados pessoais do cliente</i>	127
Tabela 5.20 - Tabela de composição nível 1 para o requisito aspectual <i>Limitar valor de movimentação</i>	127
Tabela 5.21 - Tabela de composição nível 1 com perspectiva do caso de uso <i>Visualizar saldo de conta</i>	128
Tabela 5.22 - Tabela de composição nível 1 com perspectiva do caso de uso <i>Visualizar extrato de conta</i>	128
Tabela 5.23 - Tabela de composição nível 1 com perspectiva do caso de uso <i>Realizar transferência</i>	129
Tabela 5.24 - Tabela de composição nível 1 com perspectiva do caso de uso <i>Pagar boleto bancário</i>	129
Tabela 5.25 - Tabela de conflitos.....	130
Tabela 5.26 - Mapeamento entre casos de uso e classes de análise	131
Tabela 5.27 - Mapeamento entre requisitos aspectuais e classes de análise	132
Tabela 5.28 - Tabela de composição nível 2 para classe de análise aspectual <i>CobrancaCPMF</i>	135
Tabela 5.29 - Tabela de composição nível 2 para classe de análise aspectual <i>CobrancaTransacaExtra</i>	136
Tabela 5.30 - Tabela de composição nível 2 para classe de análise aspectual <i>GravacaoInfoTransacao</i>	136
Tabela 5.31 - Tabela de composição nível 2 para classe de análise aspectual <i>Bonificacao</i>	137
Tabela 5.32 - Tabela de composição nível 2 para classe de análise aspectual <i>ChecagemSenhaIB</i>	137
Tabela 5.33 - Tabela de composição nível 2 para classe de análise aspectual <i>ChecagemDadosPessoais</i>	138
Tabela 5.34 - Tabela de composição nível 2 para classe de análise aspectual <i>VerificacaoLimite</i>	138
Tabela 5.35 - Mapeamento entre classes de análise e classes de projeto	145
Tabela 5.36 - Tabela de composição nível 3 para classe de projeto aspectual <i>CobrancaCPMF</i>	150
Tabela 5.37 - Tabela de composição nível 3 para classe de projeto aspectual <i>CobrancaTransacaExtra</i>	151
Tabela 5.38 - Tabela de composição nível 3 para classe de projeto aspectual <i>GravacaoInfoTransacao</i>	151
Tabela 5.39 - Tabela de composição nível 3 para classe de projeto aspectual <i>Bonificacao</i>	152
Tabela 5.40 - Tabela de composição nível 3 para classe de projeto aspectual <i>ChecagemSenhaIB</i>	152
Tabela 5.41 - Tabela de composição nível 3 para classe de projeto aspectual <i>ChecagemDadosPessoais</i>	153
Tabela 5.42 - Tabela de composição nível 3 para classe de análise aspectual <i>VerificacaoLimite</i>	153

Tabela 5.43 - Tabela de composição nível 3 para classe de projeto aspectual CobrançaCPMF (incluindo adição estrutural)	156
---	-----

1

INTRODUÇÃO

*The best preparation for good work
tomorrow is to do good work today.*

Elbert Hubbard

Neste capítulo apresentamos uma visão geral sobre a dissertação. Inicialmente, descrevemos o contexto no qual este trabalho de pesquisa se insere. A seguir, discutimos qual o problema que pretendemos oferecer contribuições e quais os objetivos e finalidades da pesquisa. Por fim, apresentamos a metodologia de pesquisa adotada, delimitamos o escopo da dissertação e descrevemos a estrutura dos capítulos deste trabalho.

1.1 Contexto

Cada vez mais o software tem assumido um papel essencial e crítico em nossa sociedade [PRESSMAN, 2001]. À medida que essa importância cresce, também aumenta a necessidade de desenvolver e gerenciar o processo de criação e evolução de produtos de software num menor tempo possível, com menores custos e com alta qualidade. A compreensibilidade, a manutenibilidade e a reusabilidade dos artefatos de um produto de software estão entre as características mais importantes que contribuem para satisfazer a essa necessidade [JACOBSON et al., 1992].

Compreensibilidade é a possibilidade de estudar cada artefato individualmente, sem que para isso seja necessário entender todo o sistema ou buscar informações em outro artefato [PARNAS, 1972]. No contexto atual, em que sistemas possuem um grande número de preocupações fortemente relacionadas e interdependentes, alcançar uma completa compreensibilidade de artefatos torna-se cada vez mais difícil.

Além da complexidade, outra propriedade característica dos sistemas de software modernos é a mutabilidade [BROOKS, 1987]. O fato da mudança ser algo inevitável no ciclo de vida de sistemas de informação é uma das mais importantes premissas da Engenharia de Software moderna [JACOBSON et al., 1992]. Mudanças são necessárias para reparar erros ou para refletir alterações nas necessidades dos clientes, no ambiente no qual o sistema está inserido, nas regras de negócios, nas leis ou nos regulamentos que o sistema deve seguir [KOTONYA & SOMMERVILLE, 1998]. Tais mudanças, geralmente, correspondem a mais de 60% do custo total do software [SOMMERVILLE, 1996]. Sendo assim, o software deve ser desenvolvido de maneira que seja flexível e adaptável o suficiente para que o impacto dessas mudanças seja o menor possível. Isso torna a manutenibilidade e a extensibilidade uma das principais qualidades de software. Quanto mais fácil for adicionar, remover ou modificar uma característica existente num sistema e quanto menor o número de artefatos que precisará ser alterado para esse fim, menor será o custo e a probabilidade de que novos erros sejam introduzidos quando mudanças são realizadas [SOMMERVILLE, 1996].

Por sua vez, o reuso de artefatos é importante especialmente para melhorar a produtividade do processo de desenvolvimento de software. Utilizar resultados obtidos

em projetos anteriores é um desejo comum em todo trabalho de desenvolvimento. Embora a codificação seja o contexto em que usualmente se preocupe com a reusabilidade, o reuso de artefatos em atividades iniciais de desenvolvimento, tais como requisitos, análise e projeto, pode propiciar ainda mais produtividade ao processo [JACOBSON et al., 1992].

Uma maneira de alcançar as qualidades discutidas anteriormente é utilizar o princípio da separação de preocupações [DIJKSTRA, 1976]. Isso porque, ao ser aplicado adequadamente, esse princípio contribui para [GHEZZI et al., 1991]: (i) melhorar a compreensibilidade dos artefatos de software; (ii) limitar o impacto da mudança; e (iii) facilitar a evolução, adaptação, customização e reuso dos artefatos de software.

O princípio da separação de preocupações estabelece que, para lidar com a complexidade, deve-se dividir um problema em unidades menores, claramente separadas, de maneira que cada uma delas represente uma única preocupação [AKSIT et al., 2001]. A escolha crucial, nesse caso, é como separar as preocupações e representar o relacionamento entre elas [DIJKSTRA, 1976]. Geralmente essa escolha é baseada no paradigma de programação adotado. Em métodos orientados a objetos, por exemplo, as preocupações são separadas em objetos; em métodos estruturados, elas são separadas em procedimentos. A orientação a objetos destaca-se nesse sentido por possibilitar que o sistema seja decomposto em unidades mais próximas daquelas existentes no domínio da aplicação e por fornecer mecanismos como herança, polimorfismo e encapsulamento [AMBLER, 2001].

A partir de meados da década de 90, o paradigma orientado a objetos consolidou-se como padrão de desenvolvimento de software e seu uso vem crescendo desde então [AMBLER, 2001]. No entanto, recentemente, algumas limitações vem sendo identificadas em métodos orientados a objetos [HARRISON & OSSHER, 1993; KICZALES et al., 1997; BERGMANS & AKSIT, 2000], especialmente no que se refere à separação de preocupações que devem ser aplicadas em diversas unidades do sistema (ex: preocupações não-funcionais e regras de negócio). Esse tipo de preocupação é denominado de preocupação transversal (ou aspecto) porque, utilizando mecanismos da

orientação a objetos, geralmente fica espalhado em diversas unidades que possuem preocupações diferentes da sua.

Por exemplo, usando métodos orientados a objetos não é possível separar a preocupação de controle de acesso das unidades em que essa preocupação deve ser considerada: o desenvolvedor deve inserir referências à checagem de direitos na especificação dos casos de uso e nas classes que devem considerar essa preocupação. Esse fato compromete o alcance dos benefícios de compreensibilidade, manutenibilidade e reusabilidade, obtidos com a separação de preocupações. A compreensibilidade é prejudicada porque complexidade extra é inserida nas unidades. A manutenibilidade é comprometida visto que, tanto ao adicionar como ao tentar excluir uma preocupação transversal no sistema, há uma sobrecarga para o engenheiro de software, pois ele tem que lembrar de inserir/retirar referências a essa preocupação em todos os lugares onde elas são necessárias. Por sua vez, as potencialidades de reuso integral dos artefatos diminuem, pois as especificações das unidades são misturadas com preocupações que não dizem respeito ao objetivo da unidade.

Para tentar solucionar essas limitações do paradigma orientado a objetos, foram propostas algumas abordagens na literatura, conhecidas como “abordagens para separação avançada de preocupações”: Programação Adaptativa [LIEBERHERR, 1996], Filtros de Composição [BERGMANS & AKSIT, 2000; BERGMANS et al., 2001], Separação Multidimensional de Preocupações [TARR et al., 1999; OSSHER & TARR, 2000] e Programação Orientada a Aspectos [KICZALES et al., 1997]; sendo essa última a que tem atraído mais atenção atualmente [VIEGA & VUAS, 2000]. O conjunto de conceitos, mecanismos e práticas comuns a essas abordagens está formando o que a comunidade de Engenharia de Software vem denominando de paradigma orientado a aspectos [AOSD, 2003].

Assim como o paradigma orientado a objetos (POO) não descarta as idéias de dados e operações do paradigma estrutural, o paradigma orientado a aspectos (POA) não rejeita a tecnologia existente [ELRAD et al., 2001a]. O paradigma orientado a aspectos complementa o paradigma orientado a objetos a fim de auxiliar na separação daquelas preocupações que o POO manipula deselegantemente. Para isso, o paradigma orientado a aspectos provê mecanismos para localizar preocupações transversais (ou

aspectos) e compor essas preocupações com as unidades que ela afeta de maneira não-invasiva, ou seja, evitando que elas fiquem explicitamente espalhadas nos artefatos de software ou misturadas com as preocupações específicas desses artefatos.

Esse é um paradigma emergente [ELRAD et al., 2001a] e, o estado atual da pesquisa nessa área pode ser comparado ao estado da pesquisa em orientação a objetos há vinte anos atrás: os conceitos básicos estão se formando e um grupo crescente de pesquisadores está usando esses conceitos em seus trabalhos [KICZALES et al., 1997].

1.2 Problema

As motivações para o surgimento do paradigma orientado a aspectos originaram-se, principalmente, a partir de problemas encontrados na codificação de sistemas. Por isso, a maior parte da pesquisa nessa área tem sido focada nas atividades orientadas a implementação. Atualmente, a orientação a aspectos vem sendo utilizada quase exclusivamente na codificação de sistemas através de linguagens como AspectJ [KICZALES et al., 2001] e Hyper/J [OSSHER & TARR, 2000].

Recentemente, a comunidade de Engenharia de Software tem se interessado em propagar a utilização dos fundamentos e práticas desse paradigma para as atividades iniciais do ciclo de desenvolvimento [ELRAD et al., 2001a]. Algumas das razões que impulsionam essa área de pesquisa são:

- antecipar a identificação e a modelagem de aspectos;
- antecipar o raciocínio sobre que unidades devem ser afetadas por um aspecto e como a composição entre o aspecto e as unidades afetadas por ele deve ser realizada;
- permitir que os benefícios da utilização do paradigma orientados a aspectos sejam obtidos ao longo de todo processo de desenvolvimento e não apenas nos artefatos de implementação;
- possibilitar que o entendimento de um sistema implementado com uma linguagem orientada a aspectos possa ser obtido através de modelos de análise e projeto, ao invés de exigir que ele dependa da análise do código do sistema.

Nos últimos anos, algumas abordagens iniciais vêm sendo propostas nesse sentido. Contudo, as abordagens pesquisadas geralmente focam num fluxo de desenvolvimento particular (requisitos ou projeto), não promovendo a integração com os demais fluxos. Não foram encontrados métodos com atividades bem definidas que ofereçam diretrizes para um engenheiro desenvolver um sistema orientado a aspectos desde as atividades de requisitos até projeto. Esse fato foi a principal motivação para o desenvolvimento desta dissertação.

1.3 Objetivos

Motivados pelo problema exposto na seção anterior, as contribuições que este trabalho pretende oferecer foram divididas em objetivos gerais e específicos. O objetivo geral sintetiza o que pretendemos alcançar. Os objetivos específicos tornam o objetivo geral operacional e indica o que será realizado especificamente na pesquisa.

O objetivo geral desta dissertação é estabelecer uma abordagem que utilize os fundamentos do paradigma orientado a aspectos para possibilitar a separação de preocupações transversais desde atividades iniciais do ciclo de desenvolvimento.

A partir desse objetivo geral, definimos os seguintes objetivos específicos:

- adaptar os fluxos de requisitos, análise e projeto do Processo Unificado [JACOBSON et al., 1999], utilizando fundamentos do paradigma orientado a aspectos para que preocupações transversais possam ser centralizadas numa única entidade e sua composição possa ser realizada de maneira não-invasiva (i.e. evitando que preocupações transversais fiquem explicitamente espalhadas nos artefatos de software ou misturadas com as preocupações específicas desses artefatos). Dessa maneira, a compreensão, o reuso e a manutenibilidade dos artefatos obtidos nesses fluxos podem ser potencialmente melhorados;
- permitir que a nossa abordagem seja flexível e possa ser utilizada tanto para dar suporte a uma implementação orientada a aspectos quanto para uma implementação tradicional. Para isso, a adaptação não deve se prender aos mecanismos de uma abordagem ou de uma linguagem orientada a aspectos

específica e sim aos conceitos fundamentais do paradigma orientado a aspectos e suas implicações;

- avaliar e discutir a utilização da abordagem num sistema que possua preocupações transversais e no qual as necessidades de reuso e manutenibilidade sejam evidentes.

1.4 Justificativa

Os fundamentos do paradigma orientado a aspectos podem ser usados de maneira complementar aos do paradigma orientado a objetos¹ [ELRAD et al., 2001b]. Por isso, o paradigma orientado a aspectos pode ser considerado uma evolução, não uma revolução, em relação ao paradigma orientado a objetos e pode ser usado para melhorar técnicas existentes [KICZALES et al., 1997].

É natural, portanto, a tentativa de adaptar metodologias, processos e técnicas orientadas a objetos, ao invés de criar novas abordagens para serem utilizadas no desenvolvimento de software orientado a aspectos. Seguir essa perspectiva de adaptação oferece algumas vantagens: (i) focar no problema específico de como separar preocupações transversais, herdando da abordagem tomada como base os mecanismos para separação de preocupações não-transversais; e (ii) diminuir a dificuldade e possível resistência em usar uma nova abordagem, visto que não seria necessário aprender todo um método de desenvolvimento, e sim agregar novos conceitos e práticas. Exemplo desse fato é a linguagem AspectJ [KICZALES et al., 2001], que estende a linguagem orientada a objetos Java [GOSLING et al., 2000] adicionando construções e mecanismos específicos para o tratamento de aspectos.

Pelos motivos acima descritos, a direção que escolhemos para alcançar o objetivo geral deste trabalho — estabelecer uma abordagem que possibilite a separação de preocupações transversais desde as atividades iniciais do ciclo de desenvolvimento — foi adaptar um processo existente. A escolha pelo Processo Unificado [JACOBSON et al., 1999] como ponto de partida para nossa abordagem foi determinada principalmente por três características que esse processo possui [JACOBSON et al.,

1999]: (i) contempla a transformação de modelos e artefatos em várias atividades do desenvolvimento, incluindo requisitos, análise e projeto; (ii) adota a prática de desenvolvimento guiado por casos de uso, uma prática que visa lidar com complexidade e facilitar a mudança de requisitos; e (ii) adota a linguagem de modelagem UML (*Unified Modeling Language*) [BOOCH et al., 1999]. Outra motivação para essa escolha foi o fato desse processo estar se consolidando como padrão industrial para o desenvolvimento de software orientado a objetos [AMBLER, 2001].

1.5 Escopo da Dissertação

Conforme mencionado anteriormente, para alcançar o objetivo geral desta dissertação, decidimos adaptar os fluxos de requisitos, análise e projeto do Processo de Desenvolvimento Unificado [JACOBSON et al., 1999]. Entretanto, como esse processo abrange um grande número de atividades, optamos por limitar nosso escopo de adaptação ao subconjunto dessas atividades que representa a perspectiva de desenvolvimento guiado por casos de uso desse processo (inicialmente descrita por JACOBSON et al. (1992) e esboçada no Capítulo 2). As atividades dedicadas a questões arquiteturais, assim como reflexões sobre o impacto das adaptações na perspectiva iterativa e incremental do Processo Unificado estão fora do escopo do nosso trabalho.

Também decidimos focar apenas na identificação e tratamento de aspectos que precisam ser inseridos em cenários de casos de uso, alterando seu fluxo de execução. Por isso, aspectos como distribuição, persistência e concorrência, tipicamente tratados apenas a partir das atividades de projeto, não serão abordados nesse trabalho.

1.6 Metodologia

A pesquisa em Engenharia de Software pode ser conduzida através de quatro métodos: o científico, o de engenharia, o experimental e o analítico [WOHLIN et al., 2000 *apud* TRAVASSOS et al., 2002].

¹ Mas não se limita ao paradigma orientado a objetos, podendo ser usado também em conjunto com outros paradigmas como o estruturado e o funcional [CONSTANTINIDES & HASSOUN, 2002].

O método científico foca na observação de um ambiente e na tentativa de extrair dele algum modelo ou teoria que possa explicar o fenômeno estudado. O método de engenharia, por sua vez, é baseado na observação de soluções existentes, na identificação de problemas nessas soluções e na sugestão de abordagens para melhorar as soluções analisadas. Já o método experimental pretende fornecer um modelo novo para solução de um problema e tenta estudar o impacto desse modelo. Por fim, o método analítico oferece uma base analítica (ou matemática) para o desenvolvimento de modelos.

Este trabalho de pesquisa seguiu o método de engenharia.

Inicialmente foi feito um estudo sobre três das abordagens mais difundidas na área de separação avançada de preocupações (Filtros de Composição [BERGMANS & AKSIT, 2000; BERGMANS et al., 2001], Separação Multidimensional de Preocupações [TARR et al., 1999; OSSHER & TARR, 2000] e Programação Orientada a Aspectos [KICZALES et al., 1997]) para identificar os conceitos e fundamentos comuns nessas abordagens que são atribuídos ao paradigma orientado a aspectos e que serão utilizados como base da nossa abordagem.

O passo seguinte foi fazer uma investigação crítica das abordagens nas áreas de engenharia de requisitos orientada a aspectos e projeto orientado a aspectos. Tendo feito isso, foram selecionadas algumas práticas dessas abordagens para serem usadas em nossa proposta e identificados pontos que precisavam ser melhorados nesses trabalhos.

Um dos primeiros pontos que buscamos melhorar nas abordagens de requisitos foi a maneira com que aspectos não-funcionais eram manipulados nas propostas pesquisadas. Nessas abordagens eles eram expressados como atributos abstratos e subjetivos (ex: “segurança”, “performance”, “disponibilidade”, etc), o que não oferecia informações suficientes para avaliar como esses aspectos iriam impactar os casos de uso que eles afetavam. Desta maneira, propomos a utilização do *framework* NFR [MYLOPOULOS et al., 1992; CHUNG et al., 2000; MYLOPOULOS et al., 2001] para decompor preocupações não-funcionais em operacionalizações que indicariam como de fato elas deveriam ser concretizadas no sistema. Sendo assim, em nossa abordagem, a composição é feita com as operacionalizações e não com atributos abstratos e

subjetivos. Os primeiros resultados dessa fase da pesquisa foram publicados em [SOUSA et al., 2003].

A seguir, procuramos melhorar a composição dessas operacionalizações com os casos de uso nos quais elas deveriam ser consideradas, propondo para isso o conceito de tabelas de composição e utilizando os operadores de composição propostos por MOREIRA et al. (2002). Essa proposta foi apresentada em [SOUSA & CASTRO, 2003].

Considerando os conhecimentos adquiridos e os resultados preliminares alcançados, estudamos os fluxos de requisitos, análise e projeto do Processo Unificado e examinamos em quais atividades poderíamos inserir os fundamentos de orientação a aspectos para melhorar a separação de preocupações transversais. Desta maneira, propomos algumas adaptações nessas atividades, assim como relatado em [SOUSA et al., 2004].

A fim de avaliar o trabalho, realizamos um estudo de caso no qual aplicamos a nossa proposta de adaptação num sistema de *Internet Banking*.

1.7 Organização da Dissertação

Além deste capítulo introdutório e de um glossário que incluímos no final da dissertação, este trabalho é composto pelos seguintes capítulos:

- Capítulo 2: apresenta uma visão geral sobre o Processo Unificado e descreve o subconjunto das atividades focadas na perspectiva direcionada a casos de uso desse processo nos fluxos de requisitos, análise e projeto, conforme originalmente apresentado por JACOBSON et al. (1992). Essas atividades serão utilizadas como base desse trabalho;
- Capítulo 3: apresenta uma visão geral do estado da arte do desenvolvimento de software orientado a aspectos. Inicialmente, descreve o princípio da separação de preocupações, principal alicerce do paradigma orientado a aspectos. Em seguida, detalha as motivações para o surgimento desse paradigma de desenvolvimento, descrevendo os problemas que ele se propõe a solucionar e algumas das abordagens que o fundamentaram. Na sequência,

apresenta uma coletânea dos fundamentos comumente atribuídos ao paradigma orientado a aspectos e as principais linguagens utilizadas atualmente. Por fim, faz uma revisão bibliográfica sobre propostas para dar suporte a esse paradigma nas atividades de requisitos e projeto;

- Capítulo 4: detalha a proposta desse trabalho de adaptação dos fluxos de requisitos, análise e projeto do Processo Unificado para possibilitar a separação de preocupações transversais desde atividades iniciais do ciclo de desenvolvimento do sistema.
- Capítulo 5: descreve a utilização da abordagem apresentada no Capítulo 4 num sistema de *Internet Banking*. O objetivo é propiciar discussões a respeito do uso da abordagem, de maneira que possa ser analisado se as mudanças propostas ajudam a dar suporte a separação de preocupações transversais nos fluxos de requisitos, análise e projeto do Processo Unificado [JACOBSON et al., 1999].
- Capítulo 6: apresenta as conclusões, compara a abordagem proposta com trabalhos relacionados, enumera as contribuições e propõe trabalhos futuros.

DESENVOLVIMENTO DE SOFTWARE GUIADO POR CASOS DE USO NO PROCESSO UNIFICADO

A problem well-stated is a problem half-solved.

Charles Kettering

O primeiro método de desenvolvimento de software que utilizou casos de uso para guiar as atividades de desenvolvimento, desde a análise de requisitos até os testes, foi o Objectory [JACOBSON, 1987]. O objetivo desse método era auxiliar o desenvolvimento de grandes sistemas comerciais e industriais utilizando fundamentos do paradigma orientado a objetos, da modelagem conceitual [BORGIDA, 1985] e do projeto de blocos [JACOBSON, 1986]. No entanto, por ser um método proprietário, apenas informações restritas foram publicadas a seu respeito, o que limitou a disseminação da prática de desenvolvimento de software guiado por casos de uso.

Alguns anos mais tarde, JACOBSON et al. (1992) divulgou para o público geral uma versão simplificada do Objectory e seus conceitos fundamentais, denominada de OOSE (*Object Oriented Software Engineering*). Esse método tornou-se uma referência no desenvolvimento de software guiado por casos de uso e foi o responsável pela disseminação dessa prática de desenvolvimento.

Atualmente, os princípios do desenvolvimento de software guiado por casos de uso providos pelo OOSE são usados no Processo de Desenvolvimento Unificado [JACOBSON et al., 1999], um processo genérico e customizável que cobre diversas atividades do ciclo de vida do software

e utiliza a linguagem UML [BOOCH et al., 1999] como linguagem de modelagem para representar os artefatos gerados. Além de ser guiado por casos de uso, o Processo Unificado possui duas características importantes: é centrado na arquitetura e iterativo e incremental. O RUP (*Rational Unified Process*) [RATIONAL, 2002] é uma extensão do Processo Unificado de propriedade da Rational Software Corporation [RATIONAL, 2004] que acrescenta a esse processo atividades adicionais relativas à modelagem de negócio, ao gerenciamento de projeto e configuração; além de oferecer diretrizes e *templates* para as atividades cobertas pelo processo.

Nesse capítulo, apresentamos uma visão geral sobre o Processo Unificado e descrevemos o subconjunto das atividades focadas na perspectiva direcionada a casos de uso desse processo nos fluxos de requisitos, análise e projeto, conforme originalmente apresentado por JACOBSON et al. (1992). Essas atividades serão utilizadas como base desse trabalho. As perspectivas arquitetural, iterativa e incremental do Processo Unificado estão fora do escopo do nosso trabalho e não serão apresentadas nesse capítulo.

2.1 Visão geral do Processo Unificado

O Processo Unificado [JACOBSON et al., 1999] é um processo genérico que pode ser especializado para uma grande variedade de sistemas de software, para diferentes áreas de aplicação, tipos de organização e tamanhos de projeto.

Esse processo é dividido nos seguintes fluxos de atividades: requisitos, análise, projeto, implementação e testes. Em cada um desses fluxos são produzidos modelos que representam o sistema a partir de um certo nível de abstração. A linguagem de modelagem utilizada para especificar os modelos gerados ao longo desse processo é a UML [BOOCH et al., 1999]. Os principais modelos do Processo Unificado são:

- o **modelo de casos de uso**: produzido no fluxo de requisitos, esse modelo é responsável por delimitar o sistema e definir sua funcionalidade em termos de como um usuário potencial irá usar o sistema, ou seja, em termos de casos de uso;

- o **modelo de análise:** produzido no fluxo de análise, esse é um modelo conceitual dos objetos do sistema que representa o primeiro passo em direção a transformação do sistema de uma mera declaração de requisitos para uma descrição de como o sistema irá funcionar. Esse modelo assume um ambiente de implementação ideal, ou seja, não leva em consideração: SGBD, hardware, linguagem, questões de distribuição, tempo real, etc.
- o **modelo de projeto:** produzido no fluxo de projeto, esse modelo é responsável por especificar em detalhes os objetos identificados no modelo de análise de acordo com o ambiente de implementação e as tecnologias adotadas;
- o **modelo de implementação:** produzido no fluxo de implementação, esse modelo é responsável por codificar os objetos identificados no modelo de projeto utilizando uma linguagem de programação, resultando no sistema executável; e
- o **modelo de testes:** produzido no fluxo de testes, esse modelo é responsável por verificar o modelo de implementação, gerando como saída a documentação de especificações de teste e os resultados dos testes. Os testes de unidade utilizam o modelo de projeto como referência e os testes de integração baseiam-se no modelo de requisitos, especialmente nos casos de uso.

O Processo Unificado possui três características que o diferencia dos demais processos: (i) ele é guiado por casos de uso; (ii) ele é centrado na arquitetura; e (iii) ele é iterativo e incremental.

Um processo de desenvolvimento é guiado por casos de uso quando suas atividades são iniciadas a partir de casos de uso [JACOBSON et al., 1992]. Sendo assim, no Processo Unificado casos de uso não são apenas uma ferramenta para especificar os requisitos funcionais de um sistema: eles direcionam a formação de modelos nas atividades de análise, projeto, implementação e testes do sistema (Figura 2.1). Esse fato ajuda a minimizar o impacto de constantes mudanças no sistema, pois, como todos os modelos são relacionados à perspectiva de casos de uso, quando usuário

solicita alguma mudança num caso de uso, a identificação dos artefatos impactados pela mudança é facilitada [JACOBSON & CHRISTERSON, 1995].

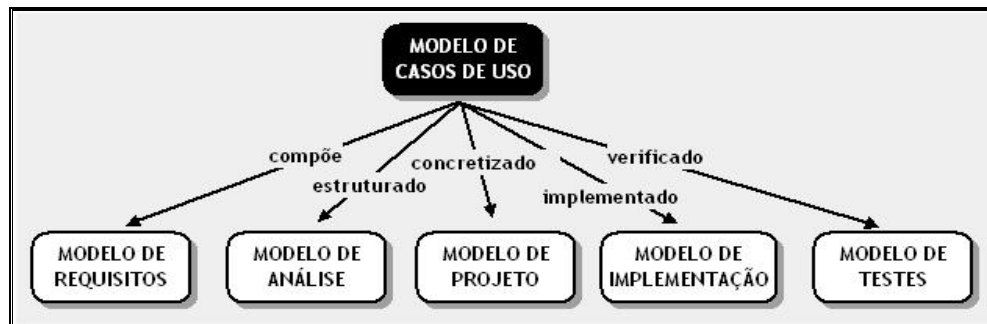


Figura 2.1 - Relação do modelo de casos de uso com os demais modelos (adaptado de [JACOBSON et al., 1992])

O fato de ser centrado na arquitetura significa que no Processo Unificado a arquitetura do sistema é usada como artefato principal para conceitualizar, construir, gerenciar e evoluir o sistema em desenvolvimento.

O Processo Unificado é iterativo porque ele divide o desenvolvimento de um sistema em uma sequência de iterações. Cada iteração pode ser pensada intuitivamente como um miniprojeto. Em cada iteração, o projetista identifica e especifica os casos de uso relevantes, cria um projeto usando a arquitetura selecionada como guia, implementa o projeto em componentes e verifica se os componentes satisfazem a estes casos de uso. Se a iteração alcança suas metas, o desenvolvimento procede para a próxima iteração. Em cada iteração, os artefatos são construídos a partir do estado em que eles foram deixados ao final da iteração anterior, por isso, esse processo também é incremental.

As seções seguintes descrevem em linhas gerais os fluxos de atividade de requisitos, análise e projeto do Processo Unificado, enfatizando apenas a perspectiva guiada por casos de uso desse processo.

2.2 Fluxo de requisitos

O principal propósito do fluxo de requisitos é capturar os requisitos em termos de casos de uso. A Figura 2.2 apresenta uma visão geral das atividades realizadas nesse fluxo. Inicialmente o analista de sistemas identifica atores e casos de uso. Em seguida, o arquiteto irá identificar os casos de uso arquiteturalmente significantes e então priorizar os casos de uso que serão desenvolvidos na iteração corrente. Na próxima atividade, os

casos de uso são descritos em mais detalhes. Se apropriado, descrições de interface são desenvolvidas para indicar como a interface deve parecer quando os casos de uso estiverem sendo executados. Por fim, o analista de sistemas reestrutura o modelo de casos de uso para torná-lo mais fácil de entender e manipular nas atividades seguintes.

As atividades em destaque na Figura 2.2 correspondem às atividades do fluxo de requisitos que serão consideradas nesse trabalho. Essas atividades serão detalhadas nas subseções a seguir. Conforme mencionamos anteriormente, nesse trabalho focamos na perspectiva de casos de uso do Processo Unificado, por isso, as atividades relacionadas à arquitetura e a interface ficaram de fora do nosso escopo.

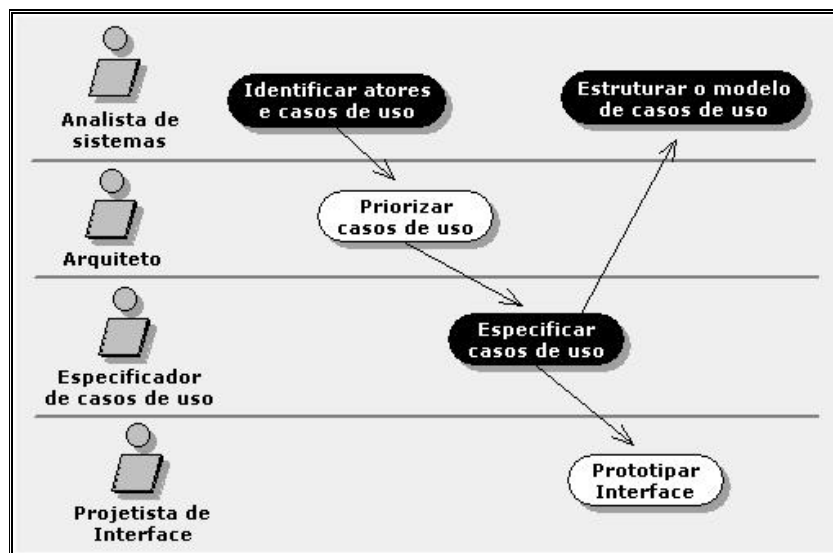


Figura 2.2 – Fluxo de atividade para capturar requisitos como casos de uso, incluindo os responsáveis por cada atividade (retirado de [JACOBSON et al., 1999])

2.2.1 Identificar atores e casos de uso

Nessa atividade são identificados atores e suas possíveis interações com o sistema.

Atores modelam qualquer entidade externa ao sistema — desde um usuário até outro sistema — que exerce algum papel na aplicação ou que precisa comunicar-se com ela. As possíveis interações entre um ator e o sistema devem ser identificadas através de casos de uso.

Em termos precisos, um caso de uso especifica uma sequência de ações, incluindo variações, que o sistema pode executar e que provoca um resultado de valor

para um ator particular [JACOBSON et al., 1999]. Essa seqüência de ações pode seguir vários caminhos, dependendo do contexto na execução do sistema. Estes caminhos são os possíveis cenários do caso de uso.

Considera-se que o caminho para realizar a funcionalidade básica do caso de uso com sucesso, sem problemas e sem erros em nenhum dos passos da seqüência, é denominado de cenário primário ou principal. Por outro lado, caminhos alternativos, bem como situações de erro, podem ser representados através de cenários secundários ou alternativos. Cenários secundários simples podem ser especificados na própria descrição do caso de uso. Porém, se o cenário secundário é bastante complexo e inclui um conjunto considerável de passos, é conveniente descrevê-lo separadamente, num outro caso de uso que estende a descrição do cenário primário [COCKBURN, 2000].

Os casos de uso são graficamente representados em diagramas, onde os atores são representados por pequenos bonecos e os casos de uso por elipses. A interação entre atores e casos de uso é representada através de linhas. Os diagramas de casos de uso proporcionam uma visão global do conjunto de casos de uso de um sistema e das suas interações com os atores.

Visto que cada ator irá realizar uma quantidade de casos de uso no sistema, a melhor maneira de encontrar casos de uso é através de atores. Desse modo, é possível identificar casos de uso analisando todos os atores, definindo os papéis de cada ator no sistema e as respostas que o sistema deve fornecer a cada um deles. A identificação de casos de uso é freqüentemente um processo muito iterativo onde várias tentativas são feitas até se estabilizar. Quando o modelo de casos de uso estiver mais estável, cada caso de uso deve ser descrito em detalhes, conforme apresentado na seção a seguir.

2.2.2 Especificar casos de uso

Ao especificar um caso de uso, primeiramente descreve-se o cenário primário e, em seguida, especifica-se variações desse cenário e erros que podem ocorrer (cenários secundários) [COCKBURN, 2000]. Normalmente um caso de uso tem um fluxo básico e vários fluxos alternativos. Apesar de casos de uso serem parte da UML [BOOCH et al., 1999], não existe nenhum modelo padrão de como especificar casos de uso.

A Figura 2.3 exibe um modelo para especificação de um caso de uso proposto por COCKBURN (2000). Este modelo requer a definição explícita do objetivo do caso de uso bem como os níveis associados com estes objetivos (contexto, usuário ou subfunção). As demais informações presentes no modelo também são importantes para tornar a descrição do caso de uso o mais clara possível.

CASO DE USO <NÚMERO>: <OBJETIVO DESCRITO NUMA FRASE CURTA INICIADA COM VERBO>	
INFORMAÇÕES CARACTERÍSTICAS	
OBJETIVO NO CONTEXTO	<uma sentença mais longa do objetivo do caso de uso, caso necessário>
ESCOPO	<qual sistema está sendo considerado (ex: organização ou sistema computacional)>
NÍVEL	<um dos tipos de objetivo: objetivo de contexto, objetivo de usuário ou objetivo de subfunção>
PRECONDIÇÕES	<o que é necessário que já esteja satisfeito para realizar o caso de uso>
CONDIÇÃO FINAL DE SUCESSO	<o que ocorre/muda após a obtenção do objetivo do caso de uso>
CONDIÇÃO FINAL DE FALHA	<o que ocorre/muda se o objetivo é abandonado>
ATOR PRIMÁRIO	<o nome do papel para o ator primário, ou descrição>
CENÁRIO PRINCIPAL DE SUCESSO	
PASSO	AÇÃO
<passo>	<descrição da ação>
CENÁRIOS ALTERNATIVOS OU EXTENSÕES	
PASSO	AÇÃO DE DESVIO
<passo desviado>	<condição do desvio> : <descrição da ação>
VARIAÇÃO EM TECNOLOGIA E DADOS	
PASSO	AÇÃO DE DESVIO
<passo onde pode ocorrer variação>	<lista de variações>
INFORMAÇÃO RELACIONADA (OPCIONAL)	
PRIORIDADE	<quão crítico é o caso de uso para seu sistema/organização>
DESEMPENHO ALVO	<o total de tempo que este caso de uso pode demorar>
FREQÜÊNCIA	<com que freqüência espera-se que o caso de uso ocorra>
CASO DE USO PAI	<opcional, nome do caso de uso que inclui este>
CASOS DE USO SUBORDINADOS	<opcional, ligações para sub-casos de uso>
ATOES SECUNDÁRIOS	<lista de outros atores necessários para realizar este caso de uso>

Figura 2.3 - - Modelo para especificação de caso de uso proposto por COCKBURN (2000)

Outro modelo para especificação de caso de uso muito conhecido é o proposto pelo RUP [RATIONAL, 2002]. Como pode ser visualizado na Figura 2.4, o diferencial desse modelo é a explícita referência a requisitos especiais (regras de negócio e requisitos não-funcionais) que afetam o caso de uso.

- 1 IDENTIFICAÇÃO DO CASO DE USO**

2 BREVE DESCRIÇÃO

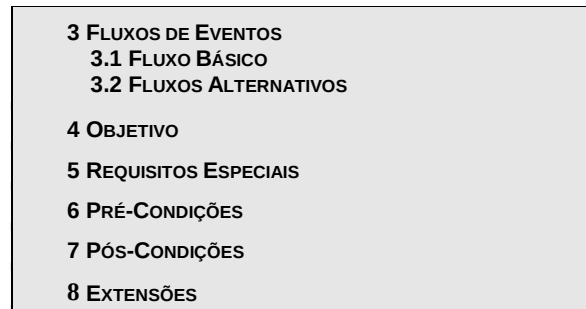


Figura 2.4 - Modelo para especificação de casos de uso proposto pelo RUP [RATIONAL, 2002]

2.2.3 Estruturar o modelo de casos de uso

O propósito dessa atividade é identificar comportamentos compartilhados, características comuns e relacionamentos entre casos de uso. Isso é feito não apenas para melhorar o reuso e entendimento dos casos de uso, mas também no intuito de preparar o modelo de casos de uso para transição realizada nos modelos seguintes.

Nessa atividade o analista pode utilizar os seguintes mecanismos para estruturar casos de uso: generalização, inclusão e extensão. Na UML [BOOCH et al., 1999], esses mecanismos são modelados respectivamente com os seguintes estereótipos de relacionamento entre casos de uso:

- `<<generalization>>`: é usado quando dois ou mais casos de uso têm algum comportamento, estrutura e propósito comum. Nesse caso, a parte comum é descrita num novo caso de uso, freqüentemente abstrato, denominado de caso de uso “pai”. Por sua vez, as variações em estrutura e comportamento são descritas em casos de uso “filhos” que especializam o caso de uso pai. O relacionamento de generalização também pode ser utilizado entre atores;
- `<<include>>`: representa a situação em que deseja-se separar uma parte do fluxo de um caso de uso num outro caso de uso à parte, denominado de caso de uso de inclusão. Em algum ponto do caso de uso base haverá uma chamada explícita a execução do caso de uso de inclusão. Em geral, esse relacionamento é usado em duas situações: (i) quando o fluxo do caso de uso de inclusão é complexo e apenas seu resultado é importante para o caso de uso base; ou (ii) para possibilitar o reuso do caso de uso de inclusão por outros casos de uso;

- <<extend>>: modela uma variação no comportamento normal do caso de uso. Esse comportamento “extra” é separado num caso de uso denominado de extensão e é inserido implicitamente no caso de uso base. A extensão é implicitamente inserida visto que sua execução não é mencionada diretamente no caso de uso base, pois o caso de uso base não precisa ter conhecimento se a interrupção provida pela extensão ocorrerá ou não. Esse tipo de relacionamento é geralmente usado para modelar as seguintes situações [JACOBSON et al., 1992]: (i) partes opcionais do sistema; (ii) fluxos alternativos e complexos que raramente acontecem; (iii) fluxos que são executados somente em certos casos. O relacionamento do tipo extensão precisa das seguintes informações: (a) a condição da extensão (no caso da extensão ser condicional); e (b) uma referência ao ponto de extensão no caso de uso base, ou seja, a posição no caso de uso onde a extensão ocorrerá. No método OOSE [JACOBSON et al., 1992], os pontos de extensão são referenciados somente no caso de uso de extensão; por outro lado, a linguagem UML [BOOCH et al., 1999] sugere que os possíveis pontos de extensão sejam identificados no caso de uso base através de rótulos (exemplo na Tabela 4.6) e que o caso de uso de extensão utilize esses rótulos para referenciar onde a extensão ocorrerá (exemplo na Tabela 4.7).

As notações em UML para representação dos mecanismos de estruturação de casos de uso podem ser visualizadas na Figura 2.5.

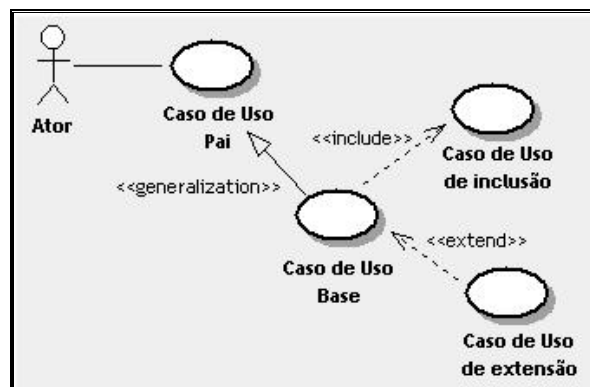


Figura 2.5 - Notações UML para mecanismos de estruturação de casos de uso

2.2.3.1 DIFERENÇAS NO USO DOS RELACIONAMENTOS DE INCLUDE, EXTEND E GENERALIZATION

Os mecanismos de generalização, inclusão, e extensão têm algumas características em comum. Por isso, freqüentemente ocorrem dúvidas sobre que tipo de relacionamento utilizar. Visando auxiliar no processo de estruturação de casos de uso, a seguir, discutimos algumas diferenças sutis entre esses mecanismos.

INCLUDE E EXTEND

Tanto na situação de inclusão de um caso de uso, como na de extensão, haverá uma interrupção no fluxo descrito no caso de uso base para a execução do caso de uso da inclusão/extensão; ao final da execução desse caso de uso, o controle será retornado para o fluxo do caso de uso base. Esse comportamento semelhante gera confusão ao escolher um dos dois mecanismos. Porém, a maior diferença entre esses mecanismos é o modo como o caso de uso é estruturado para indicar a interrupção na seqüência de seu fluxo de eventos [JACOBSON, 2003a]:

- no caso da inclusão, o fluxo do caso de uso base depende do resultado do caso de uso de inclusão para realizar seu objetivo, por isso existe uma referência explícita no caso de uso base à inserção do fluxo de eventos do caso de uso incluído;
- no caso da extensão, o fluxo do caso de uso base não especifica a interrupção, pois ele não depende do caso de uso de extensão para realizar seu objetivo. O caso de uso de extensão representará apenas uma variação ou situação atípica no fluxo do caso de uso base.

Para exemplificar a diferença no uso dos relacionamentos do tipo *include* e *extend*, vamos analisar o significado da utilização de cada um deles na seguinte situação: num sistema acadêmico de uma instituição de ensino particular, algumas operações são autorizadas somente quando o aluno está com as mensalidades em dia, por isso, deve ser feita a verificação de débito em casos de uso como *Emitir Histórico* e *Fazer matrícula em disciplina*.

Se modelarmos os casos de usos *Emitir Histórico* e *Fazer matrícula em disciplina* com uma relação do tipo *include* com o caso de uso *Verificar débito*, em algum ponto da especificação desses dois casos de uso, haverá uma chamada **explícita** ao caso de uso *Verificar débito*. Sendo assim, os casos de uso *Emitir Histórico* e *Fazer matrícula em disciplina* ficam “amarrados” ao caso de uso *Verificar Débito*. Por isso,

esses casos de uso só poderiam ser completamente reusados em sistemas que possuíssem essa mesma “amarração”. Por isso, o construtor *include* deve ser usado preferencialmente nos casos onde o caso de uso incluído é intrínseco ao caso de uso base, ou seja, quando a execução do caso de uso incluído é uma parte essencial e inseparável do caso de uso base. Um exemplo típico dessa situação é o relacionamento que existe entre os casos de uso *Pagar mensalidade atrasada* e *Verificar débito*.

GENERALIZATION E INCLUDE

Tanto o relacionamento de generalização quanto o de inclusão podem ser usados para que um caso de uso reuse o comportamento de outro. A diferença é que na generalização, a execução dos casos de uso “filhos” é dependente da estrutura e comportamento do caso de uso “pai”, enquanto num relacionamento de inclusão a execução do caso de uso base depende apenas do resultado obtido pela execução do caso de uso de inclusão. Outra diferença é que numa generalização os casos de uso filhos compartilham semelhanças em propósito e estrutura, enquanto num relacionamento do tipo *include* os casos de uso que estão reusando o mesmo caso de uso de inclusão podem ter propósitos completamente diferentes, mas precisam da função fornecida pelo caso de uso de inclusão para serem executados [RATIONAL, 2002].

A Figura 2.6 ilustra um exemplo para esclarecer as diferenças entre esses dois relacionamentos. Os caso de uso *Comprar produtos pela Internet* e *Comprar produtos por telefone* têm propósitos similares e suas semelhanças são generalizadas no caso de uso *Comprar produtos*. Sendo assim, qualquer alteração no comportamento do caso de uso *Comprar produtos* causará alteração no fluxo de execução dos casos de uso *Comprar produtos pela Internet* e *Comprar produtos por telefone*. Por sua vez, os casos de uso *Comprar produtos* e *Reservar produtos* possuem propósitos distintos e ambos reusam o caso de uso *Buscar informações de um produto* em seu fluxo de execução através de um relacionamento do tipo *include*. Nesta situação, se o caso de uso *Buscar informações de um produto* modificar seu método de busca, os casos de uso que o incluem não serão alterados, pois eles dependem apenas do resultado obtido pela execução do caso de uso *Buscar informações de um produto*.

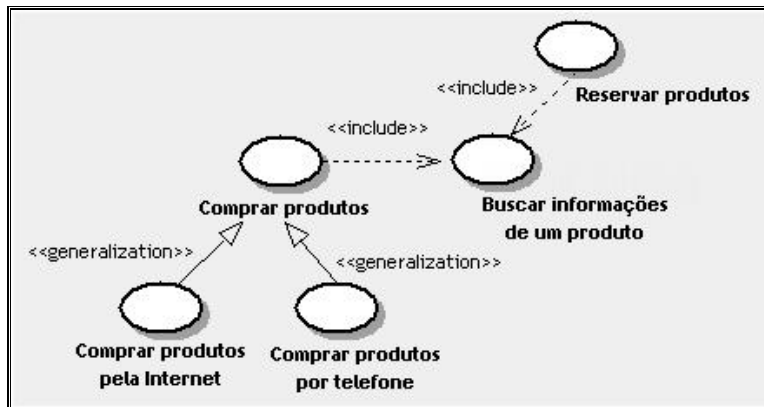


Figura 2.6 - Diagrama de caso de uso ilustrando diferença entre os relacionamentos de generalização e inclusão

2.3 Fluxo de análise

O fluxo de análise tem como objetivo fornecer uma visão conceitual, independente de implementação, de como o sistema irá se comportar internamente para satisfazer os requisitos. No fluxo de requisitos, os detalhes sobre o que o sistema faz em resposta as ações de um ator são superficialmente descritos. Por outro lado, o fluxo de análise foca em descrever o comportamento interno do sistema para concretizar os casos de uso utilizando para isso um tipo de abstração chamado de classes de análise.

Classes de análise representam um modelo conceitual inicial para entidades no sistema que possuem responsabilidades e comportamento. Existem três tipos de classes de análise: fronteira, entidade e controle. Cada uma tem seu próprio propósito, modelando um papel específico de um componente do sistema. Essas classes eventualmente evoluem em classes e subsistemas no modelo de projeto.

A Figura 2.7 exhibe o fluxo das principais atividades que devem ser executadas para produzir o modelo de análise, destacado aquelas que serão focadas nesse trabalho. Este fluxo é iniciado pelos arquitetos, que agrupam casos de uso inter-relacionados em pacotes. A seguir, os engenheiros analisam cada caso de uso e os decompõem em termos de classes de análise responsáveis por concretizar seu comportamento. Na seqüência, as responsabilidades, os atributos e os relacionamentos de cada classe de análise são especificados. Por fim, pacotes de análise identificados inicialmente pelos arquitetos são refinados e mantidos.

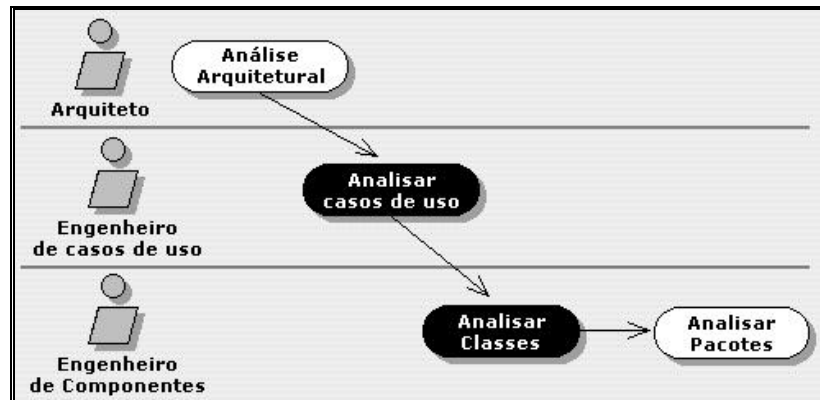


Figura 2.7 – Atividades e seus responsáveis no fluxo de análise (reproduzido de [JACOBSON et al., 1999])

A seguir, detalhamos as atividades do fluxo de análise que serão consideradas nesse trabalho.

2.3.1 Analisar casos de uso

O objetivo dessa atividade é analisar cada caso de uso identificado no fluxo de requisitos no intuito de: (i) identificar classes de análise que encapsulam o comportamento e atributos necessários para executar o fluxo de eventos do caso de uso; e (ii) descrever como os objetos de análise dessas classes interagem para realizar o caso de uso.

2.3.1.1 IDENTIFICAR CLASSES DE ANÁLISE

O propósito dessa atividade é distribuir o comportamento de cada caso de uso entre classes de análise. Uma classe de análise deve se encaixar em um desses três tipos: fronteira, entidade ou controle, representados em UML respectivamente pelos seguintes estereótipos de classe: <<boundary>>, <<entity>> e <<control>>. A Figura 2.8 ilustra o particionamento da funcionalidade de um caso de uso entre classes de análise e os símbolos usados para representar cada um dos três tipos de classe de análise.

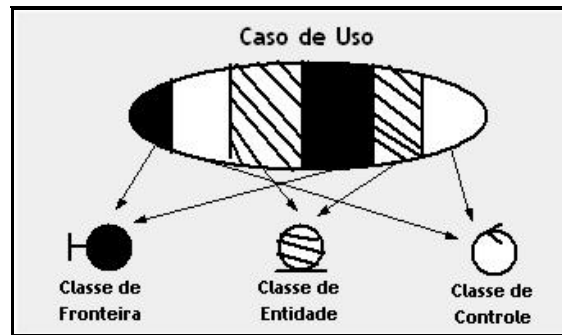


Figura 2.8- Representação do particionamento da funcionalidade de um caso de uso entre classes de análise (adaptado de [JACOBSON et al., 1992])

A razão em classificar classes de análise em três tipos é conseguir uma estrutura que será mais estável a mudanças [JACOBSON et al., 1992]. A estabilidade ocorrerá no sentido de mudanças serem localizadas, afetando (preferencialmente) apenas um tipo de classe no sistema. Então, sabendo que os tipos mais comuns de mudança estão relacionadas à interface e à funcionalidade do sistema, mudanças na interface afetarão apenas classes do tipo fronteira; mudanças na funcionalidade afetarão classes de entidade ou classes de controle, dependendo se a mudança está relacionada à representação da informação armazenada ou à lógica de controle ou interação entre objetos, respectivamente. Portanto, a razão em escolher esses três tipos de classes é conseguir obter localização nas mudanças do sistema.

Uma classe do tipo fronteira modela a parte do caso de uso responsável pela comunicação entre os atores e o sistema. A tarefa da classe de fronteira é traduzir as entradas do ator em eventos no sistema, e traduzir eventos do sistema em algo que é apresentado ao ator. Em outras palavras, classes de fronteira descrevem a comunicação bidirecional entre o sistema e seus usuários. Geralmente, classes de fronteira estão relacionadas à interface com o usuário e a protocolos de comunicação. Existem algumas estratégias para identificar esse tipo de classe:

- analisar as descrições de interface do sistema obtidas no modelo de requisitos;
- analisar as interfaces necessárias para cada ator se comunicar com o sistema;
- e

- ler as descrições de casos de uso e extrair a funcionalidade que é específica da interface, tais como: apresentação de informações para o ator, obtenção de dados do ator, etc.

Uma classe do tipo entidade representa informações no sistema que devem ser armazenadas e mantidas mesmo após a finalização do caso de uso. Todo comportamento naturalmente associado à manipulação ou atualização dessas informações também deve estar contido na classe do tipo entidade. Esse tipo de classe, geralmente, corresponde a algum conceito do domínio do problema, é passiva e persistente e tem como principal responsabilidade armazenar e gerenciar informação no sistema.

Por fim, classes do tipo controle são responsáveis por coordenar as atividades de outras classes que implementam a funcionalidade de um caso de uso, ou seja, são responsáveis por delegar trabalho para outras classes e definir a ordem entre os eventos do caso de uso. Além disso, classes de controle podem encapsular lógica de negócio que envolve vários objetos. No entanto, na situação onde um caso de uso envolve apenas a simples manipulação de informação armazenada, o caso de uso pode ser concretizado usando apenas classes de interface e entidade, sem utilizar classes de controle. Exemplos típicos de funcionalidade colocadas em classes de controle são: controle de transações, gerenciamento de tarefas e de recursos e manipuladores de exceção.

O processo de identificação de classes de análise é iterativo, por isso, sempre que possível, é importante reusar classes de análise identificadas em casos de uso anteriores. Então, como uma classe pode ser comum a diferentes casos de uso, é importante declarar explicitamente qual é responsabilidade de cada classe nos casos de uso.

2.3.1.2 DESCREVER INTERAÇÕES ENTRE OBJETOS DE ANÁLISE

Quando se tem uma visão geral das classes de análise necessárias para realizar um caso de uso, é necessário descrever como os objetos de análise dessas classes interagem para realizar esse caso de uso. No fluxo de análise, essa interação geralmente é representada utilizando diagramas de colaboração [BOOCH et al., 1999], os quais

contém os atores, os objetos de análise e as interações entre eles, responsáveis por concretizar o caso de uso.

Um diagrama de colaboração é criado percorrendo-se cada passo do fluxo de eventos do caso de uso, desde seu início, e decidindo que iterações entre objetos de análise e/ou instância de atores são necessárias para realizar aquele passo. Pelo menos um diagrama de interação é necessário para o fluxo principal de eventos, acrescido de pelo menos um diagrama para cada fluxo alternativo/excepcional.

2.3.2 Analisar classes

Nessa atividade as responsabilidades, os atributos e os relacionamentos de cada classe de análise são identificados e mantidos.

As responsabilidades de uma classe são derivadas a partir das mensagens que essa classe recebe nos diagramas de colaboração nos quais ela está presente. Geralmente, os atributos são identificados quando, para realizar uma responsabilidade, a classe de análise deve ter alguma informação armazenada. Já os relacionamentos entre classes são identificados a partir das ligações entre objetos nos diagramas de colaboração. Na análise, são criados relacionamentos entre classes para representar alguma mensagem que deve ser trocada entre objetos de duas classes. No fluxo de projeto do sistema, essas ligações podem ser concretizadas de várias maneiras: determinando que o objeto da classe que recebe a mensagem terá escopo global; ou que esse objeto será passado como parâmetro; ou ainda que esse objeto tem uma associação permanente com o objeto que lhe envia as mensagens (agregação); etc.

2.4 Fluxo de projeto

O fluxo de projeto tem o objetivo de refinar o modelo de análise de acordo com o ambiente de implementação e mecanismos de projeto que serão adotados.

A Figura 2.9 exibe o fluxo das principais atividades que devem ser executadas para produzir o modelo de projeto, destacado aquelas que serão focadas nesse trabalho. Inicialmente os arquitetos definem os nós físicos do sistema, os subsistemas e suas interfaces e mecanismos de projeto que serão utilizados. Em seguida, os engenheiros

definem como os casos de uso serão concretizados em termos de interações entre classes de projeto e/ou subsistemas. O passo seguinte é identificar operações, atributos e relacionamentos para cada classe de projeto e subsistema. Através desse fluxo de atividades, os desenvolvedores identificarão candidatos a novos subsistemas, interfaces, classes e mecanismos de projeto, exigindo que os engenheiros de componentes refinem e mantenham os subsistemas previamente definidos.

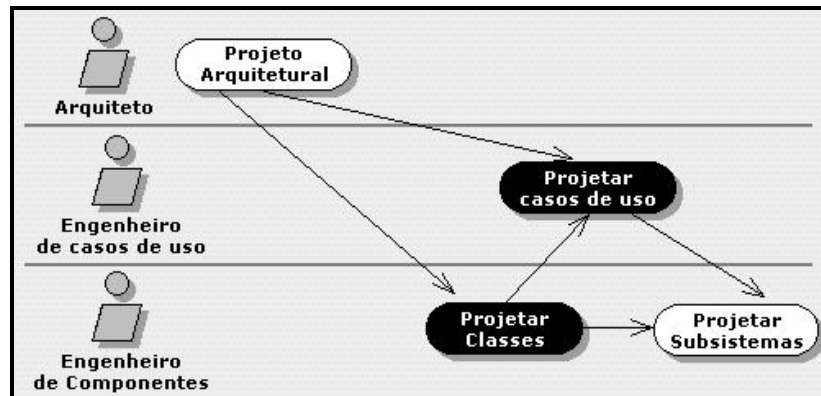


Figura 2.9 - Atividades e seus responsáveis no fluxo de projeto (reproduzido de [JACOBSON et al., 1999])

Como pode ser visualizado na Figura 2.9, as atividades do fluxo de projeto são semelhantes às atividades do fluxo de análise. A diferença está na semântica dessas atividades: enquanto o fluxo de análise preocupa-se em desenvolver um esboço conceitual do sistema, o fluxo de projeto estabelece uma abstração de como o sistema realmente será implementado, levando em consideração questões relacionadas a linguagens de programação, a sistemas operacionais, a tecnologias de distribuição, concorrência, gerenciamento de transações, persistência, etc. Portanto, enquanto as entidades manipuladas no fluxo de análise são meramente conceituais, as entidades manipuladas no fluxo de projeto têm um mapeamento direto com as entidades manipuladas na implementação.

A seguir, detalhamos as atividades do fluxo de projeto que serão consideradas nesse trabalho.

2.4.1 Projetar casos de uso

Os principais passos na atividade de projetar um caso de uso são:

- identificar as classes de projeto e/ou subsistemas cujas instâncias são necessárias para executar o fluxo de eventos de um caso de uso;
- descrever o comportamento do caso de uso em função da interação entre objetos de projeto e/ou subsistemas;

Cada um desses passos será detalhado a seguir.

2.4.1.1 IDENTIFICAR E ESPECIFICAR ENTIDADES DE PROJETO

Classes de análise encapsulam responsabilidades necessárias para concretização de um caso de uso, independentes da tecnologia e do ambiente de implementação adotado. Essas responsabilidades podem ser realizadas por um ou mais elementos de projeto (entidades dependentes da tecnologia adotada). Além do mais, dependendo da tecnologia que o elemento de projeto representa, ele pode encapsular responsabilidades de várias classes de análise.

Inicialmente, para cada classe de análise dada como entrada para essa atividade, é criada uma ou várias classes de projeto. Para cada tipo de classe de análise (fronteira, entidade e controle), existem estratégias específicas que podem ser usadas para criar um conjunto inicial de entidades de projeto (classes, subsistemas e interfaces) (detalhes em [RATIONAL, 2002]). A derivação de classes de projeto a partir de classes de análise também deve considerar a estrutura de camadas da aplicação definida no projeto da arquitetura e questões como distribuição, persistência, controle de transações, etc. É importante manter o rastreamento entre uma entidade de projeto e a(s) classe(s) de análise que a gerou.

À medida que as entidades de projeto vão sendo especificadas, será possível refinar, ajustar, dividir ou fundir as classes inicialmente identificadas.

2.4.1.2 DESCREVER INTERAÇÕES ENTRE OBJETOS DE PROJETO

Quando já se tem uma visão geral das classes de projeto necessárias para realizar um caso de uso, é necessário descrever como os seus objetos de projeto interagem. Enquanto na análise a interação entre os objetos é preferencialmente modelada usando diagramas de colaboração, nessa atividade, essas interações são

modeladas principalmente utilizando diagramas de seqüência [BOOCH et al., 1999] contendo instâncias dos atores, objetos de projeto e mensagens transmitidas entre eles.

Os diagramas de colaboração construídos na atividade de análise podem ser utilizados como ponto de partida na construção dos diagramas de seqüência, fornecendo uma visão macro de como os objetos de projeto irão interagir. Enquanto as mensagens trocadas entre objetos num diagrama de colaboração denotam responsabilidades, as mensagens trocadas entre objetos num diagrama de seqüência denotam operações.

Como geralmente nas atividades de projeto muitos objetos são adicionados, pode-se simplificar os diagramas de seqüência, encapsulando sub-fluxos de eventos dentro de subsistemas. Fazendo isso, largas subseções de um diagrama de seqüência são substituídas por uma única mensagem para o subsistema. Dentro do subsistema, um diagrama de seqüência à parte pode ilustrar as interações internas que provêem o comportamento fornecido pelo subsistema. A Figura 2.10 mostra um exemplo dessa simplificação: as linhas de vida no diagrama central representam subsistemas; as interações nos círculos representam as interações internas dos membros do subsistema em resposta a mensagem.

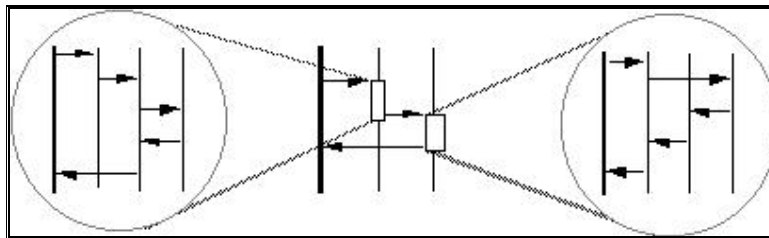


Figura 2.10 – Exemplo de substituição de subfluxos num diagrama de seqüência por subsistemas

Se um caso de uso possui diferentes cenários, é geralmente conveniente criar um diagrama de seqüência para cada um deles.

2.4.2 Projetar classes

A seguir, as responsabilidades de cada classe de análise são distribuídas entre a(s) entidade(s) de projeto que representa(m) a classe de análise, criando uma operação na entidade para cada responsabilidade. Os diagramas de interação gerados podem auxiliar na especificação das operações, seus parâmetros e tipos de retorno. Também devem ser identificados atributos para cada entidade de projeto.

A comunicação entre objetos de projeto implica que existe algum tipo de relacionamento entre classes: de associação, de agregação ou de generalização. Esses relacionamentos são modelados em diagramas de classe.

Associações mostram que objetos de uma classe precisam ter referências a objetos de outras classes para trocar mensagens. Uma agregação é uma forma especial de associação que modela um relacionamento entre uma classe e suas partes agregadas. Por sua vez, a generalização modela relacionamentos entre instâncias de classes mais específicas e instâncias de classes mais genéricas.

2.5 Tratamento de requisitos não-funcionais

Como casos de uso representam situações de utilização do sistema, eles são bons para especificar requisitos funcionais. Porém, não são adequados para manipular os requisitos não-funcionais de um sistema.

O Processo de Desenvolvimento Unificado não detalha nenhum tratamento sistemático para requisitos não-funcionais em nenhum dos seus fluxos. No fluxo de requisitos apenas menciona como eles devem ser especificados:

- requisitos não-funcionais que referem-se ao sistema como um todo: esse tipo de requisito não pode ser associado a um caso de uso particular, por isso, deve ser especificado numa seção especial do documento de requisitos, denominada de “Requisitos adicionais” (*Supplementary requirements*) [JACOBSON et al., 1999]. Exemplos de requisitos nessa categoria podem ser: “o sistema deve ser portátil entre as plataformas Microsoft Windows e Unix” e “o sistema deve utilizar o SGBD Oracle 8.0”;
- requisitos não-funcionais que se aplicam a casos de uso específicos: esse tipo de requisito geralmente representa qualidades adicionais que o sistema deve suportar ao fornecer a funcionalidade descrita no caso de uso e devem ser especificados numa parte da descrição do caso de uso específica para esse tipo de requisito (item 5 na Figura 2.4). Um exemplo de requisitos nessa categoria é o seguinte: “as transações devem ser completadas em menos de dois segundos”.

Caso um requisito não-funcional implique em alguma restrição comportamental que altere o fluxo de eventos de um ou mais casos de uso, essa restrição deve ser inserida diretamente no fluxo de eventos.

Já no fluxo de análise e projeto, o Processo Unificado indica pontos nas atividades desses fluxos onde requisitos não-funcionais devem ser identificados e especificados, mas não entra em detalhes em como esse processo deverá ser conduzido.

2.6 Considerações finais

Nos últimos anos, o Processo Unificado [JACOBSON et al., 1999] tem se tornado um padrão industrial para o desenvolvimento de software orientado a objetos [AMBLER, 2001]. O estilo de desenvolvimento guiado por casos de uso é uma das práticas mais bem aceitas e difundidas desse processo.

Contudo, o foco em casos de uso tem feito com que métodos que adotam esse estilo de desenvolvimento dêem pouca atenção a preocupações não-funcionais. O Processo Unificado apesar de incluir atividades voltadas para essa preocupação, praticamente só as menciona, não oferecendo diretrizes nem métodos que indiquem como preocupações não-funcionais devem ser tratadas de maneira sistemática.

Além disso, outro aspecto que pode ser melhorado no Processo Unificado é a separação de preocupações nos artefatos de software, especialmente no que se refere à separação de preocupações não-funcionais e regras de negócio que alteram o fluxo de execução dos casos de uso. Geralmente, essas preocupações ficam espalhadas e misturadas com preocupações próprias dos artefatos nos quais elas precisam se aplicadas.

Conforme veremos no capítulo 4, este trabalho oferece contribuições para as atividades da perspectiva guiada por casos de uso do Processo Unificado nesses dois sentidos: (i) propiciando o tratamento sistemático de preocupações não-funcionais no fluxo de requisitos através do *framework* NFR [MYLOPOULOS et al., 1992; CHUNG et al., 2000; MYLOPOULOS et al., 2001]; e (ii) dando suporte a uma melhor separação de preocupações nos artefatos de software a partir dos fundamentos do paradigma orientado a aspectos.

DESENVOLVIMENTO DE SOFTWARE ORIENTADO A ASPECTOS: ESTADO DA ARTE

We think of the current state of Aspect Oriented Programming research as analogous to that of Object Oriented Programming 20 years ago. The basic concepts are beginning to take form, and an expanding group of researchers are using them in their work.

Gregor Kiczales (1997)

A adoção de um novo paradigma de desenvolvimento de software freqüentemente progride a partir de práticas estabelecidas na programação, as quais vão sendo posteriormente incorporadas nas atividades de projeto, análise e requisitos. Assim como já ocorreu com os paradigmas estruturado e orientado a objetos, esse está sendo o curso trilhado pelo paradigma emergente de orientação a aspectos.

Nesse capítulo, apresentamos uma visão geral do estado da arte do desenvolvimento de software orientado a aspectos. Inicialmente, descrevemos o princípio da separação de preocupações, principal alicerce do paradigma orientado a aspectos. Em seguida, detalhamos as motivações para o surgimento desse paradigma de desenvolvimento, descrevendo os problemas que ele se propõe a solucionar e algumas das abordagens que o fundamentaram. Na sequência, apresentamos uma coletânea dos fundamentos comumente atribuídos ao paradigma orientado a aspectos. Por fim, fazemos uma revisão bibliográfica sobre propostas para dar suporte a esse paradigma nas atividades de requisitos e projeto, e apresentamos nossas considerações finais.

3.1 Separação de preocupações

Produzir software com qualidade e num menor tempo possível é um dos principais objetivos da Engenharia de Software [PRESSMAN, 2001]. Para alcançar tal objetivo, essa área de pesquisa estabeleceu alguns princípios que devem ser aplicados ao longo do processo de desenvolvimento e nos artefatos de software: **separação de preocupações**, modularidade, rigor e formalidade, incrementalidade, abstração, generalidade e antecipação de mudanças [GHEZZI et al., 1991].

O principal propósito do princípio da separação de preocupações é reduzir o raciocínio para uma quantidade factível [DIJKSTRA, 1976]. Esse princípio estabelece que um problema deve ser decomposto em unidades menores, claramente separadas, de maneira que cada uma delas represente uma única preocupação [AKSIT et al., 2001]. A idéia básica é manipular com uma preocupação do sistema de cada vez. Como consequência, a aplicação desse princípio possibilita [GHEZZI et al., 1991]:

- diminuir a complexidade do desenvolvimento de software, concentrando-se em diferentes preocupações separadamente;
- raciocinar sobre diferentes preocupações de maneira relativamente independente;
- facilitar a inserção/remoção de preocupações na aplicação;
- dividir esforços e separar responsabilidades entre membros da equipe de desenvolvimento; e
- melhorar a modularidade de artefatos de software.

Além disso, quando os artefatos de software possuem a propriedade de separação de preocupações, as preocupações de cada artefato tendem a ser similares (forte coesão) e a dependência entre os artefatos é minimizada (fraco acoplamento). Conseqüentemente, mudanças em artefatos relacionadas a uma preocupação, tendem a ter um efeito limitado — ou até mesmo nenhum efeito — em artefatos relacionados a outras preocupações [HURSCH & LOPES, 1995]. Isso facilita a compreensão, evolução, adaptação, customização e reuso dos artefatos de software.

No entanto, para incorporar o princípio da separação de preocupações no processo de desenvolvimento e nos artefatos gerados, o engenheiro de software precisa utilizar métodos apropriados [GHEZZI et al., 1991]. Esses métodos são geralmente baseados no paradigma de programação adotado. Em métodos orientados a objetos, por exemplo, as preocupações são separadas em objetos e classes; em métodos estruturados, elas são separadas em módulos ou subsistemas.

De maneira geral, os métodos para separação de preocupações envolvem as seguintes atividades [AKSIT et al., 2001]:

- identificação de preocupações: seleção de preocupações que o sistema vai incluir;
- representação de preocupações separadamente: especificação de preocupações em unidades ou num conjunto de unidades que concretizem a realização de cada uma das preocupações; e
- composição de preocupações: integração de unidades separadas a fim de dar a elas alguma coerência para formar o conjunto do sistema.

Uma boa prática a ser considerada na representação de preocupações separadamente, é a “Lei de Demeter” [LIEBERHERR & HOLLAND, 1989], a qual determina que unidades só devem ter conhecimento de outras unidades que compartilham da mesma preocupação. O paradigma orientado a objetos provê bons mecanismos para modelar o relacionamento entre unidades que representam preocupações semelhantes, tais como herança, agregação e troca de mensagens. Contudo, ele possui limitações para modelar o relacionamento entre unidades que têm preocupações distintas, especialmente aquelas com um escopo abrangente no sistema [ELRAD et al., 2001a]. Para tentar solucionar essas limitações, foram propostas algumas abordagens na literatura, como veremos na seção a seguir. Essas propostas pertencem a uma área de pesquisa denominada de Separação Avançada de Preocupações (*Advanced Separation of Concerns* ou ASoC).

3.2 Abordagens para separação avançada de preocupações

Atualmente, o paradigma de desenvolvimento de software dominante é o orientado a objetos [AMBLER, 2001]. Entretanto, apesar desse paradigma ser conveniente para separação da maior parte das preocupações relacionadas à funcionalidade básica do sistema, ele tem dificuldades em separar preocupações relacionadas à restrições globais do software e propriedades sistêmicas, tais como: segurança, performance, confiabilidade, persistência, distribuição, tratamento de erro e exceções, etc [KICZALES et al., 1997]. O tratamento dessas preocupações não consegue ser claramente capturado em uma das abstrações disponíveis e, portanto, fica espalhado por várias abstrações e/ou misturado com as preocupações específicas dessas abstrações [FENG et al., 2001]. Por apresentar tal comportamento, esse tipo de preocupação é denominado de **preocupação transversal**.

A Figura 3.1 ilustra um exemplo que demonstra o comportamento típico de preocupações transversais na hierarquia de classes. A preocupação relativa à política de segurança deve ser tratada em várias classes e, como não existe nenhum mecanismo que possibilite localizar essa preocupação numa única abstração, o tratamento da política de segurança fica espalhado entre várias classes. Além disso, a preocupação de segurança fica misturada com as preocupações específicas de cada classe.

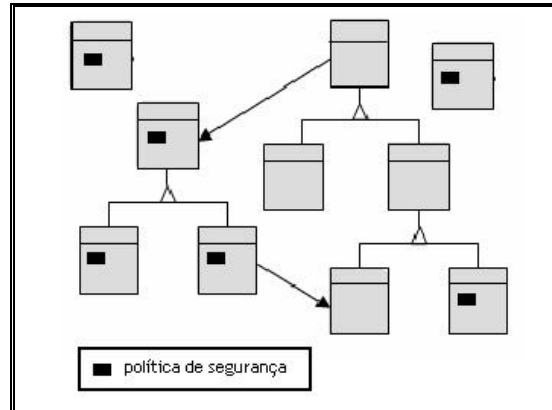


Figura 3.1 – Exemplo de espalhamento e entrelaçamento de preocupação transversal na hierarquia de classes

Nos últimos anos, algumas abordagens foram propostas a fim de complementar o paradigma orientado a objetos, fornecendo mecanismos de projeto e implementação para uma melhor separação de propriedades transversais. Entre elas, podemos citar: Programação Adaptativa [LIEBERHERR, 1996], Filtros de Composição [BERGMANS & AKSIT, 2001; BERGMANS et al., 2001], Separação Multidimensional de Preocupações [TARR et al., 1999; OSSHER & TARR, 2000] e Programação Orientada a Aspectos [KICZALES et al., 1997]. Contudo, a Programação Orientada a Aspectos tem sido a que tem sobressaído mais ultimamente [VIEGA & VUAS, 2000].

A seguir, apresentamos uma visão geral de três das mais difundidas dessas abordagens (Programação Orientada a Aspectos, Separação Multidimensional de Preocupações e Filtros de Composição), mostrando suas diferenças em abstrações, terminologias, e mecanismos usados para separar e compor preocupações transversais.

3.2.1 Programação Orientada a Aspectos

O termo “programação orientada a aspectos” é atribuído ao grupo de pesquisa da Xerox PARC liderado por Gregor Kiczales e corresponde a uma técnica de programação que possibilita a separação de preocupações transversais na implementação de sistemas [KICZALES et al., 1997].

Essa abordagem denomina de maneira distinta dois tipos de preocupações: as que conseguem ser adequadamente modularizadas pelos paradigmas estruturado ou orientado a objetos são chamadas de **componentes**; e aquelas que não conseguem, ou

seja, as preocupações transversais, são denominadas de **aspectos** [KICZALES et al., 1997].

Na Programação Orientada a Aspectos, assume-se que existe uma decomposição dominante do sistema (ex: em classes ou módulos) para os componentes, e que os aspectos representam outra unidade de decomposição do sistema. Dessa maneira, componentes são implementados numa linguagem procedural ou orientada a objetos, enquanto aspectos são codificados em uma linguagem especialmente designada para esse propósito. Linguagens orientadas a aspectos utilizam cinco elementos para permitir a modularização de preocupações transversais [ELRAD et al., 2001b]:

- (i) um modelo de pontos de junção descrevendo os possíveis pontos onde o comportamento do aspecto pode ser adicionado;
- (ii) um meio de identificar esses pontos de junção;
- (iii) um meio de especificar o comportamento adicional nesses pontos;
- (iv) unidades que encapsulam a especificação dos pontos de junção e as melhorias comportamentais providas pelo aspecto; e
- (v) um mecanismo para combinar o comportamento transversal do aspecto com o comportamento do(s) componente(s) que ele afeta.

Em resumo, na Programação Orientada a Aspectos, preocupações transversais são encapsuladas em unidades denominadas aspectos, nas quais são especificados os pontos de execução nos componentes que a preocupação transversal afetará e o comportamento/propriedade que devem ser adicionados nesses pontos. Como ilustrado pela Figura 3.2, o sistema final é formado a partir da combinação de aspectos e componentes, num processo denominado de *weaving*.

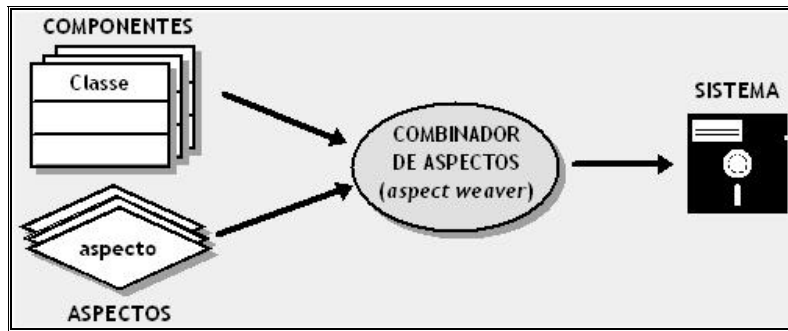


Figura 3.2 - Processo de combinação de aspectos e classes para gerar o sistema

3.2.1.1 ASPECTJ

A linguagem AspectJ [KICZALES et al., 2001] é um superconjunto da linguagem Java [GOSLING et al., 2000] para ser usado na Programação Orientada a Aspectos. Em uma aplicação orientada a aspectos em AspectJ, os componentes são implementados usando a sintaxe padrão de Java, e os aspectos são implementados usando uma sintaxe específica de AspectJ. Essa linguagem utiliza um processador de aspectos — denominado de *weaver* — para combinar o código do aspecto com o código dos componentes e conseqüentemente produzir o sistema executável. Após a compilação e o processo de *weaving*, o código objeto gerado pode ser executado em qualquer máquina virtual Java.

Em AspectJ, o código referente uma preocupação transversal é especificado numa unidade de codificação denominada *aspect* (veja Figura 3.3). Dentro da unidade do tipo *aspect*, o código relacionado à preocupação transversal utiliza construtores lingüísticos especiais:

- o construtor *pointcut* é utilizado para capturar ou identificar pontos de junção no fluxo do programa sobre os quais o aspecto atua. Exemplos de possíveis pontos de junção incluem inicialização de objetos, chamadas de métodos e acesso a campos;
- *advice*s são blocos de código no quais devem ser definidos as regras de composição e o comportamento a ser executado pelo aspecto. Para determinar as regras de composição nos pontos de junção identificados, utilizam-se as regras de composição *before*, *after* e *around*. Essas regras indicam que o comportamento do aspecto deve ser executado,

respectivamente, antes, após, ou ao invés de cada ponto de junção definido pelo *pointcut* associado.

- *inter-type declarations* possibilita que classes presentes no sistema sejam modificadas estruturalmente: (i) acrescentando métodos, atributos e classes internas; e (ii) alterando a hierarquia de herança de classes e interfaces.

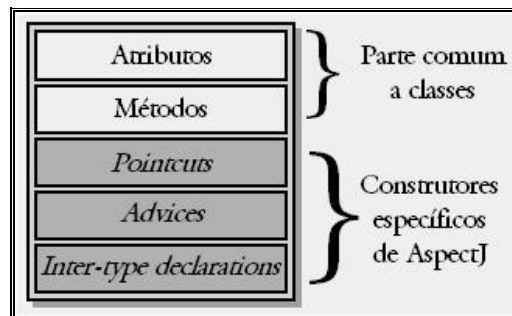


Figura 3.3 - Estrutura de uma entidade *aspect* em AspectJ

Conforme ilustra a Figura 3.3, além dos construtores próprios de AspectJ, uma unidade do tipo *aspect* pode conter membros de dados e métodos, assim como uma classe.

A Figura 3.4 apresenta um exemplo de código em AspectJ que implementa um aspecto de *logging* para rastrear a execução de métodos. A princípio, é definido um ponto de junção *loggedMethod* (linha 3), incluindo todas as chamadas de métodos. Entre a linha 5 e a 11 são definidos dois *advices* para que sejam exibidas mensagens imediatamente antes da execução e imediatamente após o retorno do ponto de junção.

```

1 public aspect Logging {
2
3     pointcut loggedMethod(): call(* *(..));
4
5     before(): loggedMethod() {
6         System.out.println("-- ANTES DA EXECUÇÃO DA CHAMADA DE UM MÉTODO -- ");
7     }
8
9     after(): loggedMethod() {
10        System.out.println("-- DEPOIS DO RETORNO DA CHAMADA DE UM MÉTODO -- ");
11    }
12 }

```

Figura 3.4 - Exemplo de código em AspectJ

Para permitir a implementação de experimentos com aspectos em sistemas operacionais e sistemas embutidos escritos em C/C++, foram desenvolvidas extensões orientadas a aspectos dessas linguagens baseadas num subconjunto de AspectJ: AspectC [COADY et al., 2001] e Aspect C/C++ [SPINCZYK et al., 2002]. Também existem propostas de extensão da linguagem C# [KIM, 2002] para incluir orientação a aspectos.

3.2.2 Separação Multidimensional de Preocupações

A abordagem de Separação Multidimensional de Preocupações é a evolução do trabalho anterior em *Subject-Oriented Programming* [HARRISON & OSSHER, 1993]. A principal motivação dessa abordagem é permitir que um sistema possa ser gerado a partir da composição de unidades espalhadas por diferentes visões ou dimensões de preocupações². Essa abordagem pretende combater a “tirania da decomposição dominante” [TARR et al., 1999] que predomina nas atuais abordagens para separação de preocupações, nas quais o sistema só pode ser modularizado de uma única maneira por vez e, caso as preocupações não se adequem a esse tipo de decomposição, elas acabam espalhadas por muitos módulos e entrelaçadas com preocupações que não são as suas.

Portanto, o objetivo da Separação Multidimensional de Preocupações (MDSOC) é permitir a separação, modularização e integração de preocupações distribuídas em várias dimensões. Para alcançar esse objetivo foi desenvolvida uma abordagem particular denominada hiper-espacos³ [TARR et al., 1999; OSSHER & TARR, 2000; OSSHER & TARR, 2001].

Um conceito importante na abordagem de hiper-espacos é o conceito de unidade. Uma unidade é um construtor sintático numa linguagem de descrição de artefatos (ex: uma classe, um caso de uso, um método, uma variável, um pacote, etc.) [OSSHER & TARR, 2000]. O papel da abordagem de hiper-espacos é organizar as unidades de um sistema, de maneira que seja possível: (i) separar todas as unidades correspondentes a uma mesma preocupação; (ii) descrever os vários tipos de inter-

² Uma dimensão de preocupação é uma maneira distinta de decompor o sistema.

³ tradução livre do inglês *hyperspaces*

relacionamentos entre unidades de diferentes preocupações (iii) indicar como os componentes de software e os sistemas podem ser construídos e integrados a partir de unidades que representam essas preocupações.

A abordagem de hiper-espacos pode ser usada para organizar e manipular com unidades escritas em qualquer linguagem, porém, o suporte de ferramentas é geralmente específico para determinada linguagem. Hyper/J (Seção 3.2.2.1), por exemplo, manipula somente unidades Java (ex: classe, pacote, método).

Como pode ser visualizado na Figura 3.5, na abordagem de hiper-espacos, unidades são organizadas em uma matriz multidimensional. Cada eixo representa uma dimensão de preocupação e, cada ponto num eixo representa uma preocupação naquela dimensão. Isso torna explícito todas as dimensões de interesse, as preocupações que pertencem a cada dimensão e quais unidades afetadas por determinada preocupação da dimensão.

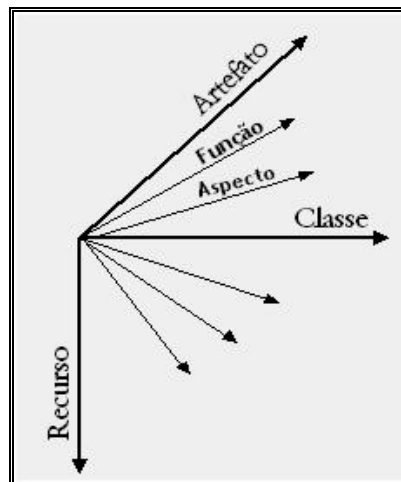


Figura 3.5 - Exemplo de possíveis dimensões de um hiper-espaço

Resumidamente, a proposta é possibilitar a representação de preocupações em diferentes dimensões, encapsular o conjunto de unidades pertencentes a uma mesma preocupação num módulo denominado de *hyperslice*⁴ e, por fim, compor os diversos *hyperslices* para formar o sistema. *Hyperslices* são compostos usando regras de composição específicas definidas em módulos denominadas de *hypermodules*. A composição é baseada na existência de conceitos comuns entre unidades de *hyperslices*

distintos, isto é, diferentes unidades descrevendo o mesmo conceito são compostas numa única unidade descrevendo aquele conceito mais completamente. Existem diferentes tipos de relacionamentos entre unidades de *hyperslices*. Alguns exemplos incluem:

- associação de declarações abstratas em um hyperslice a implementações de métodos em outro hyperslice. Por exemplo, associar um método abstrato num hyperslice à implementação desse método em outro hyperslice;
- correspondência (similaridade) de múltiplas implementações para integrar suas respectivas capacidades; ou
- elaboração de unidades. Por exemplo, associar uma unidade de projeto a unidade de requisito correspondente, e, de maneira semelhante, associar uma unidade de código a uma unidade de projeto correspondente.

Ao contrário da Programação Orientada a Aspectos, que faz a separação explícita das preocupações transversais numa entidade especial denominada aspecto, na abordagem de hiper-espacos preocupações transversais e não-transversais são tratadas igualmente. No entanto, um aspecto pode ser visualizado como um *hyperslice* que encapsula uma preocupação transversal. Outra diferença é que na Programação Orientada a Aspectos, as regras de composição e os pontos de junção são especificados no próprio aspecto, enquanto na abordagem de hiper-espacos essas informações são definidas à parte, numa unidade denominada de *hypermodule*.

3.2.2.1 HYPER/J

A ferramenta IBM Hyper/J [OSSHER & TARR, 2000] suporta a abordagem de hiper-espacos para manipular unidades Java, ou seja, classes, pacotes, métodos, etc.

Como Hyper/J é baseada na linguagem orientada a objetos Java, nessa linguagem cada *hyperslice* representa um fragmento de uma hierarquia de classes. Porém, cada classe contém apenas o subconjunto de métodos e variáveis que pertencem especificamente à preocupação que está sendo modularizada. Portanto, cada *hyperslice* pode ser imaginado como um tipo de visão de classes para uma preocupação específica.

⁴ anteriormente denominado de *subjects* em *Subject-Oriented Programming* [HARRISON & OSSHER, 1993]

Em Hyper/J, sistemas são construídos a partir da composição de *hyperslices*. Esta composição envolve encontrar os pontos de junção correspondentes nos diferentes *hyperslices* que estão sendo compostos e combinar cada *hyperslice* nesses pontos de junção. Na linguagem Hyper/J pontos de junção incluem classes, interfaces, métodos e variáveis.

Antes de especificar a composição entre *hyperslices*, deve-se analisar os relacionamentos entre membros de classes em *hyperlices* distintos. Regras de composição são feitas através de operadores simples e genéricos, tais como *mergeByName* e *override*. O primeiro operador indica que a composição em pontos de junção como os mesmos nomes (e assinaturas) em diferentes *hyperslices* devem ser feitos através de *merging*. O segundo operador indica que a implementação de um método num *hyperslice* sobrepõe a implementação de outro método em outro *hyperslice*. Além desses operadores de composição, outros operadores estão disponíveis tanto para lidar com questões como correspondência entre membros com nomes distintos, como para lidar com ordem de execução e valor de retorno de métodos combinados [OSSHER & TARR, 2001].

Ao implementar um sistema em Hyper/J, inicialmente é necessário identificar quais são as unidades (classes, pacotes) pertencentes a cada preocupação de uma dimensão. Para isso, deve ser feito um mapeamento de preocupações, consistindo de especificações na seguinte forma *x: dimensão.preocupação*, onde *x* é o nome da unidade (ex: classe) ou coleção de unidades (ex: pacote). Um exemplo de mapeamento pode ser visto na Figura 3.6.

x:	dimensão.preocupação
package Pessoal :	Recurso.Pessoal
operation posicao :	Recurso.FolhaPagamento
operation pagar :	Recurso.FolhaPagamento

Figura 3.6 - Exemplo em Hyper/J de mapeamento entre dimensões de preocupações (retirado de [OSSHER & TARR, 2000])

O exemplo apresentado na Figura 3.6 determina que existem dois tipos de preocupações na dimensão de Recurso do sistema: *Pessoal* e *FolhaPagamento*. A primeira sentença indica que, por *default*, todos os membros das classes e interfaces no pacote *Pessoal* pertencem à preocupação denominada *Pessoal*. As declarações seguintes

indicam que qualquer método (independente de classe) cujo nome seja `posicao` ou `pagar` pertence à preocupação denominada *FolhaPagamento*.

Quando processado por Hyper/J, esse mapeamento resultará na criação de um hiper-espço composto de duas dimensões: *Recurso* e *Classe*; como mostrado na Figura 3.7. Hyper/J cria um *hyperslice* para cada preocupação da dimensão *Recurso*, correspondendo as linhas na Figura 3.7. Isso possibilita que as preocupações com folha de pagamento e pessoal sejam separadas em *hyperslices* distintos e cada uma delas possa ser manipulada independentemente da outra.

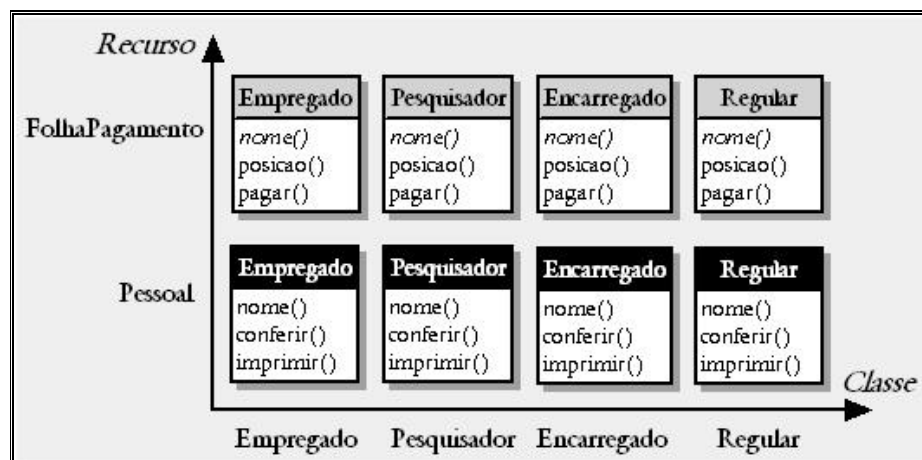


Figura 3.7 – Exemplo de hiper-espço em Hyper/J (adaptado de [OSSHER & TARR, 2000])

A Figura 3.8 representa uma declaração de composição entre esses *hyperslices* para gerar o sistema. Essa declaração indica que todas as entidades com mesmo nome nos *hyperslices* *FolhaPagamento* e *Pessoal* devem ser combinadas para formar uma única entidade.

```
hypermodule FolhaPagamentoPlusPessoal
    hyperslices: FolhaPagamento, Pessoal;

    relationships:
        mergeByName;

end hypermodule;
```

Figura 3.8 - Exemplo de declaração de composição entre *hyperslices* em Hyper/J

3.2.3 Filtros de Composição

A abordagem de Filtros de Composição [BERGMANS & AKSIT, 2001; BERGMANS et al., 2001] propõe estender o modelo convencional de objetos a fim de lidar com problemas da modelagem orientada a objetos atual, tais como: a dificuldade em modularizar preocupações como sincronização e suporte a múltiplas visões sobre um mesmo objeto.

Num sistema orientado a objetos todo comportamento é implementado através da troca de mensagens entre objetos. Baseado nesse princípio, a idéia principal dessa abordagem é separar preocupações complexas e transversais usando filtros para manipular as mensagens recebidas e enviadas por objetos. Sendo assim, cada mensagem que é recebida por um objeto deve primeiramente ser processada por todos os filtros no conjunto de filtros de entrada, na ordem em que eles foram declarados. De maneira semelhante, durante a execução de um método, todas as mensagens enviadas para outros objetos devem primeiramente passar pelo conjunto de filtros de saída. Dependendo do processamento do filtro, uma mensagem que chega pode, portanto, ser ignorada, substituída por outra mensagem e/ou enviada para um objeto diferente.

Um filtro é composto de três partes: (i) uma condição ou expressão, que deve ser verdadeira para o filtro ser ativado; (ii) uma parte de combinação especificando um padrão para ser comparado com a mensagem; e (iii) uma parte de substituição especificando substituições para o transmissor ou o alvo da mensagem.

Ao processar uma mensagem, cada filtro tenta fazer o casamento de padrão das propriedades da mensagem (ex: objeto destinatário, nome) com a expressão do filtro⁵. O resultado da avaliação pode implicar na aceitação ou rejeição da mensagem pelo filtro. A semântica associada à aceitação ou rejeição da mensagem dependerá do tipo de filtro. Alguns tipos de filtros pré-definidos e sua semântica correspondente são apresentados na Tabela 3.1. No entanto, outros tipos de filtro podem ser introduzidos na abordagem.

⁵ detalhes sobre a sintaxe da especificação dos filtros podem ser encontrados em [BERGMANS et al., 2001]

Tabela 3.1 - Exemplos de tipos de filtros pré-definidos e sua semântica [BERGMANS & AKSIT, 2001]

TIPO DE FILTRO	AÇÃO DE ACEITAÇÃO	AÇÃO DE REJEIÇÃO
DISPATCH	Enviar mensagem para o objeto destinatário da mensagem	Continuar a avaliar o próximo filtro do conjunto
ERROR	Continuar a avaliar o próximo filtro no conjunto	Gerar uma exceção
WAIT	Continuar a avaliar o próximo filtro no conjunto	Colocar a mensagem numa fila enquanto a avaliação da expressão do filtro resultar numa rejeição
META	Enviar a mensagem como parâmetro de outra (meta) mensagem para um outro objeto.	Continuar a avaliar o próximo filtro no conjunto
REALTIME	Ajustar algumas propriedades de tempo real da mensagem	Continuar a avaliar o próximo filtro no conjunto

O mecanismo de filtragem de mensagens apresentado até aqui é capaz de descrever apenas a adição de comportamento transversal em métodos dentro de uma mesma classe. Para modelar a situação onde uma preocupação afeta múltiplas classes, o modelo de Filtros de Composição fornece uma extensão do mecanismo de filtragem, denominada *superimposition* [BERGMANS et al., 2001].

3.3 Paradigma orientado a aspectos

O conjunto de conceitos, mecanismos e práticas comuns às abordagens para separação de preocupações transversais está formando o que a comunidade de Engenharia de Software vem denominando de paradigma orientado a aspectos [AOSD, 2003]. O diferencial desse paradigma é que ele tem o objetivo de localizar preocupações transversais, evitando que elas fiquem espalhadas nos artefatos de software ou misturadas com as preocupações específicas desses artefatos [ELRAD et al., 2001a].

Esse paradigma representa um avanço em relação aos paradigmas anteriores, pois contribui para uma melhor separação de preocupações nos artefatos de software e, conseqüentemente, possibilita que os benefícios da aplicação do princípio da separação de preocupações (apresentados na Seção 3.1) possam ser melhor incorporados no processo de desenvolvimento.

Ainda não existe nenhum consenso a respeito dos conceitos essenciais que caracterizam o paradigma orientado a aspectos. Recentemente, a Programação Orientada a Aspectos [KICZALES et al., 1997] foi considerada uma possível convergência das abordagens para separação de preocupações transversais [ELRAD et al., 2001a]. Por isso, atualmente, muitos dos conceitos apresentados na literatura sobre esse paradigma são influenciados pela abordagem de KICZALES et al. (1997). No entanto, alguns desses conceitos podem ser encontrados nas demais abordagens para separação de preocupações transversais, com diferenças apenas em terminologia, mecanismos e notações [MASUHARA & KICZALES, 2003]. A seguir, apresentamos uma coletânea dos fundamentos comumente atribuídos a esse paradigma e que servirão de base conceitual nesse trabalho.

3.3.1 Fundamentos

De maneira geral, o paradigma orientado a aspectos **complementa** paradigmas anteriores (estruturado e orientado a objetos) em dois sentidos [ELRAD et al., 2001a]: (i) localizando preocupações transversais em unidades separadas, denominadas de aspectos; e (ii) provendo mecanismos para fazer a composição de preocupações transversais com as unidades de decomposição do sistema que elas afetam (também denominadas de unidades base).

A Figura 3.9 ilustra essas características: o comportamento da preocupação transversal é centralizado numa única entidade, conhecida como aspecto (Figura 3.9a); por sua vez, cabe ao mecanismo de composição fazer a junção do comportamento do aspecto nas classes que ele afeta de maneira não-invasiva (Figura 3.9b), ou seja, sem que a preocupação transversal fique explicitamente espalhada nas unidades que ela afeta e misturada com as preocupações próprias dessas unidades.

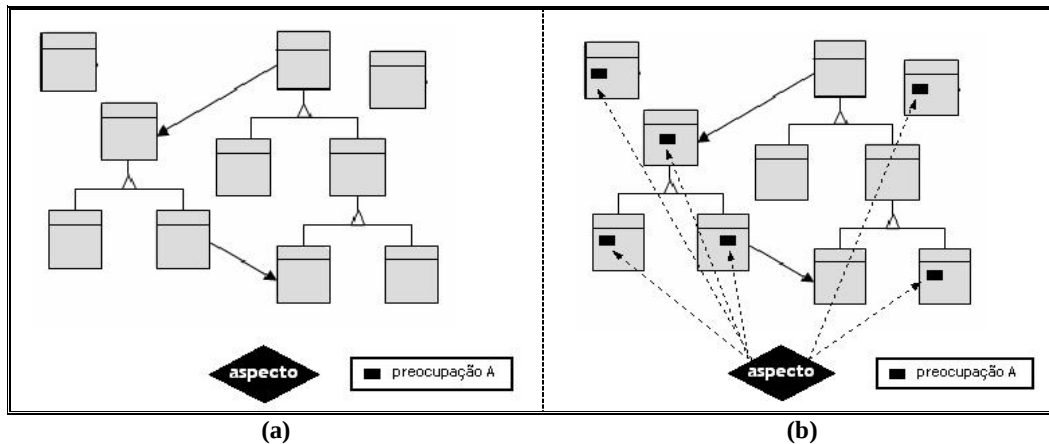


Figura 3.9 – Ilustração do funcionamento da separação e composição de aspectos

Para que um aspecto seja combinado com a(s) unidade(s) que ele afeta, são necessárias algumas informações:

- (i) ponto(s) de junção: indica em que ponto(s) na especificação da(s) unidade(s) base o comportamento do aspecto deve ser incluído [BRITO & MOREIRA, 2003];
- (ii) regra de composição: expressa a ordem seqüencial na qual o aspecto deve ser combinado com a(s) unidade(s) que ele afeta [MOREIRA et al., 2002], ou seja, especifica como o comportamento do aspecto deve ser aplicado no(s) ponto(s) de junção. As regras de composição mais comuns são: “antes”, “depois”, e “ao invés de”;

Exemplificando: quando dito que o aspecto de autenticação de senha deve ser aplicado antes da execução de transações bancárias, o termo antes corresponde a uma regra de composição e a sentença execução de transações bancárias corresponde ao ponto de junção.

Existem quatro possibilidades de especificar a composição entre um aspecto e unidades base [KERSTEN & MURPHY, 1999]:

- fechada: o aspecto e a unidade base não “tem conhecimento” sobre a existência um do outro, ou seja, eles não se referenciam diretamente;
- aberta: aspecto e a unidade base “têm conhecimento” sobre a existência um do outro, ou seja, eles se referenciam mutuamente;
- dirigida ao componente: somente o aspecto referencia a unidade base; e

- dirigida ao aspecto: somente a unidade base referencia o aspecto.

A associação do tipo fechada é a preferencial, visto que permite a independência tanto das unidades base quanto do aspecto, promovendo, então, melhor potencial para que os artefatos envolvidos possuam qualidades como reusabilidade, compreensibilidade e manutenibilidade. A associação do tipo aberta deve ser evitada, pois resulta em acoplamento entre os dois tipos de artefatos e, conseqüentemente, compromete o alcance da separação de preocupações e seus benefícios. Por sua vez, as associações do tipo dirigida ao componente e dirigida ao aspecto, promovem as qualidades já citadas apenas com relação a unidade base e ao aspecto, respectivamente. Os tipos de associações mais comuns nas abordagens orientadas a aspectos são: fechada e dirigida ao componente.

3.3.2 Identificação de aspectos

A orientação a aspectos está passando pelos mesmos problemas vivenciados pela orientação a objetos há alguns anos atrás: faltam processos, técnicas e diretrizes que auxiliem o desenvolvedor a aplicar os fundamentos dessa abordagem. Um exemplo dessa situação é a identificação de aspectos. Atualmente, não existem meios sistemáticos de identificar aspectos: essa atividade é largamente baseada na experiência e intuição do desenvolvedor.

Um ponto de partida para essa identificação geralmente são preocupações não-funcionais. Requisitos originados de preocupações não-funcionais são naturalmente transversais porque impõem restrições em como a funcionalidade do sistema será realizada, e por isso, são quase sempre associados a requisitos funcionais [KOTONYA & SOMMERVILLE, 1998; CYSNEIROS et al., 2001]. No entanto, a natureza de aspectos não se limita a esse tipo de requisito; aspectos podem representar tanto preocupações funcionais como não-funcionais [CONSTANTINIDES & HASSOUN, 2002].

Nos demais casos, o desenvolvedor geralmente recorre aos “sinais de preocupações transversais”, ou seja, considerando uma unidade de decomposição base (ex: casos de uso, classe), identificam-se preocupações que, usando os mecanismos

atuais de orientação a objetos, ficariam espalhadas nas unidades que elas devem ser consideradas ou misturadas com as preocupações próprias dessas unidades.

3.4 Desenvolvimento de software orientado a aspectos

Como mencionado anteriormente, o paradigma orientado a aspectos busca promover meios para uma melhor separação de preocupações transversais. Já que os fundamentos desse paradigma partiram de abordagens voltadas para o projeto e implementação, suas práticas vêm sendo utilizadas quase exclusivamente na implementação de sistemas. Entretanto, problemas de espalhamento e emaranhamento de preocupações transversais em artefatos de software não ocorrem exclusivamente na codificação do sistema. Por isso, recentemente, a comunidade de pesquisa tem se interessado em utilizar os fundamentos desse paradigma em atividades iniciais do ciclo de desenvolvimento.

Nesse contexto, o desenvolvimento de software orientado a aspectos surge com o objetivo de tratar preocupações transversais de maneira explícita, fornecendo meios para sua sistemática identificação, separação, representação e composição ao longo de todas as atividades do ciclo de desenvolvimento. Nesse tipo de desenvolvimento, preocupações transversais são encapsuladas em unidades separadas, conhecidas como aspectos, a fim de que a localização dessas preocupações possa ser alcançada. Isso resulta em melhor suporte a modularização, o que contribui para alcançar qualidades como reusabilidade, manutenibilidade e compreensibilidade nos artefatos de software e, conseqüentemente, para redução de custos de desenvolvimento, manutenção e evolução.

A seguir, apresentamos uma revisão bibliográfica sobre propostas para dar suporte ao desenvolvimento de software orientado a aspectos nas atividades de requisitos e projeto.

3.4.1 Abordagens relacionadas à engenharia de requisitos orientada a aspectos

Propostas relacionadas às atividades de requisitos no contexto de desenvolvimento de software orientado a aspectos pertencem a uma área de pesquisa muito recente, denominada de Engenharia de Requisitos Orientada a Aspectos. Essa área de pesquisa concentra-se, entre outras coisas, nas seguintes questões [WORKSHOP ON EARLY ASPECTS, 2003]: (i) como identificar e modelar aspectos no nível de requisitos; (ii) como integrar e compor aspectos com outros mecanismos de modelagem de requisitos, tais como metas, pontos de vista e casos de uso; e (iii) como rastrear aspectos identificados no nível de requisitos nas atividades seguintes de desenvolvimento. Atualmente a maior parte dos trabalhos nessa área estão focadas nos dois primeiros itens, como mostramos a seguir.

3.4.1.1 PROCESSO AORE E EXTENSÕES

RASHID et al. (2002) propôs um modelo de processo genérico, denominado AORE (*Aspect-Oriented Requirements Engineering*), para separar preocupações transversais no nível de requisitos. No modelo AORE, o termo “preocupação” (ou *concern*) é utilizado num sentido mais restrito, correspondendo a um requisito não-funcional de alto nível de abstração (ex: segurança, compatibilidade).

Segundo o modelo AORE original, inicialmente é necessário identificar requisitos funcionais e preocupações^{*6}. A ordem dessas duas atividades não é determinada pelo modelo: ela dependerá da interação entre os engenheiros e os usuários do sistema. A seguir, deve-se relacionar preocupações^{*} a requisitos funcionais numa matriz (veja Tabela 3.2).

Tabela 3.2 - Matriz para relacionar preocupações^{*} com requisitos funcionais (retirado de [RASHID et. al, 2003])

	Req 1	Req 2	...	Req n
Preocupação [*] 1				✓
Preocupação [*] 2		✓		✓
...				
Preocupação [*] n	✓	✓		

⁶ Para distinguir o uso geral do termo preocupação e o uso nos trabalhos de RASHID et al. (2002) e RASHID et al. (2003), nessa seção representaremos esse último como preocupação^{*}.

Se uma preocupação* influenciar ou restringir mais de um requisito funcional, ela é considerada um aspecto candidato. A próxima atividade é responsável por analisar aspectos candidatos em mais detalhes, identificando conflitos entre eles e estabelecendo prioridades. Por fim, deve-se especificar o impacto de aspectos candidatos em termos de duas dimensões: mapeamento e influência. A dimensão de mapeamento especifica como cada aspecto candidato será manipulado nos estágios seguintes do desenvolvimento. Aspectos candidatos podem ser mapeados em uma função, decisão arquitetural ou num aspecto. Por sua vez, a dimensão de influência estabelece quais os pontos do ciclo de desenvolvimento que o aspecto candidato afetará. A Tabela 3.3 mostra um exemplo de saída dessa atividade. Contudo, não são apresentadas heurísticas para determinar como o engenheiro de requisitos estabelecerá essas dimensões de aspectos candidatos.

Tabela 3.3 – Exemplo de Saída da Atividade *Especificar Dimensão de Aspectos Candidatos* do modelo AORE (retirado de [RASHID et. al, 2002])

ASPECTO CANDIDATO	INFLUÊNCIA	MAPEAMENTO
Compatibilidade	Especificação, arquitetura, projeto e manutenção	<i>Função</i>
Tempo de Resposta	Arquitetura, Projeto	<i>Aspecto</i>
Restrições Legais	Especificação	<i>Função</i>
Corretude	Especificação e Projeto	<i>Função</i>
Segurança	Arquitetura e Projeto	<i>Aspecto</i>
Disponibilidade	Arquitetura	<i>Decisão</i>
Sistema Multi-usuário	Arquitetura e Projeto	<i>Aspecto</i>

RASHID et al. (2003) apresenta uma melhoria no modelo AORE original. A principal modificação realizada foi a inclusão de duas novas atividades: uma responsável por compor aspectos candidatos e requisitos, determinando como um aspecto candidato influencia ou restringe um requisito através de regras de composição; e a outra responsável por lidar com conflitos. Essa última atividade é realizada preenchendo-se uma matriz de contribuição positiva e negativa entre aspectos candidatos e atribuindo-se pesos quando um aspecto candidato contribuir negativamente para a realização de outro. O modelo de processo resultante é apresentado na Figura 3.10.

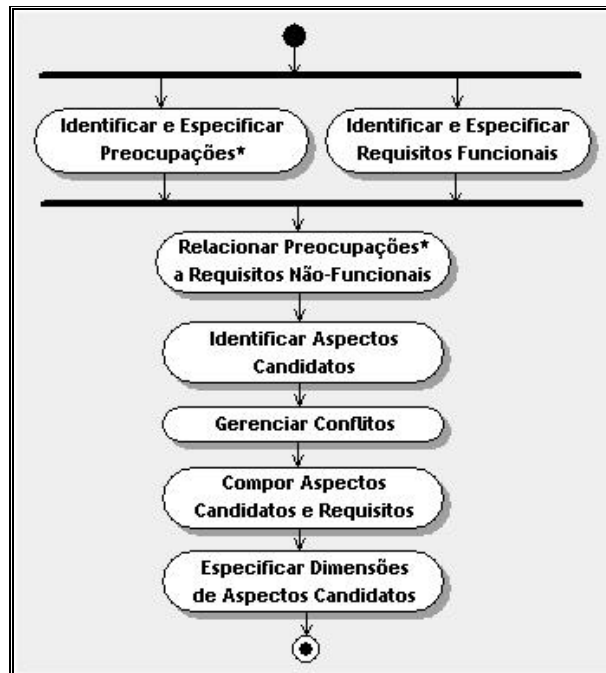


Figura 3.10 - Modelo de Processo AORE (adaptado de [RASHID et. al, 2003])

RASHID et al. (2002) e RASHID et al. (2003) apresentam uma realização do processo AORE baseada em pontos de vista (*viewpoints*) [FINKELSTEIN & SOMMERVILLE, 1996].

Por sua vez, MOREIRA et al. (2002), ARAUJO et al. (2002) e BRITO & MOREIRA (2003) usam uma instância simplificada do modelo AORE baseada em UML[BOOCH et al., 1999]. Nesses trabalhos, os requisitos funcionais são representados usando diagramas UML (diagramas de casos de uso e seqüência) e os requisitos não funcionais (preocupações*) são descritos usando modelos pré-definidos nos quais são inseridas referências explícitas aos requisitos funcionais afetados pelo requisito não funcional. A composição de requisitos não-funcionais e funcionais é representada graficamente através de adaptações de caso de uso e diagramas de seqüência. Para especificar como a composição deve ser realizada, são utilizados os seguintes operadores de composição:

- overlap: usado quando o comportamento do aspecto deve ser aplicado antes (*before*) ou depois (*after*) do requisito que ele afeta;

- override: utilizado quando o comportamento do aspecto substitui o comportamento do requisito que ele afeta;
- wrap: usado quando a realização do aspecto encapsula o comportamento do requisito que ele afeta

Um exemplo de composição em diagrama de caso de uso, usando esses operadores de composição num sistema de cobrança de pedágio é mostrado na Figura 3.11. Nesse exemplo, o aspecto de segurança afeta os casos de uso *Pagar fatura* e *Registrar veículo* com o operador *overlap*; e o aspecto de tempo de resposta afeta os casos de uso *Registrar veículo*, *Entrar na rodovia*, *Sair da rodovia* e *Passar por cobrança de pedágio*. Entretanto, não é indicado em que pontos do fluxo dos casos de uso esses aspectos devem ser aplicados; nem tão pouco é especificado como as preocupações de segurança e tempo de resposta serão operacionalizadas no sistema de cobrança de pedágio.

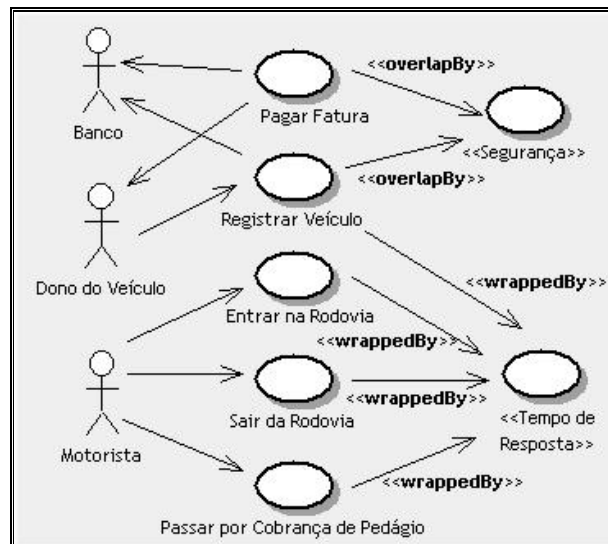


Figura 3.11 - Exemplo de composição em diagrama de caso de uso (adaptado de [BRITO & MOREIRA, 2003])

3.4.1.2 CASOS DE USO

Recentemente, Jacobson, o criador do conceito de caso de uso, tem se interessado na relação entre casos de uso e o desenvolvimento orientado a aspectos.

JACOBSON (2003a) defende que o mecanismo de extensão de caso de uso poderia ser utilizado para modelar aspectos no nível de requisitos. Em resumo, esse mecanismo modela o relacionamento onde um caso de uso de extensão adiciona comportamento a um caso de uso base, sem modificar o caso de uso original. O relacionamento de extensão necessita das seguintes informações: uma condição para a extensão (quando a extensão é condicional) e uma referência para o ponto de extensão no caso de uso alvo (uma posição no caso de uso onde a adição de comportamento deve ser feita). Detalhes sobre o mecanismo de extensão podem ser encontrados na Seção 2.2.3

A Tabela 3.4 exhibe algumas das equivalências entre o mecanismo de extensão de casos de uso e alguns conceitos utilizados na orientação a aspectos, segundo JACOBSON (2003b).

Tabela 3.4 - Equivalências entre conceitos de extensão de casos de uso e os de orientação a aspectos (extraído de [JACOBSON, 2003b])

EXTENSÃO DE CASO DE USO	ORIENTAÇÃO A ASPECTOS
Caso de uso de extensão	Aspecto
Ponto de extensão	Ponto de junção
Comportamento do caso de uso estendido	<i>Advice</i>

Outra proposta de JACOBSON (2003b) é modificar a filosofia do desenvolvimento de software guiado por casos de uso utilizando princípios semelhantes aos de MDSOC (Seção 3.2.2). A idéia é que exista um modelo de análise e projeto e um modelo de implementação específicos para cada caso de uso, não havendo, a princípio, a preocupação em reusar partes de modelos de outros casos de uso. Apenas as classes, atributos e métodos essenciais à implementação do caso de uso são criados, mesmo embora seja esperado que outros casos de uso possam precisar de classes conceitualmente similares com métodos e atributos adicionais. Assim, casos de uso poderão ser desenvolvidos independentemente durante todo o processo de desenvolvimento e depois combinados para compor o sistema.

A separação de casos de uso seria feita produzindo-se partes do sistema, nomeadas de módulos de caso de uso⁷ similares a idéia de um *hyperslice* em MDSOC. Um módulo de caso de uso encapsularia todos os componentes necessários para sua concretização. Por fim, essas partes seriam combinadas para produzir o sistema desenvolvido. Por exemplo, a Equipe 1 fica responsável por todo desenvolvimento dos casos de uso A e B, a Equipe 2 fica responsável pelo caso de uso C e a Equipe 3 pelos casos de uso D e E. A aplicação é então composta através da combinação dessas implementações.

Contudo, JACOBSON, 2003b apenas apresenta essas idéias, mas não indica como isso poderia ser realizado na prática. Uma implicação negativa que identificamos nessa abordagem que ele propõe é que, partindo do princípio que os casos de uso serão desenvolvidos independentemente, possivelmente por equipes diferentes, é muito provável que ocorram redundâncias nas implementações; por exemplo, uma mesma função ser implementada por diferentes equipes. Outro problema é que nomes diferentes podem ser utilizados para um mesmo elemento conceitual ou pior: o mesmo nome pode ser usado para diferentes propósitos. A princípio, o processo de combinação pode resolver esses problemas, mesclando redundâncias e fornecendo métodos para corrigir nomeações e outras divergências. Algumas práticas como: estabelecimento de convenções para nomenclatura; o uso de um glossário de termos e conceitos específico para o domínio da aplicação; e a comunicação freqüente com outras equipes, ajudam a minimizar esses problemas [WAMPLER, 2003].

3.4.1.3 COMPARAÇÃO ENTRE ABORDAGENS APRESENTADAS

Pode-se dizer que o modelo de processo AORE [RASHID et. al, 2002; RASHID et. al, 2003] consistiu no primeiro passo em direção ao estabelecimento de um conjunto de atividades de requisitos preocupadas com a separação de requisitos transversais e com sua composição com os requisitos que eles afetam. Os trabalhos de MOREIRA et al. (2002), ARAUJO et al. (2002) e BRITO & MOREIRA (2003) estabeleceram uma instância simplificada do processo genérico AORE, preocupando-se em melhorar a composição entre requisitos transversais e não-transversais utilizando

⁷ tradução livre do inglês *use-case module*

operadores de composição que refletiam aqueles utilizados nas principais abordagens orientadas a aspectos [OSSHER & TARR, 2000; BERGMANS & AKSIT, 2000; KICZALES et al., 2001]. Contudo, esses trabalhos manipulam com preocupações não-funcionais de maneira abstrata, vaga e informal (ex.: segurança, disponibilidade, tempo de resposta, compatibilidade, etc), o que dificulta o entendimento sobre o real significado da preocupação não-funcional, como satisfazê-la e verificá-la, já que diferentes interpretações são possíveis [SOMMERVILLE, 1996; CHUNG et al., 2000]. Para um sistema, a preocupação de segurança pode significar apenas a utilização de um *login*, enquanto para outros ela poderá corresponder à assinatura, reconhecimento de voz, impressão digital, etc. Outras duas inconveniências desses trabalhos com o modelo AORE são: (i) eles focam primordialmente em aspectos não-funcionais; e (ii) a composição desses aspectos com os casos de uso que eles afetam não especifica em que parte do cenário dos casos de uso o aspecto deverá ser considerado.

JACOBSON (2003b) apresenta uma outra iniciativa, defendendo que o mecanismo de extensão de casos de uso de UML poderia ser usado para modelar aspectos no nível de requisitos. Porém, esse mecanismo vem sendo utilizado em situações que nem sempre se adequam ao conceito de aspectos como, por exemplo, para representar fluxos alternativos de um caso de uso. Por isso, seria interessante distinguir o papel do mecanismo de extensão nas situações em que ele modela aspectos e naquelas em que ele modela apenas fluxos alternativos. Além disso, JACOBSON, 2003b não oferece heurísticas para identificação de aspectos no nível de requisitos e nem menciona como a interferência dos aspectos seria modeladas nos diagramas responsáveis pela concretização do caso de uso (diagramas de seqüência e colaboração). Outra dificuldade dessa abordagem é que a especificação das extensões de um caso de uso é realizada de maneira invasiva, tanto no caso de uso base como no caso de uso de extensão (detalhes na Seção 4.2.3.4).

A Tabela 3.5 apresenta uma comparação entre as abordagens para dar suporte ao desenvolvimento de software orientado a aspectos nas atividades de requisitos apresentadas na Seção 3.4.1. Percebe-se que existem algumas convergências nessas abordagens, tais como a especificação do aspecto em artefato específico para ele. Porém, existem algumas divergências, como: (i) a maneira de representar o

relacionamento entre aspecto e unidades base, os pontos de junção e as regras de composição; e (ii) o tipo de composição adotada.

Tabela 3.5 -Comparação entre abordagens pesquisadas para modelar aspectos no fluxo de requisitos

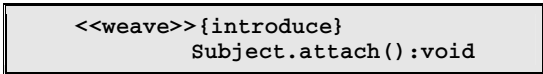
ABORDAGEM EM AORE	REPRESENTAÇÃO DOS FUNDAMENTOS DE ORIENTAÇÃO A ASPECTOS				
	ASPECTO	RELACIONAMENTO ENTRE ASPECTO E UNIDADES BASE	PONTOS DE JUNÇÃO	REGRAS DE COMPOSIÇÃO	TIPO DE COMPOSIÇÃO
RASHID ET AL. (2003)	<i>Num arquivo XML com estrutura pré-definida para o aspecto</i>	—	<i>Definidos num arquivo XML específico para composição entre o aspecto e as unidades base</i>	<i>Definidas num arquivo XML específico para composição entre o aspecto e as unidades base</i>	<i>Fechada</i>
ARAUJO ET AL. (2002); BRITO & MOREIRA (2003)	<i>Tabela com campos para especificar as propriedades do aspecto</i>	<i>Relacionamento entre casos de uso, estereotipado com rótulos correspondentes aos operadores de composição</i>	<i>Especificados na tabela de cada aspecto</i>	<i>Uso de estereótipos nos relacionamentos entre casos de uso aspectuais e casos de uso base</i>	<i>Dirigida ao componente</i>
JACOBSON (2003b)	<i>Caso de uso estendido</i>	<i>Representada por relação do tipo extend entre casos de uso</i>	<i>Pontos de junção correspondem a ponto de extensão, que são especificados no caso de uso de extensão</i>	—	<i>Dirigida ao componente</i>

Vale a pena ressaltar que o tipo de composição é um dos fatores determinantes para se obter vantagens da orientação a aspectos como melhor reuso e menor acoplamento entre artefatos. A composição fechada é o tipo de composição que permite um melhor desacoplamento dos artefatos envolvidos (Seção 3.3.1). Porém, um dos poucos trabalhos a se preocupar em fazer essa composição de maneira fechada, ou seja, num artefato a parte, foi o de RASHID et al. (2003). Em [ARAUJO et al., 2002; BRITO & MOREIRA, 2003], a composição é dirigida ao componente, já que na própria especificação do aspecto são referenciados o requisitos afetados pela preocupação transversal. A proposta de JACOBSON (2003b), que propõe a utilização do mecanismo de extensão de casos de uso para modelar a composição, também caracteriza uma composição do tipo dirigida ao componente, visto que os pontos de junção são inseridos na especificação do caso de uso de extensão.

3.4.2 Abordagens relacionadas ao projeto orientado a aspectos

O projeto orientado a aspectos é uma parte crítica do desenvolvimento de software orientado a aspectos. Essa área de pesquisa tem se focado em notações, técnicas e ferramentas para identificar, estruturar e representar preocupações transversais no projeto de software [PACE & CAMPO, 2001], especialmente no que se refere a extensão da linguagem UML para modelar aspectos. A tendência atual é a criação de um perfil (*profile*) UML para a modelagem de aspectos [ALDAWUD et al., 2001], de maneira análoga ao que aconteceu com UML-RT (para sistemas de tempo real) e UML-CORBA (para sistemas que usam a IDL de CORBA).

Uma das primeiras propostas de extensão do meta-modelo da linguagem UML para suportar a modelagem de aspectos nas atividades de projeto foi apresentada por SUZUKI & YAMAMOTO (1999). A primeira modificação proposta foi a inclusão de um novo tipo de classificador no meta-modelo UML, semelhante ao classificador de classe, chamado *aspect*. Segundo essa proposta, os elementos (métodos e variáveis) introduzidos pelo aspecto nas classes que ele afeta são definidos na entidade *aspect* de maneira semelhante à definição de métodos e variáveis em uma classe, mas devem ser antecedidos do estereótipo⁸ <<weave>> (exemplo na Figura 3.12). Para modelar o relacionamento entre o aspecto e as classes afetadas por ele, SUZUKI & YAMAMOTO (1999) usam um tipo de relacionamento de dependência com o estereótipo <<realize>>, já estipulado pela UML.



```
<<weave>>{introduce}  
Subject.attach():void
```

Figura 3.12 - Exemplo de declaração de combinação (retirado de [SUZUKI & YAMAMOTO, 1999])

No entanto, como apontado por STEIN et al. (2002), a proposta de SUZUKI & YAMAMOTO (1999) possui duas dificuldades fundamentais: (i) foca apenas no caso onde o aspecto introduz atributos e operações na classe, não mencionando como modelar o caso onde comportamentos são adicionados pelo aspecto dentro de métodos

⁸ Estereótipo é um dos mecanismos de extensão de UML cuja função é representar uma subclasse de um elemento existente no metamodelo com a mesma forma (relacionamentos e atributos), mas com uma intenção diferente [OMG, 2003]

da classe; e (ii) utiliza de maneira inadequada o tipo de relacionamento de dependência *realize*, já que este deve ser usado para representar o relacionamento entre um elemento de modelo e outro elemento que o implementa, o que não caracteriza o relacionamento entre um aspecto e as classes afetadas por ele.

Outra proposta é apresentada por ZAKARIA et al. (2002), que sugere o uso de um estereótipo do elemento *class* rotulado de `<<aspect>>` para modelar unidades que encapsulam comportamento transversal em UML. Nesse trabalho, a relação aspecto-classe é representada através de um relacionamento do tipo associação. Além disso, ZAKARIA et al. (2002) propõe outras extensões na linguagem UML baseadas nos construtores de AspectJ [KICZALES et al., 2001], como é ilustrado na Figura 3.13 :

- modelar um *pointcut* como um estereótipo do elemento *class*. O elemento `<<pointcut>>` seria vinculado ao elemento `<<aspect>>` através de uma associação rotulada com o estereótipo `<<has pointcut>>`. Ao invés da declaração de métodos, a classe do tipo *pointcut* definiria quais métodos das classes o aspecto iria afetar. Visto que um estereótipo representa uma distinção de uso [BOOCH et al., 1999], esse tipo de estereótipo para o elemento *class* não nos parece adequado, pois uma classe do tipo *pointcut* serve apenas para definir os pontos onde o aspecto afeta as classes do sistema e não possui propriedades e métodos associados a ele, assim como uma classe possui;
- modelar um *advice* usando um estereótipo do elemento *operation* dentro da classe do tipo *aspect*. Para cada tipo de *advice* em AspectJ, existiria um estereótipo correspondente.

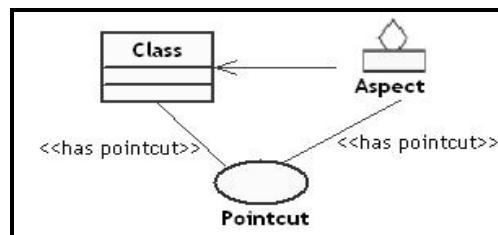


Figura 3.13 - Representação gráfica da modelagem de aspectos segundo ZAKARIA et al. (2002)

PAWLAK et al. (2002) apresenta outro trabalho que modela um aspecto usando o estereótipo `<<aspect>>` para o elemento *class*, pois considera um aspecto

semelhante a uma classe comum (i.e. com atributos e métodos), exceto por conter métodos com uma semântica particular. Esses métodos “especiais”, chamados de métodos aspectuais, representam o comportamento que o aspecto incluirá em outras classes. Estereótipos especiais são colocados antes desses métodos para especificar regras de composição:

- `<<before>> m(args)`: indica que o método `m` é executado antes do ponto de junção da classe estendida;
- `<<after>> m(args)`: indica que o método `m` é executado depois do ponto de junção da classe estendida;
- `<<around>> m(args)`: determina que uma parte do método `m` é executada antes e outra é executada depois do ponto de junção da classe estendida (essas duas partes são definidas dentro da implementação de `m` e dependerá da linguagem orientada a aspectos utilizada);
- `<<replace>> m(args)`: determina que o método `m` modifica o ponto de associação da classe estendida substituindo-o pela implementação de `m`.

Na abordagem de PAWLAK et al. (2002), a associação entre um aspecto e cada classe afetada por ele segue a representação exibida na Figura 3.14. O relacionamento de associação estereotipado com o rótulo `<<pointcut>>` determina que existe algum(s) método(s) do aspecto `A` que estende algum(s) método(s) da classe `C`. O rótulo `r2` define o conjunto de pontos afetados na classe base usando expressões e palavras-chaves apresentadas em [PAWLAK et al., 2002]. O rótulo `r1` deve ser substituído pelo nome do método do aspecto que é aplicado em cada ponto denotado pelo rótulo `r2`.

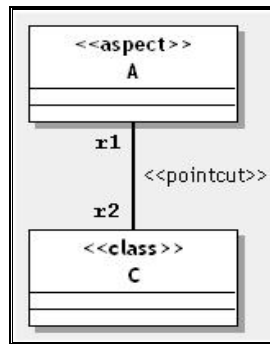


Figura 3.14 - Representação gráfica da modelagem de aspectos segundo PAWLAK et al. (2002)

As abordagens anteriormente apresentadas inserem a semântica de orientação a aspectos em diagramas de classe UML basicamente através de estereótipos. Diferentemente, CLARKE & WALKER (2001) apresentam uma abordagem, denominada padrões de composição (*composition patterns*), que propõe modelar aspectos através de *templates* de pacotes UML⁹ parametrizáveis. Essa técnica é dividida em três partes:

- i) especificar um *template* de pacote UML para o aspecto, no qual devem ser incluídas as classes necessárias para modelagem do comportamento transversal e definidos os parâmetros que representarão o papel das classes afetadas pelo aspecto. Um exemplo de *template* para um aspecto que rastreia a execução de métodos num sistema é apresentado na Figura 3.15. A classe *Trace* é uma classe auxiliar com métodos que serão utilizados pelo aspecto. A classe *TracedClass* e o método *_tracedOp()*, representam parâmetros formais que serão substituídos respectivamente pelas classes e pelos métodos afetados pelo aspecto.

⁹ UML define um *template* como um elemento do modelo que é parametrizado e não pode ser usado diretamente num modelo de projeto [OMG, 2003].

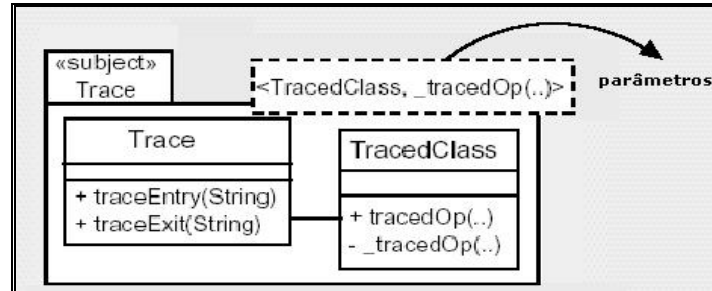


Figura 3.15 – Exemplo de *template* de pacote UML parametrizável para modelar o aspecto de rastreamento (retirado de [CLARKE & WALKER, 2001])

ii) especificar como o comportamento provido pelo aspecto deve interagir com o comportamento das classes que ele afeta usando diagramas de interação. Na Figura 3.16, é ilustrado como as operações do aspecto de *Trace* iriam interagir com a operação *_tracedOp()*, que representa uma possível operação interceptada pelo aspecto.

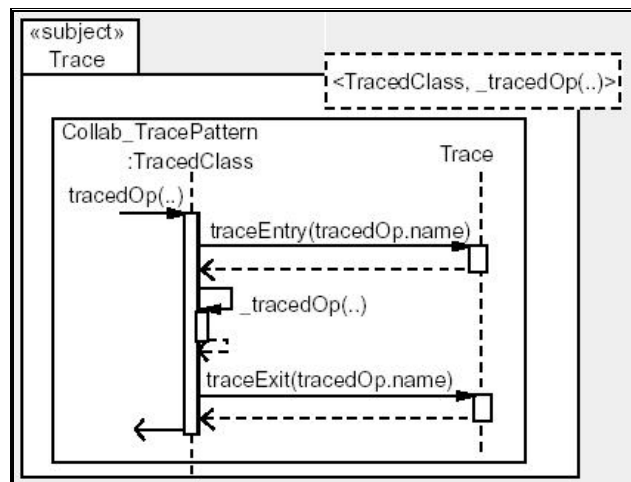


Figura 3.16 - Exemplo de padrão de comportamento para o aspecto de *Trace* (retirado de [CLARKE & WALKER, 2001])

iii) especificar o relacionamento de *binding* entre o *template* do pacote do aspecto e os pacotes que ele irá afetar. A Figura 3.17 apresenta um exemplo de especificação desse tipo de relacionamento. Nesse exemplo o *template* do pacote de *Trace* é associado ao pacote S1 e o relacionamento indica que o parâmetro *TracedClass* desse *template* deverá ser substituído por todas as classes do pacote S1 e o parâmetro *_tracedOp()* deverá ser trocado por todos

os métodos em cada uma dessas classes. Dessa maneira, todas as classes do pacote S1 incluirão a preocupação de rastreamento em seus modelos.

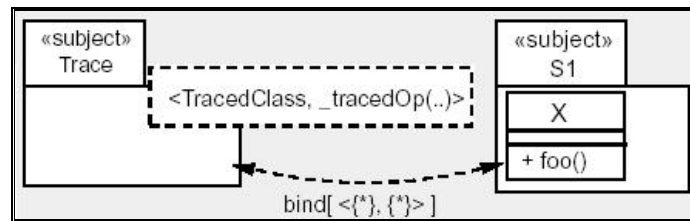


Figura 3.17 – Exemplo de composição do aspecto de *Trace* usando o relacionamento de *binding* (retirado de [CLARKE & WALKER, 2001])

Pelo fato de modelar o aspecto de forma parametrizável, não sendo feitas referências às classes que ele afeta e sim utilizando parâmetros para representá-las, a abordagem de CLARKE & WALKER (2001) possibilita que o modelo de projeto do aspecto seja extensível e reusável. Uma descrição mais completa das extensões do metamodelo UML necessárias para suportar essa abordagem de padrões de composição pode ser encontrada em [CLARKE, 2002]. Uma evolução desse trabalho é a abordagem Theme/UML [CLARKE & WALKER, 2002] que pretende provê uma linguagem de projeto padrão para AOSD a partir dos princípios de *subject-oriented design* [CLARKE et al., 1999] e da abordagem de padrões de composição.

3.4.2.1 COMPARAÇÃO ENTRE ABORDAGENS APRESENTADAS

A Tabela 3.6 compara algumas características das abordagens de projeto orientado a aspectos apresentadas na Seção 3.4.2.

Das quatro propostas analisadas, duas delas (ZAKARIA et al. (2002) e PAWLAK et al. (2002)) apresentam semelhanças em relação a: (i) representação de um aspecto usando o estereótipo <<aspect>> do elemento *class*; (ii) utilização do relacionamento de associação para representar relacionamento entre aspecto e unidades base; (iii) representação de regras de composição usando estereótipos para as operações do aspecto. Por outro lado, em nenhuma das propostas analisadas houve convergência no que se refere à representação dos pontos de junção. Com relação ao tipo de composição, somente a abordagem de CLARKE & WALKER (2001) utilizou o tipo de composição fechada; nas demais o tipo de composição adotado foi dirigida ao componente.

Tabela 3.6 - Comparação entre algumas abordagens para modelar aspectos no nível de projeto

ABORDAGEM EM AOD	REPRESENTAÇÃO DOS FUNDAMENTOS DE ORIENTAÇÃO A ASPECTOS				
	ASPECTO	RELACIONAMENTO ENTRE ASPECTO E UNIDADES BASE	PONTOS DE JUNÇÃO	REGRAS DE COMPOSIÇÃO	TIPO DE COMPOSIÇÃO
SUZUKI & YAMAMOTO (1999)	<i>Novo tipo de classificador, chamado aspect</i>	<i>Relacionamento de dependência, rotulado com o estereótipo <<realize>></i>	<i>Definidos na própria unidade que representa o aspecto</i>	-	<i>Dirigida ao componente</i>
ZAKARIA ET AL. (2002)	<i>Estereótipo do elemento class rotulado de <<aspect>></i>	<i>Relacionamento de associação</i>	<i>Definidos numa classe com estereótipo <<pointcut>></i>	<i>Definidos na classe do tipo aspect, usando estereótipos (before, after, around, after returning, after throwing) para as operações</i>	<i>Dirigida ao componente*¹⁰</i>
PAWLAK ET AL. (2002)	<i>Estereótipo do elemento class rotulado de <<aspect>></i>	<i>Relacionamento de associação, rotulado com o estereótipo <<pointcut>></i>	<i>Definido como uma tag do relacionamento de associação entre o aspecto e a classe</i>	<i>Definidos na classe do tipo aspect, usando estereótipos <<before>>, <<after>>, <<around>> ou <<replace>> para as operações</i>	<i>Dirigida ao componente*</i>
CLARKE & WALKER (2001)	<i>template de pacote UML</i>	<i>Relacionamento do tipo binding entre template de pacote UML e os pacotes nos quais o aspecto deverá ser aplicado</i>	<i>Correspondem aos parâmetros reais definidos no template do aspecto</i>	<i>Definidas em diagramas de interação dentro do template do aspecto</i>	<i>Fechada</i>

3.5 Considerações finais

Orientação a aspectos é uma evolução dos paradigmas de desenvolvimento de software existentes. Seu objetivo é complementar os fundamentos estabelecidos nesses paradigmas para melhorar a separação de preocupações transversais no desenvolvimento de software, minimizando o acoplamento entre os artefatos e, conseqüentemente, provendo melhor compreensibilidade, manutenibilidade e reusabilidade nos artefatos gerados ao longo do processo de desenvolvimento.

O paradigma orientado a aspectos está em processo de formação e ainda não existe um consenso sobre os conceitos e práticas que ele inclui. Se por um lado, as

¹⁰ O “*” nesse caso indica que é parcialmente dirigida ao componente já que os pontos de associação são especificados fora do aspecto, mas as regras de composição são definidas dentro do aspecto.

linguagens de programação que suportam o paradigma orientado a aspectos estão relativamente estabelecidas, por outro lado, métodos para serem usados nas atividades de requisitos, análise e projeto orientado a aspectos ainda são objetos de pesquisa e encontram-se em estágios iniciais: existem muitas propostas, mais pouca coerência e continuidade entre elas.

Além disso, atualmente, a grande maioria das propostas para o desenvolvimento de software orientado a aspectos é pontual, ou seja, refere-se a um estágio de desenvolvimento específico. Em especial, há uma carência de métodos de desenvolvimento orientados a aspectos que cubram atividades desde a análise de requisitos até o projeto de aspectos. Esse fato dificulta a adoção do desenvolvimento orientado a aspectos pelo mercado. Em contrapartida, a pouca experiência prática da utilização dos fundamentos da orientação a aspectos em todo o processo de desenvolvimento impossibilita a validação prática dos possíveis benefícios da utilização dos fundamentos desse paradigma em todo processo de desenvolvimento.

Nesse contexto, conforme veremos no capítulo seguinte, esta dissertação fornece meios para que os fundamentos do paradigma orientado a aspectos sejam utilizados em algumas das atividades dos fluxos de requisitos, análise e projeto do Processo de Desenvolvimento Unificado [JACOBSON et al., 1999]. Assim, será possível alcançar uma melhor separação de preocupações nos artefatos dessas atividades e, além disso, ter um processo de desenvolvimento que considere a orientação a aspectos desde requisitos até o projeto.

4

ADAPTANDO O PROCESSO UNIFICADO PARA O DESENVOLVIMENTO DE SOFTWARE ORIENTADO A ASPECTOS

Neither use-case driven development nor aspect-oriented programming is a silver bullet. They merely represent two best practices. However, I believe that integrating them will dramatically improve the way software is developed.

Ivar Jacobson

A orientação a aspectos é um paradigma emergente que visa prover a separação de preocupações transversais (ou aspectos) nos artefatos de software para obter benefícios como a melhora da reusabilidade, manutenibilidade e compreensibilidade desses artefatos.

Atualmente, o engenheiro de software tem disponíveis linguagens relativamente consolidadas, como AspectJ [KICZALES et al., 2001], para implementar um sistema utilizando fundamentos da orientação a aspectos. Porém, o mesmo não pode ser dito em relação a métodos, processos e técnicas para serem usados nas demais atividades de desenvolvimento, como requisitos, análise e projeto.

Em consequência disso, os fundamentos do paradigma orientado a aspectos vêm sendo utilizados quase que exclusivamente ao codificar o sistema, fazendo com que os benefícios da utilização desse paradigma fiquem restritos aos artefatos de implementação. Além disso, quando somente os artefatos de implementação refletem a orientação a aspectos, a coerência e o rastreamento entre esses artefatos e aqueles gerados nas atividades iniciais do ciclo de desenvolvimento é prejudicada. Uma decorrência direta

dessa impossibilidade de rastreamento entre artefatos é que o entendimento da construção de um sistema orientado a aspectos fica limitado à análise da implementação do sistema, ao invés de poder ser realizado através da análise de modelos de requisitos, de análise ou de projeto.

Nesse contexto, este capítulo apresenta uma proposta de adaptação dos fluxos de requisitos, análise e projeto do Processo Unificado para possibilitar a separação de preocupações transversais desde atividades iniciais do ciclo de desenvolvimento do sistema. Para isso, propomos a incorporação de mecanismos baseados nos fundamentos do paradigma orientado a aspectos em algumas atividades desses fluxos. Essa adaptação possibilita obter os benefícios da orientação a aspectos nos artefatos gerados ao longo desses fluxos de atividades. Além disso, permite que a identificação e modelagem de aspectos sejam realizadas não apenas na implementação, mas desde a análise dos requisitos.

4.1 Visão geral

O propósito da nossa abordagem é possibilitar a separação de preocupações transversais desde as atividades iniciais do ciclo de desenvolvimento utilizando os fundamentos do paradigma orientado a aspectos. Para isso, adaptamos algumas atividades nos fluxos de requisitos, análise e projeto do Processo Unificado (apresentadas no Capítulo 2). As adaptações propostas fornecem meios para identificação, representação e composição de preocupações transversais nas atividades desses fluxos.

A Tabela 4.1 apresenta uma visão geral do modelo conceitual que propomos na nossa abordagem para inserir os fundamentos do paradigma orientado a aspectos (vide Seção 3.3.1) nos fluxos de atividades do Processo Unificado.

Tabela 4.1- Modelo conceitual proposto para representar os fundamentos do paradigma orientado a aspectos nos fluxos de requisitos, análise e projeto do Processo Unificado

FLUXO DE ATIVIDADES	MAPEAMENTO DOS FUNDAMENTOS DE ORIENTAÇÃO A ASPECTOS					
	UNIDADES BASE	ASPECTO	RELACIONAMENTO ENTRE UM ASPECTO E UNIDADES BASE	PONTOS DE JUNÇÃO	REGRAS DE COMPOSIÇÃO	TIPO DE COMPOSIÇÃO
REQUISITOS	Casos de uso	Casos de uso, regras de negócio e restrições, denominados de requisitos aspectuais	Novo relacionamento denominado de crosscuts	São definidos em função dos eventos no cenário do caso de uso afetados pelo aspecto numa tabela de composição específica para cada requisito aspectual (tabela de composição nível 1)	São definidos através dos operadores de composição overlap, override e wrap [MOREIRA et al., 2002] numa tabela de composição específica para cada requisito aspectual (tabela de composição nível 1)	Fechada
ANÁLISE	Classes de análise	Classe de análise com estereótipo <<aspect>>, denominada de classe de análise aspectual	Novo relacionamento denominado de crosscuts	São definidos em função de mensagens trocadas entre objetos de análise numa tabela de composição específica para cada classe de análise aspectual (tabela de composição nível 2)	São definidos através dos operadores de composição overlap, override e wrap [MOREIRA et al., 2002] numa tabela de composição específica para cada classe de análise aspectual (tabela de composição nível 2)	Fechada
PROJETO	Classes de projeto	Classe de projeto com estereótipo <<aspect>>, denominada de classe de projeto aspectual	Novo relacionamento denominado de crosscuts	São definidos em função das operações de classes de projeto numa tabela de composição específica para cada classe de projeto aspectual (tabela de composição nível 3)	São definidos através dos operadores de composição overlap, override e wrap [MOREIRA et al., 2002] numa tabela de composição específica para cada classe de projeto aspectual (tabela de composição nível 3)	Fechada

Segundo nossa proposta, as unidades base (unidades que podem ser afetadas por aspectos) correspondem às unidades de decomposição comumente utilizadas em cada um dos fluxos: casos de uso no fluxo de requisitos; classes de análise no fluxo de análise; e classes de projeto no fluxo de projeto. Por sua vez, os possíveis pontos de junção referem-se a elementos nessas unidades que representam pontos de execução: no fluxo de requisitos pode-se especificá-los em função dos eventos (ou passos) nos cenários de um caso de uso; no de análise, em termos de mensagens trocadas entre objetos de análise; e no de projeto, em termos de operações de classes.

Nesse trabalho, adotamos uma prática sugerida por MURPHY et al. (2001) a qual estabelece que, em muitos casos, a melhor maneira de entender e gerenciar aspectos é representar o comportamento do aspecto usando a mesma unidade de decomposição da unidade base e utilizar uma outra abstração apenas para fazer a ligação dessas unidades. Partindo desse princípio, em nossa abordagem:

- aspectos são encapsulados nas mesmas unidades de decomposição existentes em cada um dos fluxos considerados. Porém, as unidades que encapsulam um aspecto são estereotipadas de maneira especial (ex: caso de uso aspectual, classe de análise aspectual) para indicar que o comportamento do aspecto deverá ser inserido em alguma parte das unidades que ele afeta. Denominamos de *crosscuts* o relacionamento entre uma unidade aspectual e uma unidade afetada por ela;
- informações relativas à composição entre um aspecto e a(s) unidade(s) que ele afeta são representadas numa nova abstração que criamos, denominada de tabela de composição. Para cada aspecto identificado, existirá uma tabela de composição em cada fluxo.

De maneira geral, uma tabela de composição (Figura 4.1) lista as unidades afetadas pelo aspecto e indica quais os pontos de junção e as regras de composição em cada uma dessas unidades. Essa tabela nos permite modelar a composição de aspectos de maneira fechada (Seção 3.3.1), ou seja, sem que existam referências na(s) unidade(s) base ou no aspecto sobre a relação entre eles. Esse tipo de composição permite o desacoplamento entre aspecto e a(s) unidade(s) afetada(s) por ele, possibilitando o reuso e facilitando a manutenção e compreensão tanto do aspecto como das unidades base. Conforme discutimos adiante, para cada fluxo, existirá um tipo diferente de tabela de composição que refletirá as características próprias daquele fluxo.

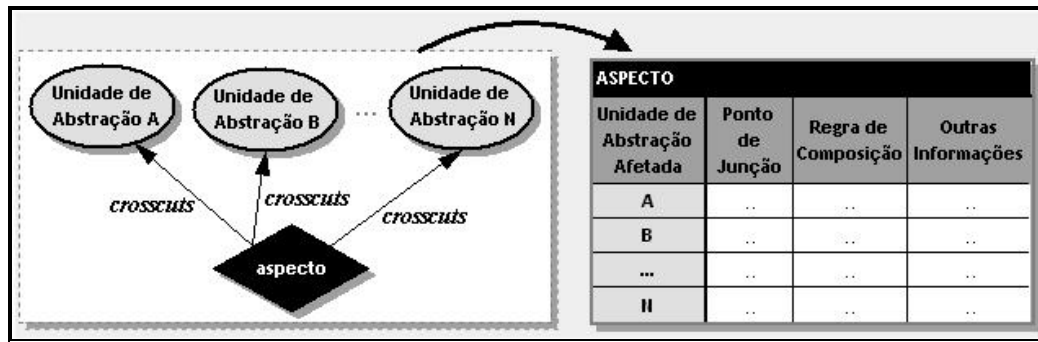


Figura 4.1- Visão geral sobre tabela de composição utilizada na nossa proposta

Neste trabalho as regras de composição serão modeladas usando os operadores de composição propostos por MOREIRA et al. (2002):

- overlap: indica que o comportamento do aspecto deve ser aplicado antes (*before*) ou depois (*after*) da execução do ponto de junção que ele afeta;
- override: indica que o comportamento do aspecto sobrepõe o ponto de junção que ele afeta. Isso significa que o comportamento do aspecto substitui o comportamento do ponto de junção;
- wrap: indica que o comportamento do aspecto encapsula o comportamento do ponto de junção que ele afeta, ou seja, indica que a maneira com que o ponto de junção será executado, ou até mesmo se ele será executado, será de responsabilidade do aspecto.

Nas seções seguintes, detalhamos e justificamos as adaptações realizadas em algumas das atividades dos fluxos de requisitos, análise e projeto do Processo Unificado.

4.2 Adaptando o fluxo de requisitos

Devido ao intrínseco relacionamento entre alguns requisitos, algumas vezes, referências a um requisito são espalhadas entre vários casos de uso e a especificação de um caso de uso possui referências a outros requisitos que não dizem respeito ao seu objetivo. Por exemplo, regras de negócio e restrições operacionais são comumente espalhadas na especificação de vários casos de uso. Esse fato torna difícil não apenas a evolução e manutenção dos casos de uso, mas também seu reuso e entendimento [BANIASSAD et al., 2002]. Requisitos que provocam esse espalhamento e

entrelaçamento são denominados de requisitos transversais ou requisitos aspectuais [ELRAD et al., 2001a; RASHID et. al, 2003].

Os fundamentos do paradigma orientado a aspectos podem ser usados para separar requisitos aspectuais da especificação dos casos de uso que eles afetam. Desta maneira, nossa proposta adaptou algumas atividades do fluxo de requisitos do Processo Unificado para possibilitar essa separação. A Figura 4.2 representa o fluxo de atividades de requisitos da nossa abordagem, destacando as adaptações propostas.

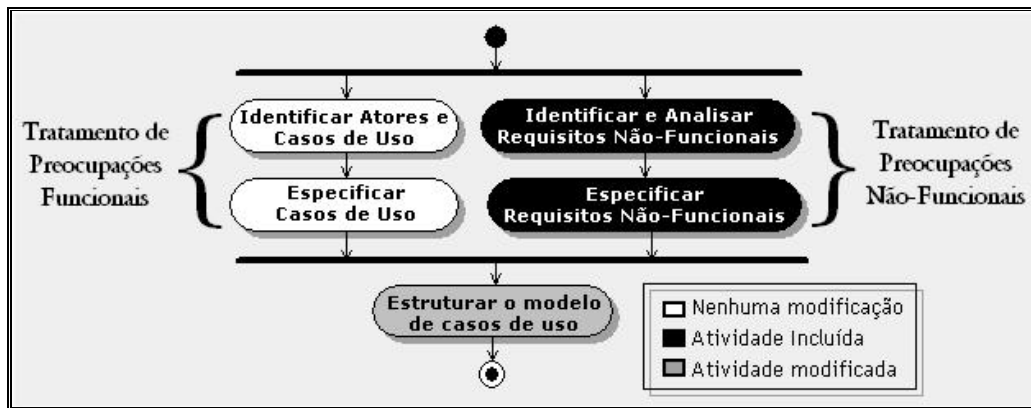


Figura 4.2 - Atividades de requisitos da nossa proposta

As atividades para o tratamento de preocupações funcionais “Identificar atores e casos de uso” e “Especificar casos de uso” não foram modificadas. Já que o tratamento sistemático de preocupações não-funcionais não é realizado no fluxo de requisitos do Processo Unificado (vide Seção 2.5), a primeira adaptação realizada nesse fluxo foi a inclusão de atividades voltadas especificamente para esse fim. A separação entre as atividades para o tratamento de preocupações funcionais daquelas para o tratamento de preocupações não-funcionais foi proposital e é justificada pelo fato desses tipos de preocupações terem características e métodos de tratamento diferentes [LOUCOPOULOS & KARAKOSTAS, 1995]. O relacionamento dessas preocupações será modelado através de tabelas de composição na atividade “Estruturar o modelo de casos de uso” (Seção 4.2.3). Essa abordagem foi inicialmente proposta em [SOUSA & CASTRO, 2003].

Nas subseções seguintes, detalhamos e justificamos cada uma das adaptações realizadas no fluxo de atividades de requisitos.

4.2.1 Identificar e analisar requisitos não-funcionais

Preocupações não-funcionais são geralmente transversais [RASHID et. al, 2002; RASHID et. al, 2003], por isso, seu tratamento adequado é um passo importante no desenvolvimento de software orientado a aspectos. Entretanto, as atividades de requisitos do Processo Unificado são focadas na funcionalidade do sistema, enquanto requisitos não-funcionais são superficialmente levados em consideração (Seção 2.5).

Portanto, decidimos incluir nas atividades de requisitos uma abordagem sistemática para analisar preocupações não-funcionais: o *framework* NFR [MYLOPOULOS et al., 1992; CHUNG et al., 2000; MYLOPOULOS et al., 2001]. Esse é um dos métodos mais completos e bem aceitos para modelar requisitos não-funcionais [CYSNEIROS & LEITE, 1999].

Algumas das características que nos motivaram a escolher o *framework* NFR para analisar preocupações não-funcionais são: (ii) ele fornece meios para que preocupações não-funcionais abstratas e subjetivas (ex: segurança, usabilidade) sejam decompostas em preocupações mais específicas e, por fim, operacionalizadas, ou seja, concretizadas em restrições operacionais ou restrições de projeto que juntos contribuem para satisfazer essas preocupações; (iii) ele dá suporte à modelagem de possíveis interações, positivas ou negativas, entre preocupações não-funcionais; e (iv) ele oferece catálogos que ajudam a refinar preocupações não-funcionais e identificar interdependências implícitas entre essas preocupações.

Além disso, conforme demonstramos em [SOUSA et al., 2003], a utilização do *framework* NFR no contexto de Engenharia de Requisitos Orientada a Aspectos traz vantagens na composição de aspectos não-funcionais quando comparada com algumas propostas atuais nessa área [RASHID et. al, 2002; ARAUJO et al., 2002; RASHID et. al, 2003; BRITO & MOREIRA, 2003] que manipulam com preocupações não-funcionais definidas de maneira abstrata, vaga e informal (ex.: segurança, disponibilidade, tempo de resposta, compatibilidade, etc). O processo de refinamento realizado no *framework* NFR provê interpretações mais específicas para essas declarações abstratas [CHUNG et al., 2000]. Ao fazer a composição de preocupações não-funcionais em termos de operacionalizações dessas preocupações, ao invés de

declarações abstratas, vagas e subjetivas, retratamos claramente como essas preocupações serão tratadas no sistema, o que nos possibilita oferecer melhores informações para modelagem de aspectos nos estágios seguintes de desenvolvimento.

Por exemplo, considerando um sistema bancário, nas propostas de ARAUJO et al. (2002) e BRITO & MOREIRA (2003) (apresentadas na Seção 3.4.1) o máximo que se poderia conseguir na etapa de composição de preocupações era definir que a preocupação de segurança deve ser aplicada nos casos de usos referentes à transações bancárias (exemplo na Figura 4.3a). No entanto, essa informação é vaga, já que podem haver diversos mecanismos e restrições de projeto para satisfazer esse tipo de preocupação, desde verificação de senha, até reconhecimento de voz, utilização de *firewall*, etc. Além disso, o tratamento de segurança nem sempre é uniforme, por exemplo, transações de consulta podem exigir mecanismos de tratamento mais simples de segurança enquanto transações financeiras utilizam mecanismos mais rígidos. Nesse mesmo exemplo, nossa abordagem poderia oferecer informações mais detalhadas sobre como a preocupação de segurança iria afetar as transações bancárias, por exemplo, poderíamos definir que os requisitos de segurança “criptografar dados” e “solicitar senha” afetariam todas as transações, enquanto requisitos de segurança “validar dados pessoais” e “verificar valor limite para transações” afetariam apenas as transações financeiras (Figura 4.3b).

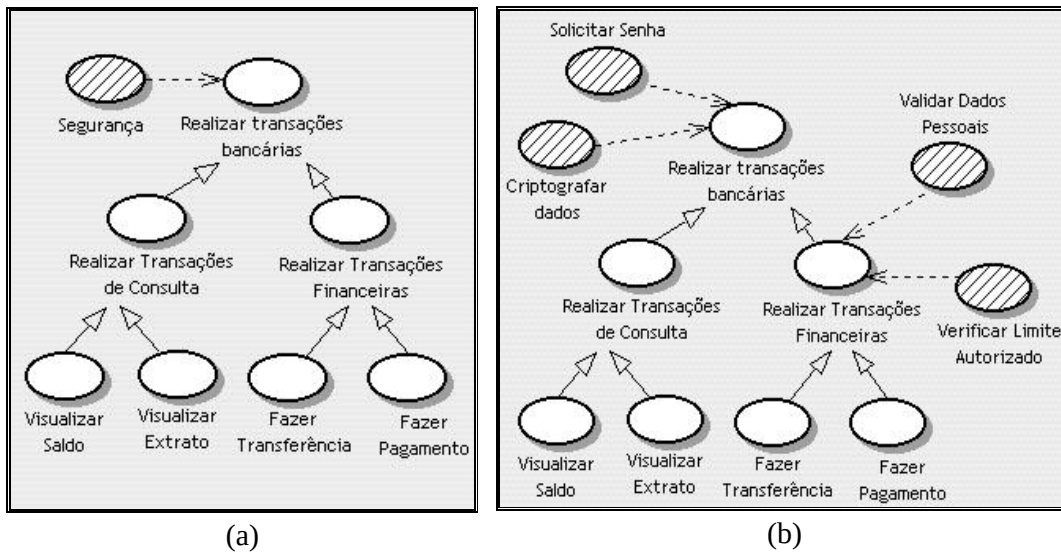


Figura 4.3 Comparação entre composição de preocupações não-funcionais nas abordagens de AORE (a) e na nossa proposta (b)

A Figura 4.4 apresenta uma visão geral das atividades do *framework* NFR. A princípio, cada preocupação não-funcional deve ser iterativamente decomposta em preocupações mais específicas. Quando uma preocupação não-funcional já tiver sido suficientemente refinada, será possível operacionalizá-la, ou seja, fornecer mecanismos mais concretos e precisos (ex: restrições operacionais, processos e restrições de projeto) que contribuirão para satisfação da preocupação. Essas decomposições podem ser realizadas de duas maneiras: (i) utilizando os operadores lógicos AND/OR; e (ii) indicando níveis de contribuição: fortemente negativa (“--”); fortemente positiva (“++”); parcialmente negativa (“-”); ou parcialmente positiva (“+”). Durante os passos de refinamento e operacionalização, contribuições e possíveis conflitos entre preocupações não-funcionais podem ser estabelecidos. Também é possível indicar preocupações prioritárias usando os símbolos “!”. No *framework* NFR, preocupações não-funcionais, suas interdependências e operacionalizações são graficamente representadas num grafo, denominado de SIG (*Softgoal Interdependency Graph*). Quando as possíveis soluções para satisfazer a preocupação não-funcional já estiverem consideradas, deve-se selecionar entre as operacionalizações: aceitando (✓) ou rejeitando (✗) cada uma delas. Por fim, identificam-se quais requisitos funcionais serão afetados pelas operacionalizações.

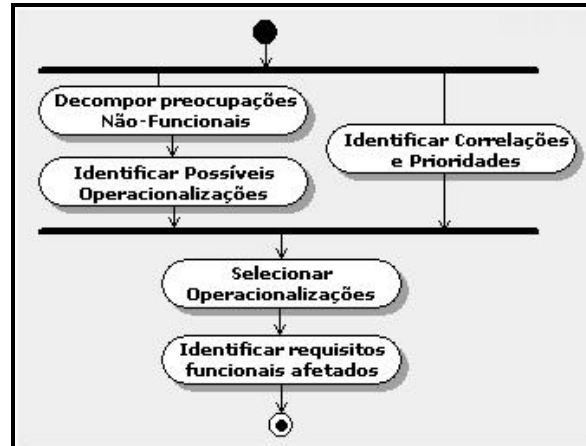


Figura 4.4 - Visão geral das atividades do *framework* NFR

Caso uma operacionalização seja específica de um caso de uso particular e altere o seu fluxo de execução, ela pode ser inserida na especificação desse caso de uso. Por outro lado, se ela deve (ou pode futuramente) alterar o fluxo de diversos casos de uso, então, ela deve ser relacionada aos casos de uso que ela afeta na atividade “Estruturar o modelo de casos de uso” (Seção 4.2.3).

4.2.2 Especificar requisitos não-funcionais

Nesse ponto, preocupações não-funcionais foram decompostas e resultaram em operacionalizações selecionadas, ou seja, restrições operacionais, dados ou restrições de projeto que contribuem para satisfação dessa preocupação. A especificação dessas operacionalizações será de acordo com o seu tipo:

- restrição operacional: deverá ser especificada de maneira semelhante a um caso de uso;
- restrição de projeto: deverá ser especificada numa seção especial do documento de requisitos relativa às restrições de projeto relacionadas a preocupações não-funcionais (fora do escopo desse trabalho).

4.2.3 Estruturar o modelo de casos de uso

Conforme anteriormente ilustrado na Figura 4.2, antes desta atividade ocorrer, as seguintes atividades já devem ter sido realizadas:

- identificação de atores e casos de uso (vide Seção 2.2.1);

- descrição detalhada de cada caso de uso (vide Seção 2.2.2); e
- identificação, análise e especificação de requisitos não-funcionais (vide Seção 4.2.1).

Tendo como entrada os artefatos gerados previamente, nesta atividade deve-se reestruturar o conjunto de casos de uso a fim de tornar o modelo de requisitos mais fácil de reusar, manter, entender e manipular nas atividades posteriores.

É a partir desta atividade que nossa abordagem começa a dar suporte a separação de preocupações transversais, fornecendo o seguinte mecanismo de estruturação para modelar o impacto de requisitos aspectuais: (a) o relacionamento entre um requisito aspectual e os casos de uso que ele afeta será representado por um novo tipo de relacionamento, denominado de *crosscuts* (Seção 4.2.3.1); e (b) informações sobre onde e como um requisitos aspectual deve ser aplicado nos casos de uso que ele afeta serão descritas num artefato à parte, denominado de tabela de composição nível 1 (Seção 4.2.3.2).

4.2.3.1 RELACIONAMENTO *CROSSCUTS*

A associação entre um requisito aspectual e os casos de uso onde esse requisito deve ser considerado é representada em nosso trabalho através de um relacionamento, que denominamos de *crosscuts*. Nos diagramas de caso de uso representamos requisitos aspectuais utilizando losangos.

Conforme descrito na Seção 3.3.2, ainda não existem meios sistemáticos de identificar aspectos: essa atividade é largamente baseada na experiência e intuição do desenvolvedor. Tendo como base o conceito de aspectos utilizado nesse trabalho (vide Seção 3.3.1) e o princípio estabelecido por FILMAN (2001) — o qual estabelece que para desenvolver uma unidade base não é obrigatoriamente necessário ter conhecimento das extensões que serão providas pelo aspecto — neste trabalho propomos algumas heurísticas para identificar um requisito aspectual:

- H1) ele representa uma preocupação que deve ser inserida num caso de uso base, alterando o fluxo de execução desse caso de uso, mas cujo propósito **não** está diretamente associado ao objetivo do caso de uso base;

H2) ele pode ser desvinculado do caso de uso base sem que isso impeça a realização do objetivo desse caso de uso; e

H3) ele é (ou pode vir a ser) aplicado em vários casos de uso e não apenas num caso de uso específico.

O comportamento de um requisito aspectual costuma ficar espalhado em especificações de vários casos de uso e misturado com o propósito dos casos de uso nos quais eles são inseridos. Por isso, é possível identificar requisitos aspectuais analisando as especificações dos casos de uso e identificando fragmentos dessas especificações que satisfazem às heurísticas apresentadas acima. Nesse caso, o comportamento do requisito aspectual deve ser extraído dos casos de uso e considerado como uma entidade (caso de uso) à parte. Outra maneira de identificar requisitos aspectuais é verificando regras de negócio e restrições não-funcionais que alteram o fluxo de execução dos casos de uso e satisfazem às heurísticas propostas.

Como mencionado inicialmente nesta seção, além dos relacionamentos do tipo *include*, *extend* e *generalization*, em nossa proposta existe um outro tipo de relacionamento para estruturar o modelo de casos de uso: *crosscuts*. Na Seção 2.2.3.1, pode-se encontrar uma discussão sobre a diferença entre esses três primeiros relacionamentos. Abaixo, reforçamos a diferença entre o relacionamento do tipo *crosscuts* e os relacionamentos do tipo *include* e *extend*, descrevendo algumas diretrizes que utilizamos para decidir qual desses relacionamentos é mais adequado ao estruturar dois casos de uso quaisquer (por exemplo: caso de uso A e caso de uso B):

- se a execução do caso de uso B é uma **parte essencial** para realizar o propósito principal do caso de uso A (i.e o caso de uso A depende do caso de uso B para satisfazer o seu objetivo), então o caso de uso A inclui o caso de uso B (i.e o caso de uso A tem um relacionamento do tipo ***include*** com o caso de uso B). Por exemplo: o caso de uso “Inserir produto na cesta de compras” inclui o caso de uso “Procurar produto”;
- se a execução do caso de uso B representa um fluxo **complexo e alternativo** que é **específico** do caso de uso A, então o caso de uso B estende o caso de uso A (i.e. o caso de uso B tem um relacionamento do tipo *extend* com o caso de uso A). Por exemplo: o caso de uso “Solicitar embalagem para

presente” tem um relacionamento do tipo *extend* com o caso de uso “Efetivar compra”, pois somente em algumas situações será necessário embalar o presente;

- se a execução do caso de uso B representa um fluxo que precisa ser aplicado no caso de uso A, mas: (i) o caso de uso A **não depende** da execução do caso de uso B para realizar seu objetivo principal; e (ii) o caso de uso B **não é um fluxo específico** do caso de uso A, podendo ser aplicado em outros casos de uso; então, o caso de uso B é transversal ao caso de uso A (i.e. o caso de uso B tem um relacionamento do tipo *crosscuts* com o caso de uso A). Por exemplo, o caso de uso “Verificar permissão de acesso” tem um relacionamento do tipo *crosscuts* com o caso de uso “Alterar informações do produto”.

EXEMPLO DE REQUISITO ASPECTUAL: VERIFICAÇÃO DE DÉBITO

Um requisito bastante comum em sistemas acadêmicos de instituições particulares é a verificação de débito. Esse requisito estabelece que alguns serviços para o aluno (tais como a emissão de histórico, a realização de matrícula em disciplina e o trancamento de matrícula em disciplina) são condicionados à inexistência de mensalidades atrasadas e de pendências na biblioteca para esse aluno. Usando as abordagens atuais, haveria uma referência a essa verificação na especificação dos casos de uso *Emitir histórico*, *Fazer matrícula em disciplina* e *Trancar matrícula em disciplina*, e na especificação de todos os casos de uso nos quais a verificação de débito precisasse ser realizada (exemplo na Figura 4.5). Sendo assim, o requisito *Verificação de débito* satisfaz as três heurísticas apresentadas acima: (i) altera o fluxo de execução dos casos de uso *Emitir histórico*, *Fazer matrícula em disciplina* e *Trancar matrícula em disciplina*, mas seu propósito não está diretamente associado ao objetivo desses casos de uso, pois não necessariamente para realizar esses casos de uso em qualquer sistema acadêmico é obrigatório realizar a verificação de débito; (ii) pode ser desvinculado desses casos de uso sem prejudicar a realização do objetivo deles; e (iii) é aplicado em vários casos de uso. Por possuir tais características, o requisito de verificação de débito se adequa a definição de requisito aspectual que apresentamos.

1	Usuário informa sua identificação do aluno.
2	O sistema identifica o aluno e recupera o registro do aluno.
3	Se não houver mensalidades atrasadas e pendências na biblioteca para o aluno , o sistema recupera registro de disciplinas oferecidas no semestre para o curso do aluno.
4	O sistema apresenta a lista de disciplinas para o aluno.
5	O aluno escolhe as disciplinas que deseja se matricular.
6	O sistema verifica se os pré-requisitos das disciplinas selecionadas já foram cursados pelo aluno.
7	O sistema registra a matrícula do aluno.

Figura 4.5 - Especificação do caso de uso *Fazer matrícula em disciplina*

Utilizando o tipo de especificação apresentado na Figura 4.5 (ver passo em destaque), se a verificação do débito deixasse de ser realizada ou se os casos de uso *Emitir histórico*, *Fazer matrícula em disciplina* e *Trancar matrícula em disciplina* precisassem ser reusados num outro sistema acadêmico que não fizesse verificação de débito em alguma das situações apresentadas, a especificação desses casos de uso teria de ser alterada. Portanto esse tipo de especificação prejudica o reuso e a manutenção dos artefatos.

A Figura 4.6 ilustra como seria representado na nossa abordagem o relacionamento entre o requisito aspectual *Verificação de débito* e os casos de uso afetados por ele. É importante frisar que, utilizando nossa abordagem, as referências à verificação de débito seriam retiradas da especificação dos casos de uso nos quais ela deve ser realizada. Como descrevemos na seção seguinte, informações sobre onde e como um requisito aspectual deve ser aplicado serão descritas num artefato à parte, denominado de tabela de composição nível 1.

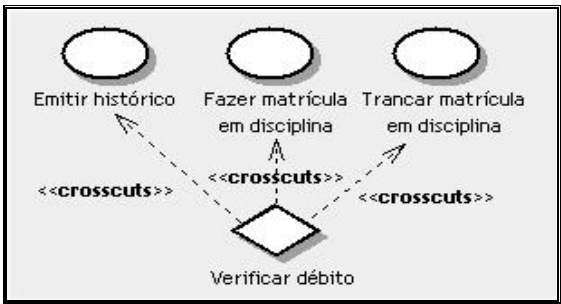


Figura 4.6 – Exemplo de relacionamento do tipo *crosscuts*

A partir dessa seção, utilizaremos esse exemplo para ilustrar nossa abordagem. No entanto, por simplificação, focaremos apenas na modelagem da composição do requisito aspectual *Verificação de débito* com o caso de uso *Fazer matrícula em*

disciplina. A modelagem da composição desse requisito com os demais casos de uso deverá ser feita de maneira semelhante.

4.2.3.2 PONTOS DE JUNÇÃO EM TABELA DE COMPOSIÇÃO

Na nossa proposta, informações a respeito da composição entre um requisito aspectual e os casos de uso afetados por ele devem ser definidas num artefato à parte, que denominamos de tabela de composição nível 1 (Tabela 4.2). Esta tabela deve ser especificada para cada requisito aspectual de acordo com o roteiro apresentado na Figura 4.7.

Tabela 4.2 - Tabela de composição para um requisito aspectual (tabela de composição nível 1)

REQUISITO ASPECTUAL: #ID <NOME>				
CASO DE USO AFETADO	CONDIÇÃO (OPCIONAL)	REGRA DE COMPOSIÇÃO	PONTO DE JUNÇÃO	INFORMAÇÕES ADICIONAIS
#ID <nome>	Condição da extensão	{overlap.after overlap.before override wrap}	Passo do cenário	Informações relativas à composição

1. Para cada ponto que um requisito aspectual precisar afetar num caso de uso, haverá uma entrada na tabela de composição nível 1 cujas colunas devem ser preenchidas da seguinte maneira:
 - 1.1. **CASO DE USO AFETADO**: preencha com o nome do caso de uso no qual o comportamento do requisito aspectual deve ser inserido.
 - 1.2. **CONDIÇÃO (OPCIONAL)**: preencha essa coluna caso haja alguma condição que deve ser satisfeita para que o comportamento do requisito aspectual seja inserido no caso de uso.
 - 1.3. **REGRA DE COMPOSIÇÃO**: preencha com algum dos seguintes operadores de composição [MOREIRA et al., 2002]: *overlap (before ou after)*, *override* e *wrap*. A escolha entre esses operadores dependerá de como o comportamento do requisito aspectual deve ser aplicado no ponto de junção (ver descrição dos operadores na Seção 4.1).
 - 1.4. **PONTO DE JUNÇÃO**: preencha com o passo do cenário do caso de uso no qual o comportamento do requisito aspectual deve ser aplicado.
 - 1.5. **INFORMAÇÕES ADICIONAIS (OPCIONAL)**: preencha com alguma outra informação que deseje acrescentar sobre a composição do requisito aspectual.

Figura 4.7 - Roteiro para o preenchimento da tabela de composição nível 1

Utilizando o exemplo do requisito aspectual *Verificação de débito* (pg. 81), sua tabela de composição nível 1 seria preenchida conforme ilustra a Tabela 4.3. Na primeira coluna seriam listados os casos de uso *Emitir Histórico*, *Fazer matrícula em disciplina* e *Trancar matrícula em disciplina*. Nas demais colunas deveriam ser

especificadas informações sobre a composição do requisito aspectual com cada um desses casos de uso. Por exemplo, para o caso de uso *Fazer matrícula em disciplina*, a regra de composição definida foi *wrap* visto que a execução do ponto de junção nesse caso de uso dependerá do resultado obtido na verificação de débito. O ponto de junção no qual o aspecto irá atuar foi o passo 3 desse caso de uso (vide Figura 4.5): “O sistema recupera registro de disciplinas oferecidas no semestre para o curso do aluno”. Por fim, na última coluna especificamos informações adicionais sobre a composição, indicando que o ponto de junção só será executado caso não haja débito para o aluno. O preenchimento das colunas para os casos de uso *Emitir histórico* e *Trancar matrícula em disciplina* seria semelhante.

Tabela 4.3 - Exemplo de especificação da tabela de composição nível 1 para requisito aspectual
Verificação de débito

REQUISITO ASPECTUAL: AR#01 VERIFICACAO DE DEBITO				
CASO DE USO AFETADO	COND.	REGRA DE COMPOSIÇÃO	PONTO DE JUNÇÃO	INFORMAÇÕES ADICIONAIS
UC#04: Emitir Histórico	...	...	...	
UC#05: Fazer matrícula em disciplina	-	Wrap	3	Se semDebito então execute(ponto junção) Senão exibe(msgErro)
UC#06: Trancar matrícula em disciplina	...	...	...	...

4.2.3.3 CONFLITOS ENTRE REQUISITOS ASPECTUAIS

Quando diferentes requisitos aspectuais devem ser aplicados num mesmo ponto de junção de um caso de uso com operadores de composição diferentes, a ordem de execução dos aspectos deve seguir a ordem de precedência dos seus respectivos operadores de composição. Entretanto, se dois ou mais aspectos devem ser aplicados num mesmo ponto de junção de um caso de uso com o mesmo operador de composição, então existe um conflito, e a ordem de precedência deve ser decidida pelo analista e definida numa tabela de conflitos (Tabela 4.4).

Tabela 4.4 - Tabela de conflitos entre requisitos aspectuais

	AR#01	AR#02	...	AR#n
UC #01				
UC #02				
...				
UC #m				

Existirá apenas uma tabela de conflitos para todo o sistema, na qual todos os casos de uso são listados nas linhas e os requisitos aspectuais nas colunas. Caso exista um conflito de um requisito aspectual num passo de um caso de uso, a ordem de precedência desse aspecto em relação aos aspectos com que ele conflita deve ser especificada na célula de interseção entre o caso de uso e o aspecto da seguinte forma: $P[numPassoConfl] - [siglaOperador] = [ordemPreced]$, onde:

- `numPassoConfl` deve ser substituído pelo número do passo do caso de uso onde existe o conflito;
- `siglaOperador` deve ser substituído por: (i) “OB”, caso o conflito ocorra com o operador *overlap.before*; (ii) “OA”, caso o conflito ocorra com o operador *overlap.after*; (iii) “O”, caso o conflito ocorra com o operador *override*; ou “W”, caso o conflito ocorra com o operador *wrap*; e
- `ordemPreced` deve ser substituído pela ordem de precedência do requisito aspectual.

Supondo que, além do requisito aspectual *AR#01- Verificacao de débito*, um outro requisito aspectual cujo identificador é *AR#03* precisasse ser aplicado no passo 3 do caso de uso *UC#05 - Fazer matrícula em disciplina* com o operador *wrap*; e que a prioridade de execução fosse concedida para o requisito *AR#03*. Então, a tabela de conflitos deveria ser preenchida conforme apresentado na Tabela 4.5

Tabela 4.5 – Exemplo de preenchimento da tabela de conflitos

	AR#01	AR#02	AR#03	AR#04
UC #01				
UC #02				
UC #03				
UC #04				
UC #05	P3-W = 2º		P3-W = 1º	

À medida que os pontos de junção vão sendo refinados nos fluxos seguintes, esses conflitos podem deixar de existir. Contudo, essas prioridades definidas na tabela de conflitos serão úteis não somente caso o conflito persista nos fluxos de análise e projeto, mas também para auxiliar na definição dos pontos de junção nesses fluxos.

4.2.3.4 CONSIDERAÇÕES SOBRE O PORQUÊ DE NÃO UTILIZAR O MECANISMO DE EXTENSÃO PARA MODELAR ASPECTOS

Jacobson [JACOBSON, 2003a ; JACOBSON, 2003b] afirma que o mecanismo de extensão de casos de uso¹¹ (Seção 3.4.1), poderia ser usado para modelar aspectos nas atividades de requisitos: um caso de uso de extensão seria equivalente a um aspecto e pontos de extensão seriam equivalentes a pontos de junção (detalhes na Seção 3.4.1.2). Contudo, analisamos que esse mecanismo não seria ideal para modelar aspectos pelas seguintes razões:

- i) o mecanismo de extensão pode ser usado em situações nas quais o caso de uso de extensão não corresponde a um requisito aspectual no sistema. Por exemplo, quando o caso de uso de extensão representa um fluxo complexo, **específico do caso de uso base**, que raramente ocorre ou que é alternativo e executado apenas em certos casos. Por exemplo, na Figura 4.8 e na Figura 4.9, os casos de uso de extensão *Fazer chamada em conferência* e *Item ficar preso* são específicos do caso de uso base e, por isso, não são requisitos aspectuais; e



Figura 4.8 – Exemplo de caso de uso de extensão (adaptado de [RATIONAL, 2002])



Figura 4.9 – Exemplo de caso de uso de extensão (adaptado de [JACOBSON et al., 1992])

- ii) o modo como pontos de extensão são definidos prejudica o reuso e a manutenibilidade do caso de uso base e do caso de uso de extensão. Isso ocorre porque a UML [BOOCH et al., 1999] sugere que seja especificado um rótulo para cada ponto de extensão no caso de uso base e que os casos de uso de extensão referenciem os rótulos dos pontos de extensão nos quais eles devem ser aplicados. Por exemplo, se fóssemos modelar o requisito aspectual “Verificar débito” e sua aplicação nos casos de uso “Emitir Histórico”, “Fazer matrícula em disciplina” e “Trancar matrícula em disciplina” utilizando o

mecanismo de extensão, esses casos de uso seriam especificados, de acordo com a UML, assim como ilustrado na Tabela 4.6 e na Tabela 4.7. Usando esse modo de especificação, ocorreriam problemas de reusabilidade e manutenibilidade tanto nos casos de uso base como nos casos de uso de extensão, por exemplo, nas seguintes situações: (1) se a verificação do débito deixasse de ser realizada; (2) se esses casos de uso precisassem ser reusados num outro sistema acadêmico que não fizesse verificação de débito em alguma das situações apresentadas; ou (3) se a verificação de débito precisasse ser aplicada em outro caso de uso. Além disso, ainda existe uma incoerência no modo como os pontos de extensão são especificados em UML [COCKBURN, 2000]: apesar do propósito original do *extend* ser evitar que o caso de uso base seja modificado devido a extensões, sempre que houver uma nova extensão no caso de uso base, ele deverá ser alterado para incluir o rótulo indicando onde nova extensão deverá ocorrer.

Tabela 4.6 - Exemplo de especificação do caso de uso base segundo a UML

UC #04: EMITIR HISTÓRICO	UC #05: FAZER MATRÍCULA EM DISCIPLINA	UC #06: TRANCAR MATRÍCULA EM DISCIPLINA
1 ...	1 ...	1 ...
2 ...	2 ...	2 ...
... ..		
m ...	n ...	P ...
PONTOS DE EXTENSÃO	PONTOS DE EXTENSÃO	PONTOS DE EXTENSÃO
<u>Débito</u> : ocorre depois do passo 1	<u>Pendência</u> : ocorre antes do passo 3	<u>DébitoAluno</u> : ocorre depois do passo 2

Tabela 4.7 - Exemplo de especificação do caso de uso de extensão segundo a UML

CASO DE USO # 07: VERIFICAR DÉBITO
Esse caso de uso estende o caso de uso <u>emitir histórico</u> . Ele é inserido no ponto de extensão <u>Débito</u>
Esse caso de uso estende o caso de uso <u>fazer matrícula em disciplina</u> . Ele é inserido no ponto de extensão <u>Pendência</u>
Esse caso de uso estende o caso de uso <u>trancar matrícula em disciplina</u> . Ele é inserido no ponto de extensão <u>DébitoAluno</u> .
1 ...
2 ...
3 ...

¹¹ O mecanismo de extensão significa o mecanismo usado para modelar um relacionamento do tipo *extend* entre casos de uso.

Pelos motivos acima expostos, na nossa abordagem, o relacionamento do tipo *extend* é utilizado somente nas situações onde o caso de uso de extensão é um fluxo complexo e alternativo (ou que raramente ocorre) **específico** do caso de uso base (exemplo na Figura 4.8 e na Figura 4.9). Nas situações em que um caso de uso de extensão satisfaz às heurísticas que utilizamos para identificar um requisito aspectual (Seção 4.2.3.1), utilizamos o relacionamento do tipo *crosscuts*. Num relacionamento do tipo *crosscuts*, nenhum dos casos de uso envolvidos farão referências à extensão provida pelo aspecto, essas informações serão colocadas somente na tabela de composição, facilitando o reuso e a manutenção dos artefatos envolvidos na composição.

4.3 Adaptando o fluxo de análise

Para que a separação de preocupações transversais obtida nos artefatos de requisitos fosse mantida nos artefatos gerados no fluxo de análise, foram propostas algumas adaptações nas atividades de “Analisar casos de uso” (Seção 4.3.1) e “Analisar classes” (Seção 4.3.2).

As adaptações propostas seguem a mesma linha de raciocínio sugerida no fluxo de requisitos: preocupações transversais serão modeladas em entidades à parte e o modo como essas preocupações interferem na interação entre entidades de análise deve ser modelado em tabelas de composição. Na sequência, descrevemos e justificamos cada uma dessas adaptações.

4.3.1 Analisar casos de uso

Nessa atividade, foram realizadas modificações tanto na parte responsável pela identificação de classes de análise (Seção 4.3.1.1) como na parte responsável por descrever interações entre objetos de análise (Seção 4.3.1.2), conforme apresentamos a seguir.

4.3.1.1 IDENTIFICAR CLASSES DE ANÁLISE

Nessa atividade a funcionalidade de cada caso de uso é distribuída entre classes de análise que colaboram para realizar o caso de uso (Seção 2.3.1.1). Classes de análise possuem responsabilidades na concretização de casos de uso.

Segundo o Processo Unificado, uma classe de análise pode ser classificada de acordo com as responsabilidades que ela encapsula em um desses três tipos: fronteira () entidade () e controle (). Em resumo, uma classe do tipo fronteira encapsula responsabilidades relacionadas à comunicação entre os atores e o sistema; uma classe do tipo entidade encapsula responsabilidades relacionadas à manipulação de informações no sistema que devem ser armazenadas e mantidas mesmo após a finalização do caso de uso; e uma classe do tipo controle encapsula responsabilidades relacionadas à delegação de trabalho para outras classes e à definição da ordem entre eventos do caso de uso.

Por sua vez, para encapsular as responsabilidades provenientes de um requisito aspectual, decidimos criar um novo tipo de classe de análise denominada de classe de análise aspectual (representada pelo estereótipo <<aspect>> e simbolizada conforme a Figura 4.10).

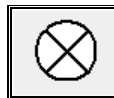


Figura 4.10 - Representação gráfica de uma classe de análise aspectual

Desta maneira, para cada requisito aspectual identificado na atividade *Estruturar o modelo de casos de uso* (Seção 4.2.3), haverá uma classe de análise aspectual para encapsular as responsabilidades desse requisito nas classes de análise que ele deve ser considerado. Por exemplo, para o requisito aspectual *Verificar débito* identificado previamente, seria criada uma classe de análise *VerificacaoDebito* com estereótipo <<aspect>> que encapsularia a responsabilidade de realizar a verificação de débito .

Algumas vezes, para realizar suas responsabilidades, a classe de análise aspectual precisa interagir com outras classes auxiliares. Por exemplo, consultando a especificação do requisito que originou a classe de análise aspectual *VerificacaoDebito* (pg. 81), verificamos que para realizar sua responsabilidade é preciso que seja feita uma consulta para identificar se existem mensalidades ou empréstimos na biblioteca com

atraso para o aluno. Então, para realizar a verificação de débito, a classe de análise aspectual *VerificacaoDebito* precisaria interagir com uma classe do tipo entidade responsável por armazenar as mensalidades dos alunos (ex: classe *MensalidadesAlunos*) e também com uma classe do tipo entidade responsável por encapsular os empréstimos na biblioteca (ex: *EmprestimosBiblioteca*). Sendo assim, o requisito aspectual *Verificacao de debito* seria mapeado nas classes de análise: *VerificacaoDebito*, *MensalidadesAlunos* e *EmprestimosBiblioteca*.

Portanto, um requisito aspectual será mapeado em uma classe de análise aspectual e, se necessário, em classes de análise auxiliares do tipo entidade, controle ou interface.

4.3.1.2 DESCREVER INTERAÇÃO ENTRE OBJETOS DE ANÁLISE

Após as classes de análise terem sido identificadas, é necessário descrever para cada caso de uso como seus objetos de análise correspondentes interagem para concretizar seu comportamento (detalhes na Seção 2.3.1.2). Geralmente, essa interação é representada em diagramas de colaboração. Para o caso de uso *Fazer matrícula em disciplina*, por exemplo, teríamos o diagrama de colaboração apresentado na Figura 4.11.

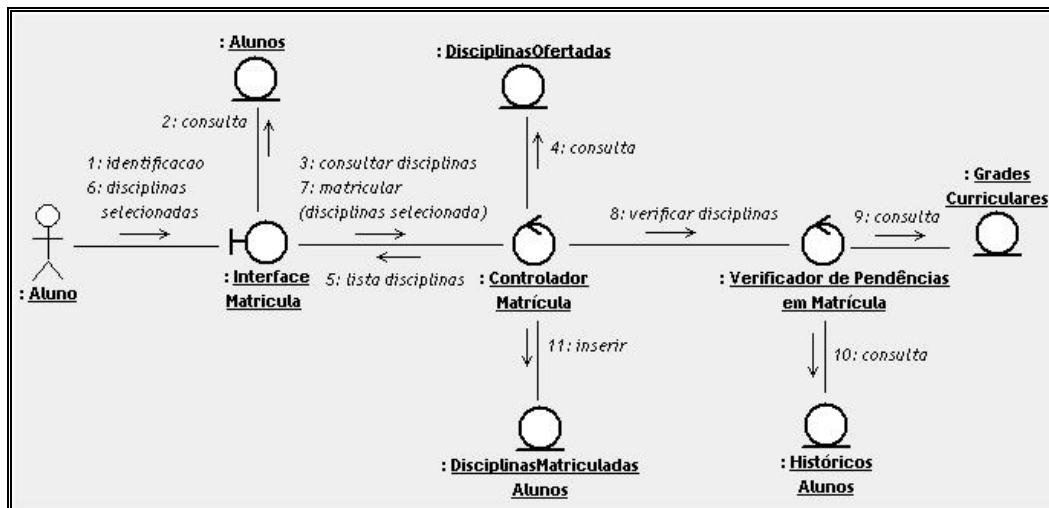


Figura 4.11 – Diagrama de colaboração para o caso de uso *Fazer matrícula em disciplina*

Conforme apresentado na seção anterior, uma classe de análise aspectual é responsável por encapsular as responsabilidades de um requisito aspectual. Estas

responsabilidades deverão ser inseridas nos casos de uso afetados pelo requisito aspectual e, conseqüentemente alterarão a interação dos objetos de análise que concretizam esses casos de uso. No entanto, um objeto de análise aspectual não deve ser colocado diretamente nos diagramas de interação nos quais ele atua para não comprometer o entendimento, reuso e manutenção dessa colaboração.

Por exemplo, na atividade anterior, identificamos a classe de análise *VerificacaoDebito* que irá alterar a colaboração entre os objetos que realizam o caso de uso *Emitir histórico*, *Fazer matrícula em disciplina* e *Trancar matrícula em disciplina*. Um impulso natural seria colocar o objeto de *VerificacaoDebito* diretamente nos diagramas de colaboração desses casos de uso. Entretanto, fazendo isso, estaríamos inserindo uma preocupação que não é própria dessas colaborações, dificultando o entendimento do propósito essencial da colaboração e prejudicando o reuso e manutenção desse artefato. À medida que o número de aspectos que devem ser aplicados na colaboração cresce, isso fica mais crítico.

Com o propósito de separar essas preocupações, a adaptação que fizemos nesta atividade foi fornecer uma tabela de composição para que se possa modelar de maneira não-invasiva (i.e. sem fazer referência explícita nos diagramas de colaboração) como um objeto de análise aspectual irá alterar a interação entre os objetos de análise dos casos de uso afetados por ele. Desta maneira, para cada classe de análise aspectual identificada, deverá haver uma tabela de composição nível 2 (Tabela 4.8). O preenchimento desta tabela deve seguir o roteiro apresentado na Figura 4.12.

Tabela 4.8 - Tabela de composição para classe de análise aspectual (tabela de composição nível 2)

CLASSE DE ANÁLISE ASPECTUAL: <NOME>						
CASO DE USO AFETADO	CLASSE DE ANÁLISE AFETADA	CONDIÇÃO (OPCIONAL)	RESPONSABILIDADE DO ASPECTO	REGRA DE COMPOSIÇÃO	PONTO DE JUNÇÃO (CLASSE//MENSAGEM)	INFORMAÇÕES ADICIONAIS
#N <nome>	<nome>	Condição da extensão	<nome da responsabilidade>	{overlap.after overlap.before override wrap}	<nome classe> / <nome da mensagem>	Informações relativas à composição

1. Para cada entrada (linha) na tabela de composição nível 1 (Tabela 4.2) do requisito que originou a classe de análise aspectual, haverá uma entrada na tabela de composição nível 2 cujas colunas devem ser preenchidas da seguinte maneira:
 - 1.1. CASO DE USO AFETADO: preencha com o mesmo conteúdo dessa coluna na tabela nível 1.
 - 1.2. CONDIÇÃO: preencha com o mesmo conteúdo dessa coluna na tabela nível 1.
 - 1.3. RESPONSABILIDADE DO ASPECTO: preencha com o papel do objeto de análise aspectual na iteração que ele afeta;
 - 1.4. REGRA DE COMPOSIÇÃO: preencha com o mesmo conteúdo dessa coluna na tabela nível 1.
 - 1.5. PONTO DE JUNÇÃO:
 - o veja qual o passo do cenário identificado como ponto de junção na tabela de composição nível 1;
 - o preencha essa coluna com a mensagem no diagrama de colaboração que corresponde ao passo do cenário identificado, juntamente com o nome da classe que recebe essa mensagem;
 - 1.6. CLASSE DE ANÁLISE AFETADA: preencha com a classe do objeto de análise onde o comportamento do requisito aspectual deve ser inserido, podendo ser a classe que envia a mensagem identificada como ponto de junção no diagrama de colaboração ou a classe que recebe essa mensagem;
 - 1.7. INFORMAÇÕES ADICIONAIS (OPCIONALMENTE): preencha com o conteúdo da mesma coluna na tabela de nível 1 e com alguma outra informação que deseje acrescentar.

Figura 4.12- Roteiro para o preenchimento da tabela de composição nível 2

Usando como exemplo a classe de análise aspectual *VerificacaoDebito* identificada na seção anterior, para preencher sua tabela de composição nível 2, o primeiro passo é recorrer as informações da tabela de composição nível 1 correspondente (Tabela 4.3). Com essas informações já é possível preencher as colunas “caso de uso afetado”, “condição”, “regra de composição” e “informações adicionais”. Verifica-se então que são afetados os casos de uso *Emitir Histórico*, *Fazer matrícula em disciplina* e *Trancar matrícula em disciplina*. Tomando como exemplo o caso de uso *Fazer matrícula em disciplina*, a partir do seu diagrama de colaboração (Figura 4.11) e do “ponto de junção” na Tabela 4.3, podemos determinar que a mensagem que deverá ser afetada pelo objeto de análise aspectual *VerificacaoDebito* é a mensagem *consulta* disparada pelo objeto de análise *ControladorMatricula* para o objeto de análise *DisciplinasOfertadas*. Logo, o conteúdo da coluna “ponto de junção” será *DisciplinasOfertadas/consulta*. Como a verificação de débito deve ocorrer quando a classe *ControladorMatricula* envia a mensagem *DisciplinasOfertadas/consulta*, definida como ponto de junção, então, o conteúdo da coluna “classe de análise afetada” será *ControladorMatricula*. A responsabilidade da classe de análise *VerificacaoDebito*

nessa colaboração é “verificar débito”. Ao final da análise de cada linha na Tabela 4.3, obteríamos a Tabela 4.9.

Tabela 4.9 – Tabela de composição para a classe de análise aspectual *VerificacaoDebito*

CLASSE DE ANÁLISE ASPECTUAL: <i>VerificacaoDebito</i>						
CASO DE USO AFETADO	CLASSE DE ANÁLISE AFETADA	COND.	RESPONSABILIDADE DO ASPECTO	REGRA DE COMPOSIÇÃO	PONTO DE JUNÇÃO (CLASSE//MENSAGEM)	INFORMAÇÕES ADICIONAIS
UC#04 - EmitirHistórico	...	...	...	...	...	...
UC#05 - Fazer matrícula disciplina	Controlador Matrícula	-	verificar débito	Wrap	<i>DisciplinasOfertadas/consulta</i>	Se semDebito então execute(pontoJuncao) Senão exibe(msgErro)
UC#06 - Trancar matrícula disciplina	...	...	...	...	...	...

A Figura 4.13 ilustra o modo como um objeto de análise aspectual interceptará o ponto de junção correspondente a mensagem `msg(param)` enviada do objeto de análise A para o objeto B, dependendo da regra de composição especificada. Caso o operador de composição seja *override*, a mensagem de A para B será substituída por uma mensagem de A para o objeto de análise aspectual. Se o operador for *overlap.before*, inicialmente é enviada uma mensagem de A para o objeto de análise aspectual realizar alguma responsabilidade e, somente depois, será enviada a mensagem de A para B; no caso de ser *overlap.after*, essa ordem é invertida. Por fim, se o operador de composição for *wrap*, ele atuará, geralmente, em duas situações: ou para condicionar o envio da mensagem de A para B a algum processamento feito pelo objeto de análise aspectual; ou para mudar o parâmetro da mensagem enviada de A para B. É importante frisar ainda que, se tivéssemos especificado que a “classe de análise afetada” seria a classe A, então, o objeto de análise aspectual iria atuar sempre que A enviasse a mensagem `msg(param)` para B; caso tivéssemos definido que a “classe afetada” seria B, então o objeto de análise aspectual iria atuar sempre que B recebesse a mensagem `msg(param)`, independente de quem enviasse a mensagem.

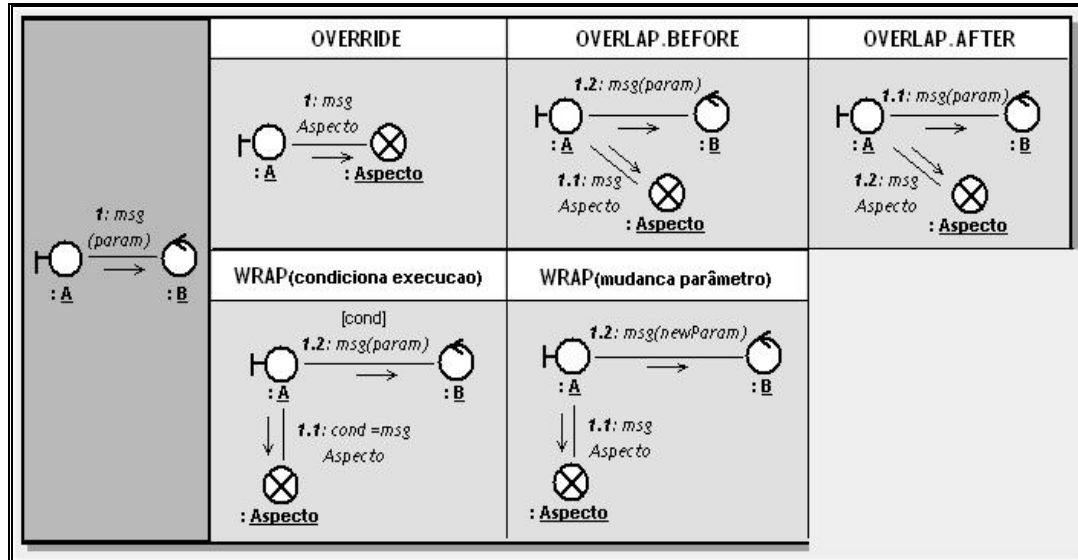


Figura 4.13 – Ilustração de como uma mensagem será interceptada, dependendo da regra de composição

A partir dos diagramas de interação dos casos de uso base e da tabela de composição nível 2, pode ser gerada uma visão desses diagramas com a presença dos objetos aspectuais que os afetarão. Essa visão é interessante para o projetista/implementador do aspecto, pois permite que ele visualize graficamente como os objetos aspectuais irão afetar a interação dos objetos de análise, mesmo antes da implementação do sistema. A Figura 4.14 apresenta o diagrama de colaboração do caso de uso *Fazer matrícula em disciplina* incluindo o objeto aspectual *VerificacaoDebito* que afetará a colaboração. Esse diagrama foi gerado a partir da Tabela 4.9 e da Figura 4.11.

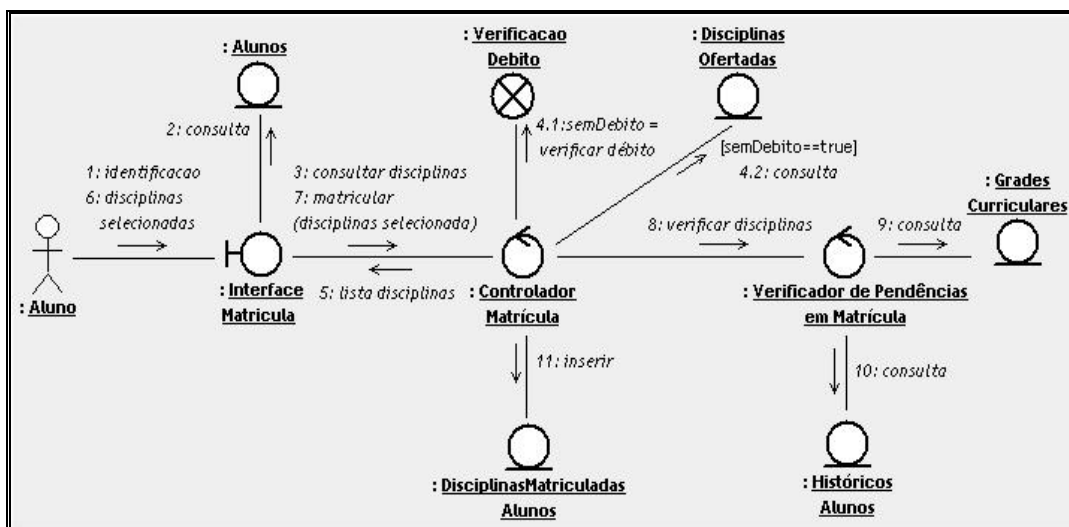


Figura 4.14 – Visão do diagrama de colaboração do caso de uso *Fazer matrícula em disciplina* COM a presença do objeto aspectual *VerificacaoDebito*

Uma alternativa que poderá ser analisada em trabalhos futuros é a utilização da abordagem e padrões de composição [CLARKE & WALKER, 2001] (Seção 3.4.2) para estabelecer diagramas de interação parametrizáveis.

4.3.2 Analisar classes

Nesta atividade as responsabilidades, os atributos e os relacionamentos de cada classe de análise são identificados e mantidos.

As responsabilidades de uma classe de análise aspectual são identificadas diretamente através da sua tabela de composição nível 2. Para a classe de análise aspectual *VerificacaoDebito*, a única responsabilidade identificada seria “verificar débito”. Se para realizar uma responsabilidade for preciso interagir com outras entidades, sugerimos que seja feito um diagrama de colaboração para representar essa interação. A Figura 4.15 representa a interação que o objeto da classe de análise *VerificacaoDebito* precisará fazer para realizar a responsabilidade de “verificar débito”.

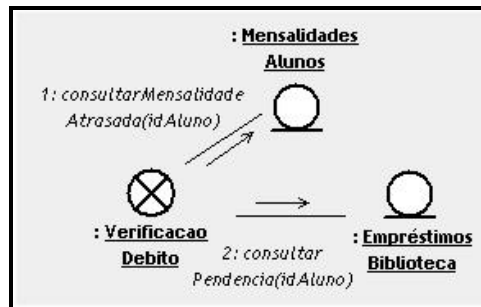


Figura 4.15 – Diagrama de colaboração para realização da responsabilidade “verificar débito”

Ligações entre objetos em diagramas de colaboração indicam a necessidade de relacionamento entre as respectivas classes. Por isso, caso desejado, é possível fazer um diagrama de classes para representar o relacionamento entre classes de análise que se comunicam em diagramas de interação. Em nossa abordagem, o relacionamento entre uma classe de análise aspectual e as classes de análise afetadas por ela será um tipo de relacionamento de associação denominado de *crosscuts*. Esse tipo de relacionamento indica que será feita uma inserção ou sobreposição do comportamento do aspecto em alguma parte da especificação da(s) unidade(s) que ele afeta. Sendo assim, a classe de análise aspectual *VerificacaoDebito* teria um relacionamento do tipo *crosscuts* com a classe de análise *ControladorMatricula*.

4.4 Adaptando o fluxo de projeto

O modelo de projeto refina o modelo de análise considerando soluções específicas da tecnologia que deverá ser utilizada para implementar o sistema. Conforme discutimos na Seção 2.4, os fluxos de análise e projeto são semelhantes, diferenciando-se basicamente no que se refere ao nível de abstração das entidades manipuladas em suas atividades. Por isso, as adaptações que realizamos nas atividades do fluxo de projeto são semelhantes às aquelas apresentadas no fluxo de análise.

Com o propósito de prover mecanismos para que a separação de preocupações transversais obtida nos fluxos anteriores pudesse ser mantida no fluxo de projeto, realizamos adaptações nas atividades “Projetar casos de uso” (Seção 4.4.1) e “Projetar classes”(Seção 4.4.2). As adaptações propostas seguem a mesma linha de raciocínio daquelas realizadas nos fluxos anteriores: preocupações transversais serão modeladas

em entidades à parte e o modo como essas preocupações interferem na interação entre entidades de projeto deve ser modelado em tabelas de composição. Na sequência, descrevemos e justificamos cada uma dessas adaptações.

4.4.1 Projetar casos de uso

Nessa atividade, foram realizadas modificações tanto na parte responsável pela identificação de elementos de projeto (Seção 4.4.1.1) como na parte responsável por descrever interações entre objetos de projeto (4.4.1.2), conforme apresentamos a seguir.

4.4.1.1 IDENTIFICAR ELEMENTOS DE PROJETO

No Processo Unificado, o objetivo desta atividade é mapear as classes de análise necessárias para realizar cada caso de uso em elementos de projeto que representam a uma abstração mais próxima de como o sistema será implementado (Seção 2.4.1.1). A Tabela 4.10 apresenta um exemplo do mapeamento entre classes de análise e entidades de projeto para o caso de uso *Fazer matrícula em disciplina*, assumindo que iremos projetar o sistema utilizando o padrão arquitetural PDC¹² [MASSONI et al., 2001] para os componentes de manipulação de dados da aplicação.

Tabela 4.10 – Mapeamento entre classes de análise necessárias para realizar o caso de uso *Fazer matrícula em disciplina*

CLASSE DE ANÁLISE	ENTIDADE DE PROJETO
-	Fachada
InterfaceMatricula	FormularioIdentificacao FormularioMatricula
ControladorMatricula	ControladorMatricula
VerificadorPendencia	
Alunos	CadastroAluno IRepositorioAlunos RepositorioAlunosBDR Aluno
DisciplinasOfertadas	CadastroDiscOfertada IRepositorioDiscOfertadas RepositorioDiscOfertadasBDR DiscOfertada

} Pacote ManipulacaoAluno

¹² O padrão arquitetural PDC distribui as classes do sistema em camadas que se comunicam através de interfaces bem definidas. Ele sugere o uso de coleções persistentes de dados e faz distinção clara entre um “dado” e uma “coleção de dados”.

Nesse ponto, a única diferença na nossa proposta ocorre no mapeamento entre classes de análise aspectuais em entidades de projeto. Uma classe de análise aspectual deverá ser mapeada numa classe de projeto aspectual.

Uma classe de projeto aspectual é uma classe que possui um estereótipo <<aspect>> para indicar que algumas de suas operações deverão ser invocadas por outras classes com base numa tabela de composição (Seção 4.4.1.2). Caso na implementação do sistema seja usada uma linguagem orientada a aspectos, a própria linguagem fornecerá mecanismos para que essa invocação de operações ocorra de maneira não-invasiva. No entanto, se a escolha for por uma implementação orientada a objetos tradicional, provavelmente essa invocação deverá ocorrer de maneira redundante em cada classe onde a operação deverá ser considerada.

Uma classe de projeto aspectual não terá, necessariamente, uma classe correspondente na implementação do sistema. Ela representa apenas uma abstração que centraliza as propriedades e operações que devem ser incorporadas no sistema devido a uma preocupação transversal particular.

Dando continuidade ao exemplo do requisito aspectual de *Verificação de débito*, no fluxo de análise identificamos as seguintes classes de análise para realizar esse requisito: *VerificacaoDebito*, *MensalidadesAlunos* e *EmprestimoBiblioteca* (Seção 4.3.1.1). Assumindo que iremos projetar o sistema utilizando o padrão arquitetural PDC [MASSONI et al., 2001], obteríamos o mapeamento apresentado na tabela Tabela 4.11.

Tabela 4.11 – Mapeamento entre classes de análise necessárias para realizar o requisito aspectual *VerificacaoDebito*

CLASSE DE ANÁLISE	ENTIDADE DE PROJETO
VerificacaoDebito (classe de análise aspectual)	VerificacaoDebito (classe de projeto aspectual)
MensalidadesAlunos	CadastroMensalidades IRepositorioMensalidades RepositorioMensalidadesBDR MensalidadeAluno
EmprestimoBiblioteca	CadastroEmprestimo IRepositorioEmprestimo RepositorioEmprestimoBDR Emprestimo

4.4.1.2 DESCREVER INTERAÇÃO ENTRE ENTIDADES DE PROJETO

A partir do mapeamento entre classes de análise em entidades de projeto, tem-se o conhecimento das classes de projeto necessárias para cada caso de uso. Nesta

atividade deve ser descrito em diagramas de seqüência como será a interação dos objetos de projeto para realizar cada caso de uso (Seção 2.4.1.2). A Figura 4.16 ilustra um diagrama de seqüência da primeira parte do caso de uso *Fazer matrícula em disciplina*, que vai desde a identificação do usuário até a exibição de disciplinas ofertadas para a matrícula.

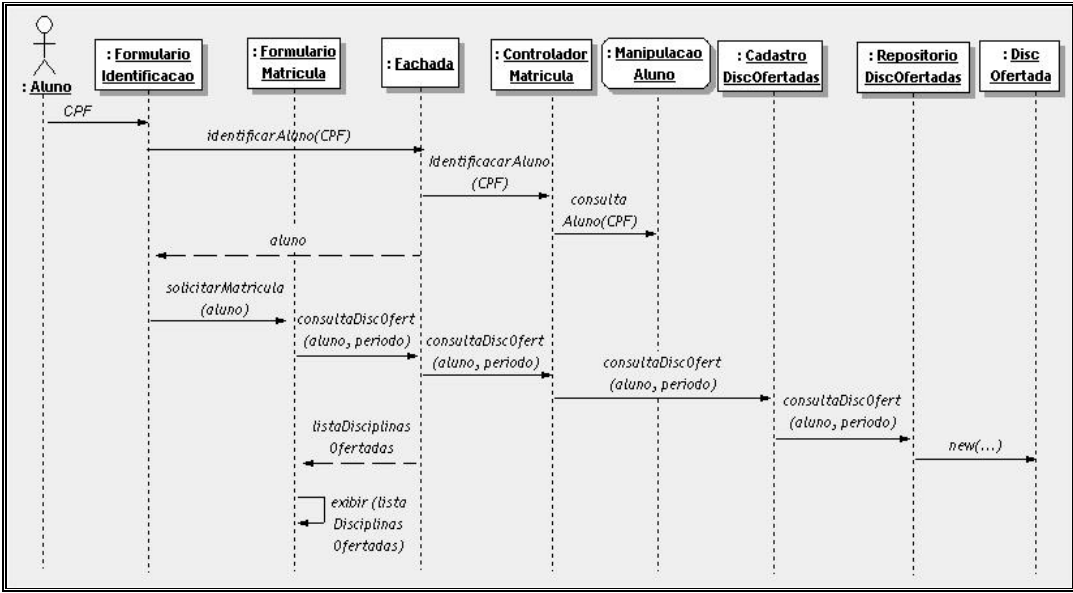


Figura 4.16 - Exemplo de diagrama de seqüência para primeira parte do caso de uso *Fazer matrícula em disciplina*

No entanto, não desejamos que o comportamento de um requisito aspectual (nessa parte do processo encapsulado numa classe de projeto aspectual), seja misturado nos diagramas de seqüência dos casos de uso nos quais eles devem ser inseridos. Assim como vem sendo feito nas atividades de requisito e análise, para evitar isso, utilizamos uma tabela de composição nível 3 (Tabela 4.12) para definir como uma classe de projeto aspectual deverá ser inserida nas interações dos casos de uso que ela afeta. Para cada classe de projeto aspectual, deverá haver uma tabela de composição nível 3.

Tabela 4.12 -Tabela de composição para classe de projeto aspectual ou tabela de composição nível 3

CLASSE DE PROJETO ASPECTUAL: <NOME>							
CASO DE USO AFETADO	CLASSE PROJETO AFETADA	CONDIÇÃO	ADIÇÕES PROVIDAS PELA CLASSE DE PROJETO ASPECTUAL				INFORMAÇÕES ADICIONAIS
			ADIÇÃO COMPORTAMENTAL			ADIÇÃO ESTRUTURAL	
			COMPORTAMENTO TRANSVERSAL	REGRA COMPOSIÇÃO	PONTO DE JUNÇÃO		
#N <nome>	<nome>	Condição da extensão	<nome operação na classe projeto aspectual>	{overlap.after overlap.before override wrap}	<nome classe.nome operacao>	<nome atributo inserido> <nome operação inserida>	Informações relativas à composição

A primeira diferença dessa tabela para a tabela nível 2 é que dividimos as adições providas pelo aspecto em estrutural e comportamental. Até então, estávamos nos preocupando apenas com adição comportamental provida pelo aspecto, isto é, preocupávamos apenas em como o aspecto iria alterar o fluxo de execução do caso de uso. Entretanto, no fluxo de projeto, passamos a refletir melhor a respeito de como o comportamento do aspecto deverá ser implementado. Algumas vezes isso exige que a estrutura de alguma classe de projeto afetada seja alterada, ou seja, que sejam adicionados membros (atributos e/ ou operações) a ela. Por esse motivo a tabela de composição utilizada nesse fluxo inclui uma coluna de nome “adição estrutural” onde devem ser especificados os membros que precisarão ser adicionados. A identificação da necessidade desse tipo de adição será realizada na atividade “Projetar classe”, quando estivermos analisando como o comportamento transversal de uma classe de projeto aspectual deverá ser implementado. Um exemplo da necessidade desse tipo de alteração será apresentado no estudo de caso (Seção 5.4.1.2). O roteiro para o preenchimento da tabela de composição nível 3 encontra-se na Figura 4.17.

1. Para cada linha na tabela de nível 2 (Tabela 4.8) correspondente, crie uma linha na tabela nível 3 para indicar **adição comportamental**, preenchendo suas colunas da seguinte maneira:
 - 1.1. **CASO DE USO AFETADO**: preencha com o mesmo conteúdo dessa coluna na tabela nível 2.
 - 1.2. **CLASSE DE PROJETO AFETADA**:
 - 1.2.1. veja qual a classe de análise especificada na coluna “classe de análise afetada” na tabela de nível 2.
 - 1.2.2. verifique em que classe de projeto essa classe de análise é mapeada, recorrendo para isso ao mapeamento entre classes de análise e classes de projeto.
 - 1.2.3. se a classe de análise foi mapeada em apenas uma classe de projeto então a coluna “classe de projeto afetada” será essa classe; senão analise em qual das classes de projeto o comportamento do aspecto deve ser inserido.
 - 1.3. **CONDIÇÃO**: preencha com o mesmo conteúdo dessa coluna na tabela nível 2.
 - 1.4. **COMPORTAMENTO TRANSVERSAL**: preencha com a operação que concretizará a responsabilidade especificada na coluna “responsabilidade do aspecto” na tabela de nível 2, identificando parâmetros e tipos de retorno;
 - 1.5. **REGRAS DE COMPOSIÇÃO**: preencha com o mesmo conteúdo dessa coluna na tabela nível 2.
 - 1.6. **PONTO DE JUNCÃO**:
 - 1.6.1. veja qual é o ponto de junção especificado na tabela de nível 2;
 - 1.6.2. preencha essa coluna com a mensagem da interação entre entidades de projeto no diagrama de sequência que corresponde ao ponto de junção especificado na tabela de nível 2;
 - 1.7. **INFORMAÇÕES ADICIONAIS (OPCIONALMENTE)**: preencha com o conteúdo da mesma coluna na tabela de nível 2 e com alguma outra informação que deseje acrescentar.
2. Se o comportamento transversal necessitar que seja feita **adição estrutural** em alguma classe, então, para cada membro que precisar ser inserido, crie uma linha na tabela de nível 3, na qual deverão ser especificadas somente as seguintes colunas:
 - 2.1. **CLASSE DE PROJETO AFETADA**: preencha com o nome da classe onde o membro deverá ser inserido;
 - 2.2. **ADIÇÃO ESTRUTURAL**: preencha com o nome do atributo/operação a ser inserido;
 - 2.3. **INFORMAÇÕES ADICIONAIS (OPCIONALMENTE)**: preencha com alguma informação que deseje acrescentar.

Figura 4.17 - Roteiro para o preenchimento da tabela de composição nível 3

Usando como exemplo a classe de projeto aspectual *VerificacaoDebito* identificada na seção anterior, para preencher sua tabela de composição nível 3, o primeiro passo é recorrer as informações da tabela de composição nível 2 correspondente (Tabela 4.9). Com essas informações é possível preencher as colunas “caso de uso afetado”, “condição”, “regra de composição” e “informações adicionais”.

Verifica-se então que são afetados os casos de uso *Emitir Histórico*, *Fazer matrícula em disciplina* e *Trancar matrícula em disciplina*. Para o caso de uso *Fazer matrícula em disciplina*, por exemplo, na Tabela 4.9 é indicado que a classe de análise afetada é *ControladorMatricula*, logo, considerando o mapeamento entre classes de análise e classe de projeto para esse caso de uso (Tabela 4.10), a classe de projeto afetada também será *ControladorMatricula*. Para descobrir o ponto de junção nesse caso de uso, analisamos seu diagrama de seqüência (Figura 4.16) e, considerando o “ponto de junção” especificado na tabela de nível 2, o “ponto de junção” na tabela nível 3 será *CadastroDiscOfertada.consultaDiscOfertadas(aluno,periodo)*. A responsabilidade desse aspecto nesse caso de uso seria concretizada pela operação *verificarDebito(idAluno)* e, por isso, a coluna “comportamento transversal” corresponderá a essa operação. Não será necessária nenhuma adição estrutural nessa situação. Ao final da análise de cada linha da Tabela 4.9, obteríamos a Tabela 4.13.

Tabela 4.13 - Tabela de composição nível 3 para classe de projeto aspectual *VerificacaoDebito*

CLASSE DE PROJETO ASPECTUAL: VERIFICACAODeBITO							
CASO DE USO AFETADO	CLASSE PROJETO AFETADA	COND.	ADIÇÕES PROVIDAS PELA CLASSE DE PROJETO ASPECTUAL				INFORMAÇÕES ADICIONAIS
			ADIÇÃO COMPORTAMENTAL			ADIC. ESTR.	
			COMPORTAMENTO TRANSVERSAL	REGRA COMPOSIÇÃO	PONTO DE JUNÇÃO		
UC#04: Emitir Histórico	...	...	...	...	...	...	...
UC#05: Fazer matrícula em disciplina	Controlador Matricula	-	verificarDebito (idAluno)	wrap	CadastroDiscOfertada/ consultaDiscOfertadas (aluno,periodo)	-	Se semDebito então execute(pontoJuncao) Senão exibe(msgErro)
UC#06: Trancar matrícula em disciplina	...	...	...	...	...	...	...

É possível gerar uma visão do diagrama de seqüência de um caso de uso com a presença dos objetos de projeto aspectuais, a partir dos diagramas de seqüência dos casos de uso base e da análise das tabelas de composição nível 3. A Figura 4.18 apresenta o diagrama de seqüência do caso de uso *Fazer matrícula em disciplina* incluindo o objeto de projeto aspectual *VerificacaoDebito* que afetará a interação. Esse diagrama foi gerado a partir da Figura 4.16 e da Tabela 4.13. Nesse diagrama, o objeto aspectual é simbolizado por um losango e as mensagens que ele alterará na interação estão destacadas por linhas espessas.

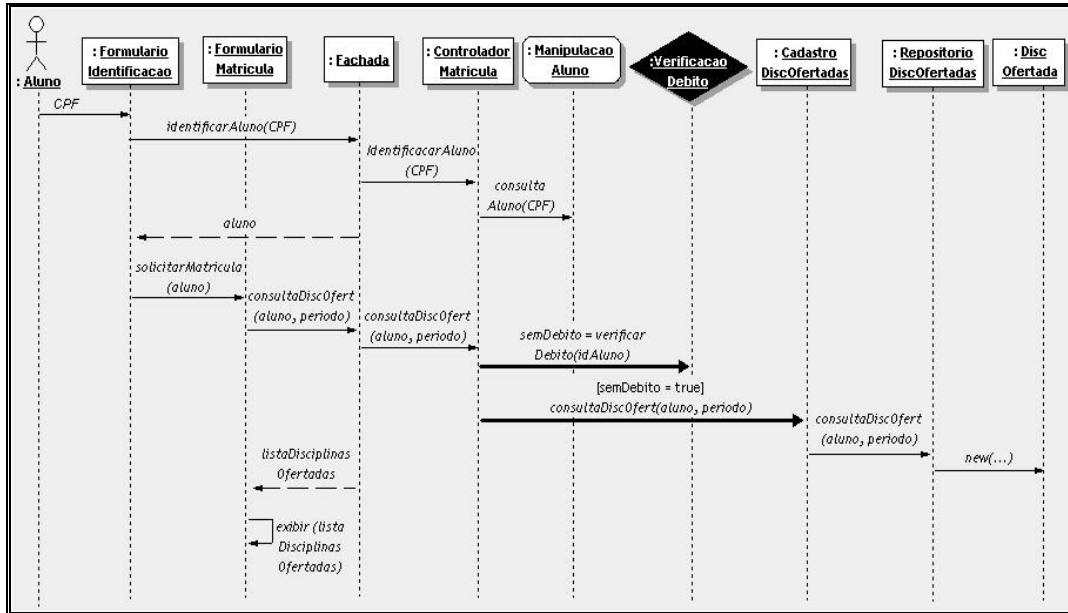


Figura 4.18 – Visão do diagrama de sequência da primeira parte do caso de uso *Fazer matrícula em disciplina* COM a presença do objeto de projeto aspectual *VerificacaoDebito*

4.4.2 Projetar classes

Nesta atividade analisamos cada classe de projeto aspectual individualmente, a fim de definir sua estrutura (atributos, operações), seus relacionamentos e como suas operações serão implementadas. A principal entrada dessa atividade serão as tabelas de composição nível 3.

A partir da coluna “comportamento transversal” da tabela de composição nível 3, identificamos as operações que deverão ser invocadas pelas classes afetadas. Na definição da estrutura da classe de projeto aspectual, essas operações serão antecedidas do estereótipo <<weave>>.

A seguir, deve-se analisar como essas operações devem ser implementadas. Caso essa implementação exija alguma adição estrutural em outra classe, essa informação deve ser inserida na tabela de composição nível 3, conforme passo 2 do roteiro apresentado na Figura 4.17. Um exemplo da necessidade desse tipo de alteração será apresentado no estudo de caso (Seção 5.4.1.2).

Diagramas de sequência podem ser usados para modelar quais são as interações necessárias para realizar uma operação de uma classe de projeto aspectual. A

Figura 4.19 exibe num diagrama de seqüência as interações que ocorrem quando a operação *verificarDebito* da classe de projeto *VerificacaoDebito* é solicitada.

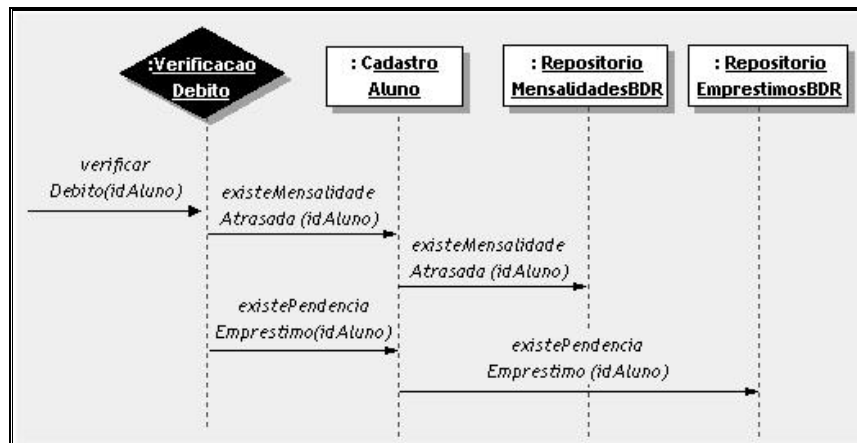


Figura 4.19 – Exemplo de diagrama de seqüência para realização da operação *verificarDebito* da classe de projeto aspectual *VerificacaoDebito*

A partir da coluna “Adição estrutural” podemos identificar atributos e operações necessários para realização do comportamento do aspecto que deverão ser inseridos em outras classes. Esses atributos e operações serão antecedidos do estereótipo <<introduction>> na definição da estrutura da classe.

As classes que deverão invocar operações do tipo <<weave>> e as classes nas quais serão feitas introduções de membros estarão especificadas na coluna “classe de projeto afetada” da tabela de nível 3. O relacionamento entre classes de projeto aspectuais e as classes afetadas por ela será um relacionamento do tipo associação com o estereótipo <<crosscuts>>.

A Figura 4.20 ilustra uma parte do diagrama de classes do sistema acadêmico, enfatizando como seria representada a estrutura da classe de projeto aspectual *VerificacaoDebito* e seu relacionamento com a classe *ControladorMatricula*. Por ser uma classe de projeto aspectual, ela terá o estereótipo <<aspect>>. A operação *verificarDebito* é antecedida do estereótipo <<weave>> porque representa uma operação que deve ser invocada pela classe *ControladorMatricula*.

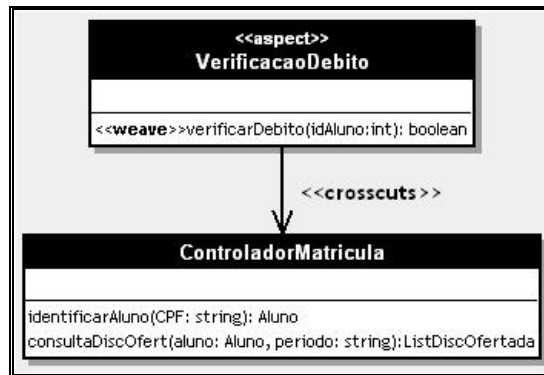


Figura 4.20 – Definição da estrutura da classe de projeto aspectual *VerificacaoDebito*

4.5 Considerações finais

As modificações propostas agregam os fundamentos do paradigma orientado a aspectos nos fluxos de requisitos, análise e projeto do Processo Unificado para permitir a separação de preocupações transversais nos artefatos manipuladas nesses fluxos.

Em resumo, nossa abordagem possibilita que, ao longo dos fluxos de requisitos, análise e projeto do Processo Unificado, o comportamento de preocupações transversais seja centralizado em uma entidade à parte dos artefatos que ela afeta e que os pontos de junção e as regras de composição nessas unidades sejam especificadas em tabelas de composição.

Nossa proposta não se prende a mecanismos de uma abordagem orientada a aspectos particular. Ao invés disso, utiliza apenas os fundamentos comuns dessas abordagens tais como: centralização da preocupação transversal numa única unidade, identificação de pontos de junção e regras de composição. Isso possibilita que mesmo usando nossa proposta desde as atividades de requisitos até o projeto, o engenheiro possa utilizar os artefatos gerados ao longo dessas atividades sem necessariamente ter que implementar o sistema com uma linguagem orientada a aspectos particular; podendo até mesmo utilizar esses artefatos para implementar o sistema com uma linguagem orientada a objetos tradicional.

As tabelas de composição têm papel essencial para conseguirmos nosso objetivo de evitar o espalhamento e entrelaçamento de preocupações transversais em artefatos de software, pois centralizam informações importantes da composição de uma preocupação transversal com as unidades nas quais ela deve ser aplicada e permitem

que preocupações transversais sejam acopladas e desacopladas facilmente no sistema, sem que vários artefatos precisem ser alterados. Contudo, como essas tabelas possuem informações muito dependentes dos artefatos que participam da composição, sempre que esses artefatos forem alterados, é necessário verificar se há a necessidade de alguma atualização na tabela de composição. O custo dessa atualização seria minimizado se fossem usadas ferramentas para gerenciar essas dependências.

Neste trabalho as tabelas de composição representam a composição da perspectiva do aspecto (Tabela 4.14), ou seja, indicam para um determinado aspecto quais unidades serão afetadas por ele e como ele afetará cada uma delas. Essa perspectiva é útil para o engenheiro de software responsável pela análise/projeto/implementação do aspecto, pois as informações que ele necessita são centralizadas na tabela de composição.

Tabela 4.14 - Tabela de composição nível 1 (perspectiva do aspecto)

REQUISITO ASPECTUAL: # ID <NOME>				
CASO DE USO AFETADO	CONDIÇÃO (OPCIONAL)	REGRA DE COMPOSIÇÃO	PONTO DE JUNÇÃO	INFORMAÇÕES ADICIONAIS

Entretanto, varrendo-se as tabelas de composição com a perspectiva do aspecto, é possível gerar uma visão das tabelas de composição com a perspectiva de casos de uso (Tabela 4.15). Essa perspectiva permite que sejam visualizados, para cada caso de uso, quais aspectos afetam aquele caso de uso e em que ponto do caso de uso esses aspectos devem ser aplicados. Esse processo pode ser automatizado por ferramentas. Uma das utilidades da perspectiva de casos de uso das tabelas de composição é ajudar a identificar conflitos entre aspectos.

Tabela 4.15 - Tabela de composição nível 1 com perspectiva de caso de uso

CASO DE USO: UC # N <NOME>				
REQUISITO ASPECTUAL	CONDIÇÃO (OPCIONAL)	REGRA DE COMPOSIÇÃO	PONTO DE JUNÇÃO	INFORMAÇÕES ADICIONAIS

Neste capítulo mostramos o uso da nossa abordagem utilizando um exemplo simples de verificação de débito num sistema acadêmico. Esse exemplo permitiu oferecer apenas uma noção dos benefícios que pretendemos alcançar com nossa abordagem. No entanto, no capítulo seguinte, que exibirá um estudo de caso completo

num sistema de *Internet Banking*, será possível avaliar melhor se nossa proposta contribuirá para reusabilidade, manutenibilidade e compreensibilidade dos artefatos gerados nos fluxos de requisitos análise e projeto do Processo Unificado.

5

ESTUDO DE CASO: INTERNET BANKING

One key issue is moving beyond code to separation of concerns across the software life cycle, including management of relationships across artifacts.

Harold Ossher and Peri Tarr

Existem duas técnicas básicas para avaliar uma abordagem: experimentos e estudos de caso. A realização de experimentos provê uma maneira de avaliação baseada em comparação direta, permitindo a pesquisadores investigar qual o impacto da tecnologia no processo de maneira detalhada [TRAVASSOS et al., 2002]. Já a realização de estudos de caso está mais preocupada em avaliar os benefícios de uma abordagem de forma a verificar se as mudanças no processo proporcionam o resultado desejado [KITCHENHAM et al., 1994].

A técnica de avaliação escolhida neste trabalho foi o estudo de caso. Neste capítulo apresentamos a utilização da nossa abordagem (apresentada no Capítulo 4) em um sistema de *Internet Banking*. O objetivo é propiciar discussões a respeito do uso da abordagem, de maneira que possa ser analisado se as mudanças propostas ajudam a dar suporte à separação de preocupações transversais nos fluxos de requisitos, análise e projeto do Processo Unificado [JACOBSON et al., 1999].

5.1 Objetivos

Este estudo de caso tem o propósito de: (i) mostrar como nossa abordagem pode ser usada para separar preocupações transversais nos fluxos de requisitos, análise e projeto do Processo Unificado; e (ii) avaliar se nossa proposta de adaptação contribuirá para reusabilidade, manutenibilidade e compreensibilidade dos artefatos gerados nesses fluxos.

Para isso, aplicamos a nossa abordagem no contexto de um sistema bancário disponível na Internet, também denominado de sistema de *Internet Banking*. O principal objetivo desse tipo de sistema é permitir que clientes realizem transações bancárias através da Internet. Os serviços cobertos por esse estudo de caso serão: visualizar saldo, visualizar extrato de conta, fazer transferência e realizar pagamento de boleto bancário. Também trataremos da preocupação de segurança nesse tipo de sistema.

Escolhemos esse sistema porque além de representar um domínio de aplicação bem-conhecido, a reusabilidade e a manutenibilidade dos seus artefatos são qualidades necessárias no seu contexto. O reuso é importante, pois a maioria dos artefatos desse sistema pode ser reutilizado em sistemas de outros bancos e também em sistemas bancários para guichês de caixa ou para caixas eletrônicos. A manutenibilidade é primordial porque as regras de negócio mudam frequentemente nesse contexto devido a novas leis ou a mudanças nas políticas adotadas pelo banco.

Conforme apresentado na Seção 3.1, a separação de preocupações é uma premissa fundamental para alcançar tais qualidades. Pretendemos demonstrar que, utilizando nossa abordagem é possível separar até mesmo preocupações não-funcionais e regras de negócio, que comumente ficam espalhadas e entrelaçadas nos artefatos de software.

Na sequência, damos uma visão geral de como nossa proposta pode ser usada nos fluxos de atividades de requisitos, análise e projeto.

5.2 Fluxo de requisitos

Nas subseções seguintes, focamos na aplicação da nossa proposta nas atividades de requisitos, enfatizando as alterações que propomos.

Inicialmente, realizamos o tratamento de preocupações funcionais, identificando atores e casos de uso (Seção 5.2.1) e especificando esses casos de uso (Seção 5.2.2). A seguir, focamos no tratamento de preocupações não-funcionais, analisando e refinando essas preocupações (Seção 5.2.3) e especificando suas operacionalizações (Seção 5.2.4). Por fim, estruturamos o modelo de requisitos para que ele seja mais fácil de manipular nos fluxos posteriores (Seção 5.2.5).

5.2.1 Identificar atores e casos de uso

Inicialmente, identificamos um ator (cliente do banco) e cinco casos de uso que esse ator pode realizar (Efetuar *login*, Visualizar saldo de conta, Visualizar extrato de conta, Realizar transferência e Pagar boleto bancário).

Também identificamos a seguinte regra de negócio:

- Regra #01: para cada transação de transferência e pagamento, deve ser calculado o imposto CPMF de 0,38% do valor debitado da conta. Esse valor deverá ser acumulado em um campo específico para cada conta. O imposto CPMF acumulado nas transações realizadas será debitado em um determinado dia da semana (sexta) da conta do cliente, zerando o imposto a ser debitado. Sendo assim, o saldo disponível para transações financeiras do cliente no dia do débito deverá considerar a cobrança do CPMF acumulado. Portanto, o `saldoDisponivelCPMF` será o `saldoDisponivel` subtraído do CPMF acumulado.

Essa regra de negócio deve ser aplicada em todos os casos de uso referentes a transações financeiras, e, normalmente, ficaria misturada com a especificação desses casos de uso. No entanto, nossa abordagem fornece mecanismos para evitar o espalhamento e entrelaçamento desse tipo de requisito nas especificações. Por isso, nas especificações dos casos de uso de transações financeiras que apresentamos na Seção

5.2.2 não há referências a essa regra de negócio; o impacto desse requisito nos casos de uso é modelado na atividade “Estruturar o modelo de casos de uso” (Seção 5.2.5).

5.2.2 Especificar casos de uso

As especificações dos casos de uso identificados são apresentadas abaixo. Como não existe nenhum modelo padrão sugerido por UML para especificar casos de uso, neste trabalho usamos uma versão simplificada do modelo sugerido por COCKBURN (2000) (vide modelo completo na Figura 2.3).¹³

Tabela 5.1 – Especificação do caso de uso *Efetuar Login*

CASO DE USO #01: EFETUAR LOGIN	
INFORMAÇÕES CARACTERÍSTICAS	
OBJETIVO	Iniciar a sessão do usuário no sistema, identificando sua conta e dando acesso ao menu principal do sistema.
DISPARADOR	O cliente escolhe a opção “Acessar conta”.
PRECONDIÇÕES	O usuário ser cliente do banco.
ATOR PRIMÁRIO	Cliente do banco, titular da conta.
CENÁRIO PRINCIPAL DE SUCESSO	
PASSO	Ação
1	O sistema solicita que o usuário informe dados da conta (número de conta, agência e senha da conta).
2	O usuário informa os dados.
3	O sistema verifica se existe conta no banco que corresponda aos dados informados.
4	O sistema exibe a tela principal do sistema.
CENÁRIOS ALTERNATIVOS OU EXTENSÕES	
PASSO	AÇÃO DE DESVIO
3a	Conta inexistente: 3a1. O sistema exibe mensagem de erro. 3a2. O sistema retorna para o passo 1.
3b	A senha informada está errada (menos de três tentativas): 3b1. O sistema exibe mensagem de erro. 3b2. O sistema retorna para o passo 1.
3c	A senha informada está errada (três tentativas): 3c1. O sistema bloqueia a conta do usuário e o objetivo do caso de uso falha.

Tabela 5.2 – Especificação do caso de uso *Visualizar saldo de conta*

CASO DE USO #02: VISUALIZAR SALDO DE CONTA	
INFORMAÇÕES CARACTERÍSTICAS	
OBJETIVO	Visualizar o saldo atual da conta.
DISPARADOR	O cliente escolhe a opção “Saldo” no menu de transações.
PRECONDIÇÕES	A conta do cliente já deve ter sido previamente identificada
ATOR PRIMÁRIO	Cliente do banco, titular da conta.
CENÁRIO PRINCIPAL DE SUCESSO	
PASSO	Ação
1	O sistema recupera a conta do cliente logado.
2	O sistema exibe o saldo da conta.
3	O sistema grava informações sobre a transação: identificação da conta, data, hora.
4	O sistema verifica se a quantidade máxima do tipo transação “visualização de saldo” para a modalidade da conta naquele mês foi ultrapassada. Se a quantidade foi ultrapassada debita o valor da tarifa na conta do cliente.

¹³ Os passos com sombreado preto serão alvo de estruturação na Seção 5.2.5.

Tabela 5.3 – Especificação do caso de uso *Visualizar extrato de conta*

CASO DE USO #03: VISUALIZAR EXTRATO DE CONTA	
INFORMAÇÕES CARACTERÍSTICAS	
OBJETIVO	Visualizar o extrato da conta exibindo informações de lançamentos num período informado.
DISPARADOR	O cliente escolhe a opção "Extrato" no menu de transações.
PRECONDIÇÕES	A conta do cliente já deve ter sido previamente identificada
ATOR PRIMÁRIO	Cliente do banco, titular da conta.
CENÁRIO PRINCIPAL DE SUCESSO	
PASSO	AÇÃO
1	O cliente informa o período do extrato.
2	O sistema recupera a conta do cliente logado.
3	O sistema valida os dados do extrato (verifica se o período é válido). O período é inválido se: (i) excede 120 dias; (ii) a data inicial é posterior a data final; ou (iii) a data inicial é posterior a data atual.
4	O sistema exibe a lista de movimentação financeira de todos os débitos e créditos que ocorreram na conta do cliente durante o período informado. Para cada item da lista são especificados: data, descrição, número do documento, valor e tipo ("d" para débito e "c" para "crédito"). Ao final da listagem, o saldo também é exibido.
5	O sistema grava informações sobre a transação: conta, data, hora, período.
6	O sistema verifica se a quantidade máxima do tipo transação "visualizar extrato de conta" para a modalidade da conta naquele mês foi ultrapassada. Se a quantidade foi ultrapassada debita o valor da tarifa na conta do cliente.
CENÁRIOS ALTERNATIVOS OU EXTENSÕES	
PASSO	AÇÃO DE DESVIO
3a	O cliente informa um intervalo de data inválido: 3a1. Uma mensagem de erro é exibida 3a2. O cliente repete o passo 1.

Tabela 5.4 – Especificação do caso de uso *Realizar Transferência*

CASO DE USO #04: REALIZAR TRANSFERÊNCIA	
INFORMAÇÕES CARACTERÍSTICAS	
OBJETIVO	Realizar transferência de valor entre uma conta e outra no mesmo banco.
DISPARADOR	O cliente escolhe a opção "Transferência" no menu de transações.
PRECONDIÇÕES	A conta do cliente já deve ter sido previamente identificada
ATOR PRIMÁRIO	Cliente do banco, titular da conta.
CENÁRIO PRINCIPAL DE SUCESSO	
PASSO	AÇÃO
1	O cliente informa os dados da transferência (valor da transferência, a agência e conta de destino).
2	O sistema recupera a conta do cliente logado.
3	O sistema valida os dados da transferência (verifica se a conta destino existe).
4	O sistema verifica se a conta do cliente tem saldo disponível suficiente para realizar a transação.
5	O sistema exibe tela para confirmação dos dados.
6	O cliente confirma os dados da transferência.
7	O sistema debita o valor da transferência da conta do cliente e credita na conta de destino.
8	O sistema grava informações sobre a transação: identificação da conta origem, identificação da conta destino, valor.
9	O sistema cadastra um número randômico para o cliente concorrer a prêmios no mês seguinte a cada nova dezena de transações financeiras realizadas no mês.
10	O cliente informa como deseja que o comprovante seja emitido.
11	O sistema emite o comprovante.
CENÁRIOS ALTERNATIVOS OU EXTENSÕES	
PASSO	AÇÃO DE DESVIO
3a	Conta de destino não existe: 3a1. O sistema exibe uma mensagem de erro 3a2. O cliente repete o passo 1 ou cancela a operação.
4a	A conta do cliente não tem saldo disponível suficiente: 4a1. O sistema exibe uma mensagem de erro. 4a2. O cliente retorna para o passo 1 ou cancela a operação.
6a	O cliente não confirma os dados 6a1. O retorna ao passo 1 ou cancela a operação.
VARIAÇÃO EM TECNOLOGIA OU DADOS	
PASSO	VARIAÇÃO
10	O comprovante pode ser emitido das seguintes maneiras: Exibido na tela; Salvo em arquivo txt; Salvo em arquivo pdf.

Tabela 5.5 – Especificação do caso de uso *Pagar boleto bancário*

CASO DE USO #05: PAGAR BOLETO BANCÁRIO	
INFORMAÇÕES CARACTERÍSTICAS	
OBJETIVO	Realizar pagamentos de contas com código de barras de conveniados.
DISPARADOR	O cliente escolhe a opção "Pagamento boleto" no menu de transações.
PRECONDIÇÕES	A conta do cliente já deve ter sido previamente identificada
ATOR PRIMÁRIO	Cliente do banco, titular da conta.
CENÁRIO PRINCIPAL DE SUCESSO	
PASSO	AÇÃO
1	O cliente informa dados do pagamento (código de barras, data de pagamento e valor do pagamento).
2	O sistema recupera a conta do cliente logado.
3	O sistema valida os dados informados.
4	O sistema exibe tela para confirmação dos dados.
5	O cliente confirma os dados do pagamento.
6	O sistema verifica se a conta do cliente tem saldo disponível suficiente para realizar a transação.
7	O sistema debita o valor da conta do cliente.
8	O sistema <u>processa a conta</u> (include Processar conta).
9	O sistema grava informações sobre a transação: identificação da conta, código de barras, data pagamento, valor.
10	O sistema cadastra um número randômico para o cliente concorrer a prêmios no mês seguinte a cada nova dezena de transações financeiras realizadas no mês.
11	O cliente informa como deseja que o comprovante seja emitido.
12	O sistema emite o comprovante.
CENÁRIOS ALTERNATIVOS OU EXTENSÕES	
PASSO	AÇÃO DE DESVIO
5a	O cliente não confirma os dados 5a1. O retorna ao passo 1 ou cancela a operação.
5b	Se a data pagamento é superior a data atual: 5b1. O sistema gera uma requisição de agendamento de pagamento. 5b2. O sistema exibe mensagem avisando que pagamento foi agendado. 5b3. Na data programada, o sistema executa os passos 7 a 10. 5b4. O sistema retorna ao passo 11.
6a	A conta do cliente não tem saldo disponível suficiente: 6a1. O sistema exibe uma mensagem de erro. 6a2. O cliente retorna ao passo 1 ou cancela a operação.
VARIAÇÃO EM TECNOLOGIA OU DADOS	
PASSO	VARIAÇÃO
11	O comprovante pode ser emitido das seguintes maneiras: Exibido na tela; Salvo em arquivo txt; Salvo em arquivo pdf.

5.2.3 Identificar e analisar requisitos não-funcionais

Uma preocupação importante ao desenvolver sistemas de informação para serem usados na Internet é a segurança, isto é, proteger o sistema contra falhas, desde acessos não-autorizados até a interrupção de serviço [CHUNG et al., 2000]. Nesta atividade, refinamos essa preocupação até chegar em operacionalizações que ajudarão a concretizá-la.

A partir de catálogos oferecidos pelo *framework* NFR [CHUNG et al., 2000], decompomos essa preocupação em três outras: confidencialidade, integridade e disponibilidade.

Confidencialidade é uma preocupação prioritária, pois as instituições bancárias têm obrigação de guardar informações de seus clientes para que não ocorra divulgação não-autorizada. Evitar interrupção de serviço (disponibilidade) é outra preocupação importante já que contribui consideravelmente para satisfação dos clientes que utilizam o sistema. Visto que invasões são comuns na Internet, a integridade dos dados (i.e. proteger informações dos clientes e das transações realizadas para que não ocorra atualização não-autorizada ou falsificação) é outra preocupação característica desse tipo de sistema. A preocupação de integridade pode ser subdividida nas preocupações de completude e precisão. A primeira significa garantir a totalidade (completude) dos dados mantidos pelo sistema; a precisão está relacionada à correspondência entre os valores no sistema e aquilo que eles de fato devem representar. Essa decomposição de preocupações pode ser visualizada na Figura 5.1.

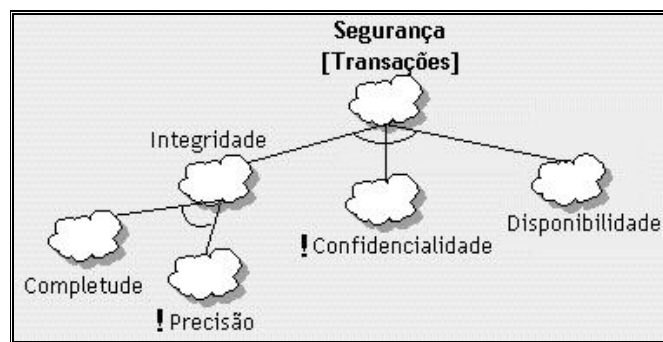


Figura 5.1 – Decomposição da preocupação segurança em outras preocupações

Após a decomposição das preocupações não-funcionais, o próximo passo é analisar as possíveis operacionalizações que podem contribuir para satisfação de cada uma dessas preocupações. A Figura 5.2 apresenta o resultado dessa análise num grafo, indicando os refinamentos realizados na preocupação de segurança até chegar em operacionalizações que contribuem para sua satisfação.

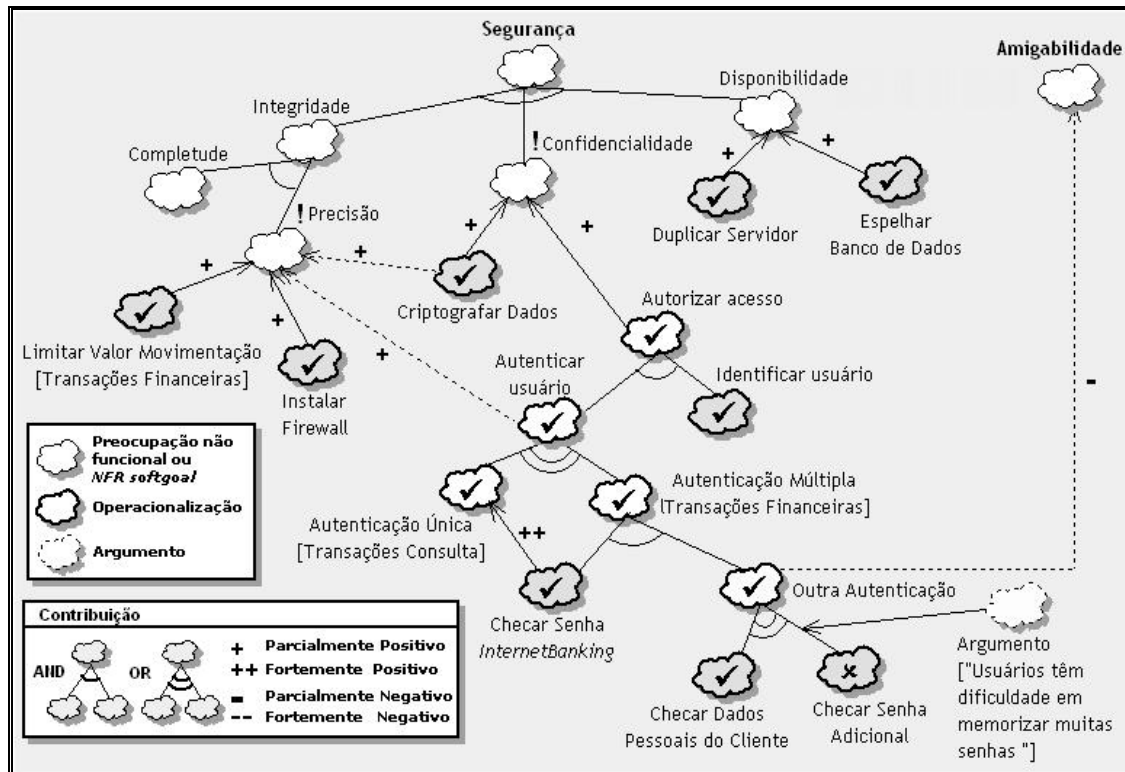


Figura 5.2 – Decomposição da preocupação de segurança em operacionalizações

Começando pela preocupação de confidencialidade, escolhemos duas operacionalizações que contribuem positivamente para sua satisfação: criptografar dados e autorizar acesso. A primeira assegura que as informações trocadas nas transações sejam decifradas apenas pelo sistema; a segunda assegura que os usuários são, de fato, quem eles afirmam ser. Por sua vez, decompomos a operacionalização de autorizar acesso, em outras duas operacionalizações: identificar usuário e autenticar usuário. No caso da autenticação, decidimos que as transações de consulta exigiriam apenas um mecanismo de autenticação (checar senha InternetBanking); enquanto as financeiras exigiriam além desse mecanismo uma outra autenticação. Teríamos duas opções para esse outro mecanismo de autenticação: checar dados pessoais do cliente ou checar senha adicional. Entretanto, verificamos que ter mais de um mecanismo de autenticação contribui negativamente para satisfação da preocupação de amigabilidade com o usuário.

Analisando a preocupação precisão, propomos duas possíveis alternativas que contribuem para diminuir o risco de fraudes: (i) limitar o valor das movimentações e

(ii) instalar um firewall para evitar acessos não-autorizados ao banco de dados. Além disso, verificamos que as operacionalizações de autenticar usuário e criptografar dados também contribuem positivamente para satisfazer a preocupação de precisão dos dados.

Por sua vez, possíveis soluções que contribuem positivamente para satisfazer a preocupação de disponibilidade são: duplicar servidor e espelhar banco de dados.

O passo seguinte desta atividade é selecionar as operacionalizações propostas, aceitando (✓) ou rejeitando (✗) cada uma delas. No nosso estudo de caso, a única operacionalização rejeitada foi checagem de senha adicional. A razão para isso está relacionada à dificuldade dos usuários em memorizar muitas senhas.

Dentre as operacionalizações selecionadas (folhas no grafo da Figura 5.2), algumas delas correspondem a restrições de projeto (ex: instalar *firewall*, duplicar servidor e espelhar banco de dados) e outras a restrições operacionais que devem ser impostas no fluxo de execução de casos de uso (ex: limitar valor de movimentação, checar senha, checar dados pessoais do cliente, identificar usuário). Para cada restrição operacional selecionada, analisamos que casos de uso elas afetam:

- Limitar valor de movimentação: deve ser aplicada nos casos de uso referentes à transações financeiras, ou seja, nos casos de uso *Realizar transferência* e *Pagar boleto bancário*;
- Checar senha Internet Banking: deve ser aplicada em todos os casos de uso referentes a transações bancárias, ou seja, nos casos de uso *Visualizar saldo de conta*, *Visualizar extrato de conta*, *Realizar transferência* e *Pagar boleto bancário*;
- Checar dados pessoais do cliente: deve ser aplicada somente nos casos de uso referentes à transações financeiras, ou seja, nos casos de uso *Realizar transferência* e *Pagar boleto bancário*;
- Identificar usuário: corresponde ao caso de uso *Efetuar Login*;
- Criptografar dados: deve ocorrer na transmissão de dados pela Internet (casos de uso não cobertos pelo estudo de caso).

Referências a essas operacionalizações geralmente ficariam misturadas com a especificação dos casos de uso nos quais elas devem ser aplicadas. No entanto, nossa

abordagem fornece mecanismos para evitar o espalhamento e entrelaçamento desse tipo de requisito nas especificações. Por isso, nas especificações dos casos de uso que apresentamos na Seção 5.2.2 não há referências a elas; o impacto desse tipo de requisito nos casos de uso é modelado na atividade “Estruturar o modelo de casos de uso” (Seção 5.2.5).

5.2.4 Especificar requisitos não-funcionais

Após analisar a preocupação de segurança, chegamos a operacionalizações, algumas delas representando restrições de projeto (ex: instalar *firewall*, duplicar servidor e espelhar banco de dados) e outras a restrições operacionais (ex: limitar valor de movimentação, checar senha, checar dados pessoais do cliente, identificar usuário).

A especificação de restrições de projeto provenientes de preocupação não-funcionais será feita numa seção especial do documento de requisitos, o que está fora do escopo desse trabalho. As restrições operacionais, conforme indicado na Seção 4.2.2, podem ser tratadas e especificadas de maneira semelhante a casos de uso.

Tabela 5.6 – Especificação do caso de uso *Checar senha Internet Banking*

OPERACIONALIZACAO #01: CHECAR SENHA INTERNET BANKING	
INFORMAÇÕES CARACTERÍSTICAS	
OBJETIVO	Checar a identidade do usuário, comparando a senha Internet Banking informada com a senha cadastrada na conta. Esse caso de uso contribui para satisfação da seguinte hierarquia de preocupação: SEGURANÇA → Confidencialidade → Autorização de acesso → Autenticação → Autenticação única e Autenticação Múltipla
PRECONDIÇÕES	A conta do cliente já deve ter sido previamente identificada
ATOR PRIMÁRIO	Cliente do banco, titular da conta.
CENÁRIO PRINCIPAL DE SUCESSO	
PASSO	AÇÃO
1	O sistema solicita que o usuário informe a senha Internet Banking.
2	O usuário informa a senha.
3	O sistema compara senha informada com a senha da conta.
4	O resultado da comparação é retornado.
CENÁRIOS ALTERNATIVOS OU EXTENSÕES	
PASSO	AÇÃO DE DESVIO
3a	A senha informada está errada (menos de três tentativas): 3a1. Uma mensagem de erro é exibida. 3a2. O usuário repete o passo 1 ou cancela a transação.
3b	A senha informada está errada (três tentativas): 3b1. A conta é bloqueada e o objetivo do caso de uso falha.

Tabela 5.7 – Especificação do caso de uso *Checar dados pessoais do cliente*

OPERACIONALIZACAO #02: CHECAR DADOS PESSOAIS DO CLIENTE	
INFORMAÇÕES CARACTERÍSTICAS	
OBJETIVO	Checar a identidade do cliente por meio de solicitação de algum de seus dados pessoais para comparar com os dados cadastrados na conta. O tipo de informação solicitada deve ser decidido randomicamente. Esse caso de uso contribui para satisfação da seguinte hierarquia de preocupação: SEGURANÇA → Confidencialidade → Autorização de acesso → Autenticação → Autenticação única e Autenticação Múltipla
PRECONDIÇÕES	A conta do cliente já deve ter sido previamente identificada
ATOR PRIMÁRIO	Cliente do banco titular da conta
CENÁRIO PRINCIPAL DE SUCESSO	
PASSO	Ação
1	O sistema seleciona randomicamente qual dado pessoal será solicitado.
2	O sistema solicita que o usuário informe esse tipo de dado pessoal
3	O usuário informa o dado pessoal.
4	O sistema compara o dado pessoal informado com o dado cadastrado na conta.
5	O resultado da comparação é retornado.
CENÁRIOS ALTERNATIVOS OU EXTENSÕES	
PASSO	AÇÃO DE DESVIO
4a	O dado informado está errado (menos de três tentativas): 4a1. Uma mensagem de erro é exibida. 4a2. O usuário repete o passo 1 ou cancela a transação.
4b	O dado informado está errado (três tentativas): 4b1. A conta é bloqueada e o objetivo falha.
VARIAÇÃO EM TECNOLOGIA OU DADOS	
PASSO	VARIAÇÃO
1	O dado pessoal solicitado pode ser: dia do nascimento; mês de nascimento; ano nascimento; número do CPF; ou número do RG.

Tabela 5.8 – Especificação do caso de uso *Limitar valor de movimentação*

OPERACIONALIZACAO #03: LIMITAR VALOR DE MOVIMENTAÇÃO	
INFORMAÇÕES CARACTERÍSTICAS	
OBJETIVO	Verificar limite do valor de transações financeiras para diminuir riscos de fraudes. Essa operacionalização contribui para satisfação da seguinte hierarquia de preocupação: SEGURANÇA → Integridade → Precisão
PRÉ-CONDIÇÕES	Valor da transação é dado como entrada.
ATOR PRIMÁRIO	Cliente do banco, titular da conta.
CENÁRIO PRINCIPAL DE SUCESSO	
PASSO	Ação
1	O sistema verifica se o valor da transação supera limite.
2	O sistema retorna o resultado da verificação

5.2.5 Estruturar o modelo de casos de uso

Nesta atividade, usamos os mecanismos disponíveis (*generalize*, *include*, *extend* e *crosscuts*) para estruturar comportamento compartilhado e transversal entre os artefatos de requisitos previamente descritos (Seção 5.2.5.1). A seguir, considerando as estruturações realizadas, atualizamos as especificações dos casos de uso (Seção 5.2.5.2). Por fim, especificamos as tabelas de composição nível 1 para os requisitos aspectuais (Seção 5.2.5.3) e identificamos e resolvemos conflitos (Seção 5.2.5.4).

5.2.5.1 IDENTIFICANDO INCLUSÕES, EXTENSÕES E REQUISITOS ASPECTUAIS

A princípio, analisamos apenas os casos de usos. Percebemos que existem os seguintes comportamentos comuns nos casos de uso descritos na Seção 5.2.2:

- i) a preocupação em cobrar tarifa por transações acima do permitido aparece:
 - no passo 4 do caso de uso Visualizar saldo da conta (Tabela 5.2); e
 - no passo 6 do caso de uso Visualizar extrato da conta (Tabela 5.3);
- ii) a preocupação em gravar informações da transação está presente:
 - no passo 3 do caso de uso Visualizar saldo da conta (Tabela 5.2);
 - no passo 5 do caso de uso Visualizar extrato da conta (Tabela 5.3);
 - no passo 8 do caso de uso Realizar transferência (Tabela 5.4); e
 - no passo 9 do caso de uso Pagar boleto bancário (Tabela 5.5);
- iii) a preocupação em oferecer bonificação para concorrer a prêmios aparece:
 - no passo 9 do caso de uso Realizar transferência (Tabela 5.4); e
 - no passo 10 do caso de uso Pagar boleto bancário (Tabela 5.5);
- iv) a preocupação em emitir comprovante está presente:
 - nos passos 10 e 11 do caso de uso Realizar transferência (Tabela 5.4); e
 - nos passos 11 e 12 do caso de uso Pagar boleto bancário (Tabela 5.5);

Extraímos cada um desses comportamentos comuns e criamos um novo caso de uso para cada um deles. Tomando como base as heurísticas apresentadas na Seção 4.2.3.1, determinamos o tipo mais adequado de relacionamento (*generalize*, *include*, *extend* ou *crosscuts*) entre esses novos casos de uso e os casos de uso nos quais eles devem ser inseridos:

- a execução do caso de uso *Emitir comprovante de transação* é uma parte essencial para realização do objetivo dos casos de uso *Fazer transferência* e *Pagar boleto bancário*. Por isso, dizemos que esses dois últimos casos de uso incluem (i.e. têm um relacionamento do tipo *include com*) o caso de uso *Emitir comprovante de transação*;

- o mesmo não ocorre com as demais situações identificadas: os requisitos *Cobrar tarifa por transação extra*, *Gravar informações da transação*, e *Oferecer bonificação* não são essenciais para realização do objetivo dos casos de uso que eles afetam, já que podem ser desacoplados destes sem prejudicar a realização do propósito dos casos de uso afetados. Além disso, esses requisitos devem ser aplicados em vários casos de uso e não apenas num caso de uso específico. Sendo assim, podemos classificá-los como requisitos aspectuais e, conseqüentemente, os casos de uso que irão representá-los terão um relacionamento do tipo *crosscuts* com os casos de uso nos quais esses requisitos devem ser aplicados.

Conforme mencionamos na Seção 4.2.3.1, uma outra maneira de identificar requisitos aspectuais é verificando restrições operacionais que alteram a execução de casos de uso e satisfazem às heurísticas apresentadas. Dessa maneira, tomando como base as heurísticas apresentadas na Seção 4.2.3.1 sobre como identificar requisitos aspectuais, analisamos as restrições operacionais obtidas na Seção 5.2.3 e identificamos aquelas que podem ser consideradas como requisitos aspectuais: *Limitar valor de movimentação*, *Checar senha InternetBanking* e *Checar dados pessoais do cliente*. Todas essas restrições terão um relacionamento do tipo *crosscuts* com os casos de uso que elas afetam.

De maneira semelhante, analisando a regra de negócio relativa à cobrança do imposto CPMF, identificamos que essa regra de negócio (i) altera o fluxo de execução dos casos de uso *Realizar transferência* e *Pagar boleto bancário*, mas seu propósito não está diretamente associado ao objetivo desses casos de uso; (ii) pode ser desvinculada desses casos de uso sem prejudicar a realização do objetivo deles; e (iii) é aplicada em vários casos de uso. Por isso, esse tipo de requisito também é um requisito aspectual e terá um relacionamento do tipo *crosscuts* com os casos de uso *Realizar transferência* e *Pagar boleto bancário*.

Por fim, identificamos um fluxo complexo no caso de uso *Pagar boleto bancário* que diz respeito ao agendamento de pagamento. Este fluxo é específico do caso de uso *Pagar boleto bancário* e só ocorre nos casos em que a data de pagamento é posterior a data atual. Sendo assim, retiramos esse fluxo do caso de uso *Pagar boleto*

bancário e criamos o caso de uso *Agendar Pagamento*, que terá um relacionamento do tipo *extends* com o caso de uso *Pagar boleto bancário*.

A Figura 5.3 ilustra graficamente como o modelo de casos de uso ficou após as estruturas realizadas.

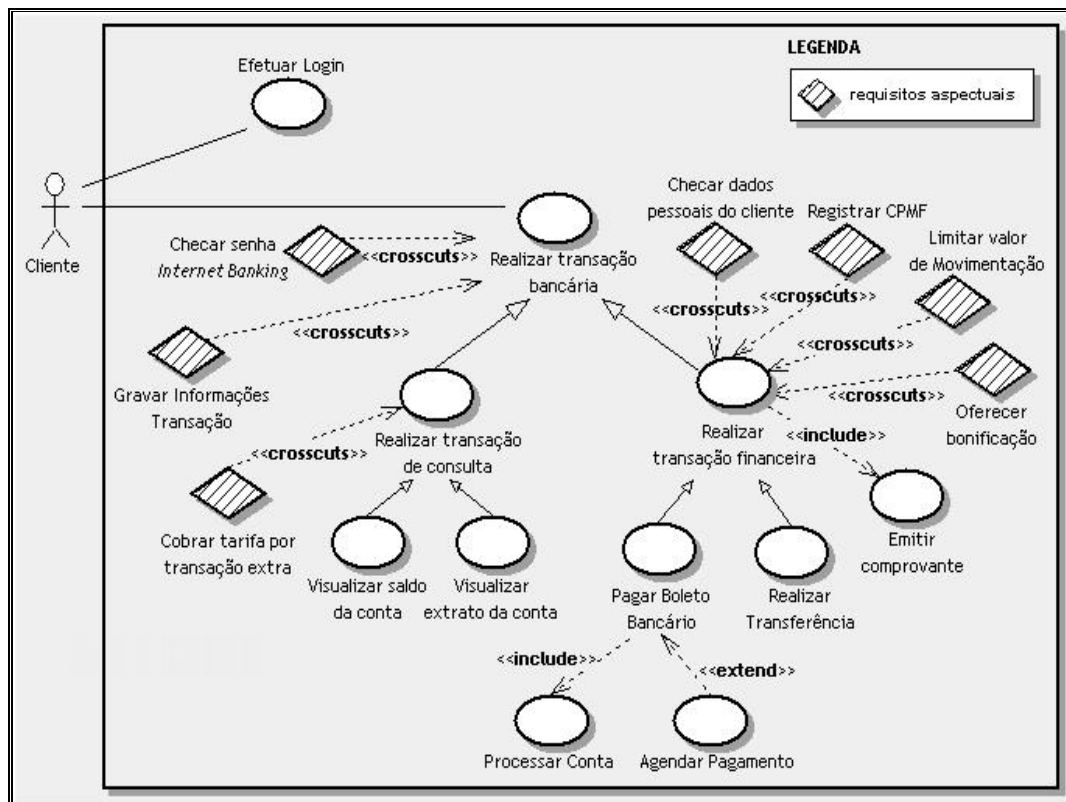


Figura 5.3 - Diagrama de caso de uso após estruturação

5.2.5.2 ATUALIZAR ESPECIFICAÇÕES DE CASOS DE USO

Nesta atividade as especificações dos casos de uso são atualizadas para refletir as estruturas realizadas.

As preocupações relativas à gravação de informações da transação e à cobrança de tarifa foram retiradas da especificação dos casos de uso *Visualizar saldo de conta* e *Visualizar extrato de conta*. A Tabela 5.9 exibe a especificação do caso de uso *Visualizar saldo de conta* e a Tabela 5.10 a especificação do caso de uso *Visualizar extrato de conta* após a estruturação.

Tabela 5.9 – Especificação do cenário principal do caso de uso *Visualizar saldo de conta* (após estruturação)

CASO DE USO #02: VISUALIZAR SALDO DE CONTA	
CENÁRIO PRINCIPAL DE SUCESSO	
PASSO	Ação
1	O sistema recupera a conta do cliente logado.
2	O sistema exibe o saldo da conta.

Tabela 5.10 – Especificação do cenário principal do caso de uso *Visualizar extrato de conta* (após estruturação)

CASO DE USO #03: VISUALIZAR EXTRATO DE CONTA	
CENÁRIO PRINCIPAL DE SUCESSO	
PASSO	Ação
1	O cliente informa o período do extrato.
2	O sistema recupera a conta do cliente logado.
3	O sistema valida os dados do extrato (verifica se o período é válido).
4	O sistema exibe a lista de movimentação financeira de todos os débitos e créditos que ocorreram na conta do cliente no período informado. Para cada item da lista são especificados: data, descrição, número do documento, valor e tipo ("d" para débito e "c" para "crédito"). Ao final da listagem, o saldo também é exibido.

Comparando as especificações acima com as inicialmente realizadas (Tabela 5.2 e Tabela 5.3), pode-se perceber que as especificações desses casos de uso após a estruturação ficaram mais coesas com o objetivo a que eles se propõem, facilitando a compreensão, reuso e manutenção dessas especificações. Vale a pena frisar ainda que, caso não estivéssemos usando a nossa abordagem, além de conter referências às preocupações de gravação de informações e cobrança de tarifa, as especificações dos casos de uso na Tabela 5.2 e na Tabela 5.3 ainda conteriam uma referência à restrição de segurança relativa à checagem de senha *InternetBanking*.

Os benefícios da nossa abordagem podem ser ainda melhor percebidos nas especificações dos casos de uso *Realizar transferência* e *Pagar boleto bancário*. Como vimos na seção anterior, identificamos seis requisitos aspectuais que deveriam ser considerados nesses casos de uso: (i) *Checar senha Internet Banking*; (ii) *Checar dados pessoais do cliente*; (iii) *Cobrar CPMF*; (iv) *Limitar valor de movimentação*; (v) *Oferecer bonificação*; e (vi) *Gravar informações da transação*. Normalmente esses requisitos estariam misturados na especificação desses casos de uso. A Tabela 5.11 mostra um exemplo de especificação do caso de uso *Realizar transferência* na qual todos os requisitos aspectuais que devem ser considerados nesse caso de uso (realçados em cinza) são misturados na especificação do cenário principal desse caso de uso. A partir desse exemplo, pode-se perceber que esse tipo de especificação dificulta:

- o entendimento do real objetivo do caso de uso, pois preocupações adicionais são misturadas no cenário essencial do caso de uso, inserindo complexidade extra na especificação;
- o reuso integral dessa especificação em sistemas de outros bancos que não adotem alguma das preocupações inseridas no caso de uso, tais como: checagem de senha ou dados pessoais; verificação de valor limite de transações; ou oferecimento de bonificação;
- a manutenção dos artefatos em situações de mudança ou evolução de requisitos, visto que tanto para inserir como para excluir alguma dessas preocupações transversais, a especificação de todos os casos de uso afetados precisariam ser alterados.

Tabela 5.11 - Especificação do cenário principal do caso de uso *Realizar Transferência* incluindo referências a requisitos aspectuais

CASO DE USO #04: REALIZAR TRANSFERÊNCIA
CENÁRIO PRINCIPAL DE SUCESSO
O cliente informa os dados da transferência (valor da transferência, a agência e conta de destino).
O sistema recupera a conta do cliente logado.
O sistema verifica se o valor da transferência ultrapassa o limite diário para conta.
O sistema checa a identidade do usuário, comparando a senha Internet Banking informada com a senha cadastrada na conta.
O sistema valida os dados da transferência (verifica se a conta destino existe).
O sistema verifica se a conta do cliente tem saldo disponível suficiente para realizar a transação. Se o CPMF acumulado for debitado na data da transação, deve-se descontar o CPMF acumulado do saldo disponível.
O sistema checa a identidade do cliente por meio de solicitação de algum de seus dados pessoais para comparar com os dados cadastrados na conta. O tipo de informação solicitada deve ser decidido randomicamente.
O sistema exibe tela para confirmação dos dados.
O cliente confirma os dados da transferência.
O sistema debita o valor da transferência da conta do cliente e credita na conta de destino.
O sistema calcula o valor do imposto CPMF, e acumula esse valor num campo específico para ser debitado futuramente da conta do cliente.
O sistema grava informações sobre a transação: identificação da conta origem, identificação da conta destino, valor.
O sistema cadastra um número randômico para o cliente concorrer ao prêmio do mês a cada nova dezena de transações financeiras realizadas no mês.
O cliente informa como deseja que o comprovante seja emitido.
O sistema emite o comprovante.

Por sua vez, usando nossa abordagem, os casos de uso *Realizar transferência* e *Pagar boleto bancário* seriam especificados como representado na Tabela 5.12e na Tabela 5.13, respectivamente. Pode-se perceber que estas especificações tratam apenas das preocupações relacionadas ao propósito do caso de uso. O modo como os requisitos aspectuais devem ser aplicadas no fluxo desses casos de uso é definido em tabelas de composição, como veremos na seção a seguir. Isso facilita o reuso desses artefatos em

sistemas semelhantes e também a sua manutenibilidade em casos de mudanças de requisitos.

Tabela 5.12 – Especificação do cenário principal do caso de uso *Realizar Transferência* (após estruturação)

CASO DE USO #04: REALIZAR TRANSFERÊNCIA	
CENÁRIO PRINCIPAL DE SUCESSO	
PASSO	AÇÃO
1	O cliente informa os dados da transferência (valor da transferência, a agência e conta de destino).
2	O sistema recupera a conta do cliente logado.
3	O sistema valida os dados da transferência (verifica se a conta destino existe). E caso exista, recupera a conta destino.
4	O sistema verifica se a conta do cliente tem saldo disponível suficiente para realizar a transação.
5	O sistema exibe tela para confirmação dos dados.
6	O cliente confirma os dados da transferência.
7	O sistema debita o valor da transferência da conta do cliente e credita na conta de destino.
8	O sistema <u>emite o comprovante</u> . (include Emitir comprovante)

Tabela 5.13 – Especificação do cenário principal do caso de uso *Pagar boleto bancário* (após estruturação)

CASO DE USO #05: PAGAR BOLETO BANCÁRIO	
CENÁRIO PRINCIPAL DE SUCESSO	
PASSO	AÇÃO
1	O cliente escolhe a opção "Pagar boleto bancário" no menu de transações.
2	O sistema recupera a conta do cliente logado.
3	O cliente informa dados do pagamento (código de barras, data de pagamento e valor do pagamento).
4	O sistema valida os dados informados.
5	O sistema exibe tela para confirmação dos dados.
6	O cliente confirma os dados do pagamento.
7	O sistema verifica se a conta do cliente tem saldo disponível suficiente para realizar a transação.
8	O sistema debita o valor da conta do cliente.
9	O sistema <u>processa a conta</u> . (include Processar conta).
10	O sistema <u>emite o comprovante</u> . (include Emitir comprovante)

5.2.5.3 ESPECIFICAR TABELAS DE COMPOSIÇÃO NÍVEL 1

Nesse passo, criamos uma tabela de composição nível 1 para cada requisito aspectual (conforme explicado na Seção 4.2.3.2). Basicamente, essas tabelas centralizam, de maneira estruturada, as informações que estariam espalhadas nas especificações de casos de uso.

Essa centralização favorece o reuso das especificações dos casos de uso e ainda facilita a manutenção e evolução de requisitos. Por exemplo, para inserir, alterar ou excluir algum requisito aspectual do sistema, o único artefato que precisará ser alterado será sua tabela de composição correspondente. Caso não estivéssemos usando nossa abordagem, todas as especificações dos casos de uso afetados pelo requisito aspectual deveriam ser alteradas.

Nesta etapa do processo já sabemos em quais casos de uso cada requisito aspectual deverá ser aplicado, mas precisamos analisar em que ponto e como o requisito aspectual deverá ser inserido no fluxo de execução desses casos de uso. Essas informações devem ser especificadas numa tabela de composição nível 1 segundo as instruções do roteiro apresentado na Figura 4.7.

A cobrança de tarifa por transação extra deve ser realizada **depois** (*overlap.after*) da execução do último passo nos fluxos de execução dos casos de uso *Visualizar saldo de conta* (Tabela 5.9) e *Visualizar extrato de conta* (Tabela 5.10). Desse modo, preenchemos a tabela de composição nível 1 para o requisito aspectual *Cobrar tarifa por transação extra*, conforme apresentado na Tabela 5.14.

Tabela 5.14 - Tabela de composição nível 1 para requisito aspectual *Cobrar tarifa por transação extra*

REQUISITO ASPECTUAL: AR #01 COBRAR TARIFA POR TRANSAÇÃO EXTRA				
CASO DE USO AFETADO	CONDIÇÃO (OPCIONAL)	REGRA DE COMPOSIÇÃO	PONTO DE JUNÇÃO	INFORMAÇÕES ADICIONAIS
UC #02 Visualizar saldo de conta	Quantidade máxima do tipo transação "visualizar saldo de conta" para a modalidade da conta naquele mês foi ultrapassada	overlap.after	2	-
UC #03 Visualizar extrato de conta	Quantidade máxima do tipo transação "visualizar extrato de conta" para a modalidade da conta naquele mês foi ultrapassada	overlap.after	4	-

A gravação de informações da transação deve ocorrer **depois** (*overlap.after*) que o passo chave da transação já tiver sido efetivado. Nos casos de uso *Visualizar saldo de conta* (Tabela 5.9) e *Visualizar Extrato de conta* (Tabela 5.10) o ponto de junção seria a exibição do saldo e do extrato; respectivamente os passos 2 e 4 desses casos de uso. No caso de uso *Realizar transferência* (Tabela 5.12) o ponto de junção seria o débito e o crédito nas contas (passo 7 desse caso de uso). No caso de uso *Pagar boleto bancário* (Tabela 5.13) o ponto de junção seria o processamento da conta (passo 9 desse caso de uso). A Tabela 5.15 reflete essas informações.

Tabela 5.15 - Tabela de composição nível 1 para requisito aspectual *Gravar informações da transação*

REQUISITO ASPECTUAL: AR #02 GRAVAR INFORMAÇÕES DA TRANSAÇÃO				
CASO DE USO AFETADO	CONDIÇÃO (OPCIONAL)	REGRA DE COMPOSIÇÃO	PONTO DE JUNÇÃO	INFORMAÇÕES ADICIONAIS
UC #02 Visualizar saldo de conta	-	overlap.after	2	-
UC #03 Visualizar extrato de conta	-	overlap.after	4	-
UC #04 Realizar transferência	-	overlap.after	7	-
UC #05 Pagar boleto bancário	-	overlap.after	9	-

A cobrança de CPMF afeta os casos de uso *Realizar transferência* e *Pagar boleto bancário* em duas situações: (i) se o dia da transação corresponder ao dia do débito periódico do CPMF, **ao invés de** (*override*) retornar o saldo disponível ao consultá-lo, deve ser retornado: $\text{saldoDisponível} - \text{CPMFacumulado}$; e (ii) **após** (*overlap.after*) debitar o valor da conta do cliente, deve-se acumular num campo específico o valor do CPMF da transação. Essas informações são modeladas de maneira estruturada na Tabela 5.16.

Tabela 5.16 - Tabela de composição nível 1 para o requisito aspectual *Cobrar CPMF*

REQUISITO ASPECTUAL: AR #03 COBRAR CPMF				
CASO DE USO AFETADO	CONDIÇÃO	REGRA DE COMPOSIÇÃO	PONTO DE JUNÇÃO	INFORMAÇÕES ADICIONAIS
UC #04 Realizar transferência	O dia da transação corresponder ao dia do débito periódico do CPMF	override	4	$\text{saldoDisponívelCPMF} = \text{saldoDisponível} - \text{CPMFacumulado}$
UC #04 Realizar transferência	-	overlap.after	7	$\text{CPMFacumulado} = \text{CPMFacumulado} + (\text{valorDebito} * \text{cpmf})$
UC #05 Pagar boleto bancário	O dia da transação corresponder ao dia do débito periódico do CPMF	override	7	$\text{saldoDisponívelCPMF} = \text{saldoDisponível} - \text{CPMFacumulado}$
UC #05 Pagar boleto bancário	-	overlap.after	8	$\text{CPMFacumulado} = \text{CPMFacumulado} + (\text{valorBoleto} * \text{cpmf})$

O oferecimento de bonificação aplica-se aos casos de uso *Realizar transferência* e *Pagar boleto bancário*. No primeiro caso de uso isso deve ser realizado **após** (*overlap.after*) o débito e o crédito nas contas (passo 7 desse caso de uso) e no segundo caso de uso após (*overlap.after*) o processamento da conta (passo 9 desse caso de uso). Essas informações são sintetizadas na Tabela 5.17.

Tabela 5.17 - Tabela de composição nível 1 para o requisito aspectual *Oferecer bonificação*

REQUISITO ASPECTUAL: AR #04 OFERECER BONIFICAÇÃO				
CASO DE USO AFETADO	CONDIÇÃO (OPCIONAL)	REGRA DE COMPOSIÇÃO	PONTO DE JUNÇÃO	INFORMAÇÕES ADICIONAIS
UC #04 Realizar transferência	A cada nova dezena de transações financeiras realizadas no mês.	overlap.after	7	-
UC #05 Pagar boleto bancário	A cada nova dezena de transações financeiras realizadas no mês.	overlap.after	9	-

O preenchimento das tabelas de composição nível 1 para os requisitos aspectuais *Checar senha Internet Banking*, *Checar dados pessoais do cliente* e *Limitar valor de movimentação* seguiram a mesma lógica que descrevemos acima. A única diferença notável é que as regras de composição utilizada nas tabelas de composição desses requisitos aspectuais (Tabela 5.18, Tabela 5.19 e Tabela 5.20, respectivamente)

foram do tipo *wrap*, já que a execução dos pontos de junção nos casos de uso que eles afetam dependerá do resultado obtido na execução desses aspectos.

Tabela 5.18 - Tabela de composição nível 1 para o requisito aspectual *Checar senha Internet Banking*

REQUISITO ASPECTUAL: AR #05 CHECAR SENHA INTERNET BANKING				
CASO DE USO AFETADO	COND.	REGRA DE COMPOSIÇÃO	PONTO DE JUNÇÃO	INFORMAÇÕES ADICIONAIS
UC #02 Visualizar saldo de conta	-	Wrap	2	Se senhaOk então execute (ponto junção) Senão exibe(msgErro)
UC #03 Visualizar extrato de conta	-	Wrap	4	Se senhaOk então execute (ponto junção) Senão exibe(msgErro)
UC #04 Realizar transferência	-	Wrap	3	Se senhaOk então execute (ponto junção) Senão exibe(msgErro)
UC #05 Pagar boleto bancário	-	Wrap	4	Se senhaOk então execute (ponto junção) Senão exibe(msgErro)

Tabela 5.19 - Tabela de composição nível 1 para o requisito aspectual *Checar dados pessoais do cliente*

REQUISITO ASPECTUAL: AR #06 CHECAR DADOS PESSOAIS DO CLIENTE				
CASO DE USO AFETADO	CONDIÇÃO (OPCIONAL)	REGRA DE COMPOSIÇÃO	PONTO DE JUNÇÃO	INFORMAÇÕES ADICIONAIS
UC #04 Realizar transferência	-	wrap	5	Se dadosOk então execute (ponto junção) Senão exibe(msgErro)
UC #05 Pagar boleto bancário	-	wrap	5	Se dadosOk então execute (ponto junção) Senão exibe(msgErro)

Tabela 5.20 - Tabela de composição nível 1 para o requisito aspectual *Limitar valor de movimentação*

REQUISITO ASPECTUAL: AR #07 LIMITAR VALOR DE MOVIMENTAÇÃO				
CASO DE USO AFETADO	CONDIÇÃO (OPCIONAL)	REGRA DE COMPOSIÇÃO	PONTO DE JUNÇÃO	INFORMAÇÕES ADICIONAIS
UC #04 Realizar transferência	-	Wrap	3	Se limiteOk então execute (ponto junção) Senão exibe(msgErro)
UC #05 Pagar boleto bancário	-	Wrap	4	Se limiteOk então execute (ponto junção) Senão exibe(msgErro)

5.2.5.4 IDENTIFICAR E RESOLVER CONFLITOS

As tabelas de composição nível 1 obtidas na seção anterior representam a composição da perspectiva do aspecto. Para identificar possíveis conflitos entre requisitos aspectuais, geramos as tabelas de composição nível 1 com a perspectiva de casos de uso. Essa perspectiva permite que sejam visualizados, para cada caso de uso, quais aspectos afetam aquele caso de uso e em que ponto dos casos de uso esses aspectos devem ser aplicados. Para gerar essa perspectiva, varremos todas as tabelas de nível 1 obtidas na Seção 5.2.5.3.

Como pode ser visualizado na Tabela 5.21, os requisitos aspectuais *Cobrar tarifa por transação extra* e *Gravar informações da transação* devem ser aplicados no caso de uso *Visualizar saldo de conta*, no mesmo ponto de junção e com a mesma regra de composição. Isso caracteriza um conflito potencial. O mesmo ocorre no caso de uso *Visualizar extrato de conta* (vide Tabela 5.22). Nos dois casos de uso, decidimos dar prioridade à gravação de informações, visto que primeiro deve-se gravar as informações da transação e depois deve ser realizada a cobrança de tarifa. Essa decisão foi especificada nas linhas L2 e L3 da tabela de conflitos (Tabela 5.25).

Tabela 5.21 - Tabela de composição nível 1 com perspectiva do caso de uso *Visualizar saldo de conta*

CASO DE USO: UC #02 VISUALIZAR SALDO DE CONTA				
REQUISITO ASPECTUAL	CONDIÇÃO (OPCIONAL)	REGRA DE COMPOSIÇÃO	PONTO DE JUNÇÃO	INFORMAÇÕES ADICIONAIS
→ AR #01 - Cobrar tarifa por transação extra	Quantidade máxima do tipo transação "visualizar saldo de conta" para a modalidade da conta naquele mês foi ultrapassada	overlap.after	2	-
→ AR #02 - Gravar informações da transação	-	overlap.after	2	-
AR #05 - Checar senha Internet Banking	-	wrap	2	Se senhaOk então execute (pontojunção) Senão exibe(msgErro)

Tabela 5.22 - Tabela de composição nível 1 com perspectiva do caso de uso *Visualizar extrato de conta*

CASO DE USO: UC #03 VISUALIZAR EXTRATO DE CONTA				
REQUISITO ASPECTUAL	CONDIÇÃO (OPCIONAL)	REGRA DE COMPOSIÇÃO	PONTO DE JUNÇÃO	INFORMAÇÕES ADICIONAIS
→ AR #01 - Cobrar tarifa por transação extra	Quantidade máxima do tipo transação "visualizar extrato de conta" para a modalidade da conta naquele mês foi ultrapassada	overlap.after	4	-
→ AR #02 - Gravar informações da transação	-	overlap.after	4	-
AR #05 - Checar senha Internet Banking	-	wrap	4	Se senhaOk então execute (ponto junção) Senão exibe(msgErro)

Conforme indicado na Tabela 5.23 e na Tabela 5.24, também identificamos conflitos nos casos de uso *Realizar transferência* e *Pagar boleto bancário* entre os requisitos aspectuais *Gravar informações da transação* e *Oferecer bonificação*. Novamente, a prioridade foi dada para o requisito de gravação de informações de transação. No caso de uso da transferência, a cobrança de CPMF também conflitava com esses dois requisitos aspectuais e decidimos que ela deveria suceder a gravação de

informações da transação. Essas decisões foram especificadas nas linhas L4 e L5 da tabela de conflitos (Tabela 5.25).

Outro conflito identificado nos casos de uso *Realizar transferência* e *Pagar boleto bancário* foi entre os requisitos aspectuais *Checar senha Internet Banking* e *Limitar valor de movimentação*, ficando esse último com a prioridade de execução, conforme especificado nas linhas L4 e L5 da tabela de conflitos (Tabela 5.25).

Tabela 5.23 - Tabela de composição nível 1 com perspectiva do caso de uso *Realizar transferência*

CASO DE USO: UC #04 REALIZAR TRANSFERÊNCIA				
REQUISITO ASPECTUAL	CONDIÇÃO (OPCIONAL)	REGRA DE COMPOSIÇÃO	PONTO DE JUNÇÃO	INFORMAÇÕES ADICIONAIS
→ AR #02 - Gravar informações da transação	-	overlap.after	7	-
AR #03 – Cobrar CPMF	O dia da transação corresponder ao dia do débito periódico do CPMF	override	4	saldoDisponivelCPMF = saldoDisponivel – CPMF acumulado
→ AR #03 - Cobrar CPMF	-	overlap.after	7	CPMFacumulado = CPMFacumulado + (valorDebito*cpmf)
→ AR #04 - Oferecer bonificação	A cada nova dezena de transações financeiras realizadas no mês.	overlap.after	7	-
→ AR #05 - Checar senha Internet Banking	-	wrap	3	Se senhaOk então execute (pontoJunção) Senão exibe(msgErro)
AR #06 - Checar dados pessoais do cliente	-	wrap	5	Se dadosOk então execute (ponto junção) Senão exibe(msgErro)
→ AR #07 - Limitar valor de movimentação	-	wrap	3	Se limiteOk então execute(ponto junção) Senão exibe(msgErro)

Tabela 5.24 - Tabela de composição nível 1 com perspectiva do caso de uso *Pagar boleto bancário*

CASO DE USO: UC # 05 PAGAR BOLETO BANCÁRIO				
REQUISITO ASPECTUAL	CONDIÇÃO (OPCIONAL)	REGRA DE COMPOSIÇÃO	PONTO DE JUNÇÃO	INFORMAÇÕES ADICIONAIS
→ AR #02 - Gravar informações da transação	-	overlap.after	9	-
AR #03 - Cobrar CPMF	O dia da transação corresponder ao dia do débito periódico do CPMF	override	7	saldoDisponivelCPMF = saldoDisponivel – CPMF acumulado
AR #03 - Cobrar CPMF	-	overlap.after	8	CPMFacumulado = CPMFacumulado + (valorBoleto*cpmf)
→ AR #04 - Oferecer bonificação	A cada nova dezena de transações financeiras realizadas no mês.	overlap.after	9	-
→ AR #05 - Checar senha Internet Banking	-	wrap	4	Se senhaOk então execute (ponto junção) Senão exibe(msgErro)
AR #06 - Checar dados pessoais do cliente	-	wrap	5	Se dadosOk então execute (ponto junção) Senão exibe(msgErro)
→ AR #07 - Limitar valor de movimentação	-	Wrap	4	Se limiteOk então execute(ponto junção) Senão exibe(msgErro)

Tabela 5.25 - Tabela de conflitos

		AR#01	AR#02	AR#03	AR#04	AR#05	AR#06	AR#07
L1	UC#01							
L2	UC#02	P2-OA = 2º.	P2-OA = 1º.					
L3	UC#03	P4-OA = 2º.	P4-OA = 1º.					
L4	UC#04		P7-OA = 1º.	P7-OA = 2º.	P7-OA = 3º.		P3-W = 2º	P3-W = 1º
L5	UC#05		P9-OA = 1º.		P9-OA = 2º.		P4-W = 2º	P4-W = 1º

À medida que os pontos de junção vão sendo refinados nos fluxos seguintes, esses conflitos podem deixar de existir. Contudo, essas prioridades definidas na tabela de conflitos serão úteis não somente caso o conflito persista nos fluxos de análise e projeto, mas também para auxiliar na definição dos pontos de junção nesses fluxos.

5.3 Fluxo de análise

Uma vez concluídas as atividades do fluxo de requisitos, iremos analisar como o sistema irá se comportar internamente para satisfazer os requisitos.

Na atividade “Analisar casos de uso” (Seção 5.3.1), decompomos os casos de usos e os requisitos aspectuais em termos de classes de análise responsáveis por concretizar seu comportamento. Em seguida, na atividade “Analisar classes” (Seção 5.3.2), especificamos as responsabilidades, os atributos e os relacionamentos de cada classe de análise identificada.

5.3.1 Analisar casos de uso

Nesta atividade, primeiramente identificamos as classes de análise que serão necessárias para concretizar cada caso de uso e cada requisito aspectual obtido no fluxo de requisitos (Seção 5.3.1.1). Depois descrevemos como os objetos de análise dessas classes interagem para realizar o caso de uso e como os objetos das classes de análise aspectuais devem ser inseridos nessas interações (Seção 5.3.1.2).

5.3.1.1 IDENTIFICAR CLASSES DE ANÁLISE

Iniciamos pela identificação de classes de análise que contribuem para realização de cada caso de uso. A partir das especificações dos casos de uso apresentadas na Seção 5.2.5.2, distribuímos as partes do fluxo de eventos dos casos de uso entre classes do tipo fronteira, entidade e controle. As responsabilidades relacionadas à comunicação entre atores e o sistema foram atribuídas a classes do tipo fronteira. As responsabilidades relacionadas à manipulação de informações no sistema foram atribuídas a classes do tipo entidade. Por sua vez, as responsabilidades relacionadas à delegação de trabalho para outras classes e a lógica de negócio que envolve vários objetos foram atribuídas a classes do tipo controle. A Tabela 5.26 apresenta o mapeamento entre casos de uso e classes de análise que realizamos.

Tabela 5.26 – Mapeamento entre casos de uso e classes de análise

CASOS DE USO	CLASSES DE ANÁLISE		
	ENTIDADE	FRONTEIRA	CONTROLE
UC #02 Visualizar saldo de conta	Conta DadosLogin	InterfaceSaldo	ControladorSaldo
UC #03 Visualizar extrato de conta	Conta DadosLogin MovimentacaoFinanceiraConta	InterfaceExtrato	ControladorExtrato
UC #04 Realizar transferência	Conta DadosLogin	InterfaceTransferencia	ControladorTransferencia EmissorComprovante
UC #05 Pagar boleto bancário	Conta DadosLogin	InterfacePagamento	ControladorPagamento ProcessadorConta EmissorComprovante

Nesta etapa do processo também identificamos as classes de análise necessárias para concretizar o comportamento dos requisitos aspectuais. Para cada um dos requisitos aspectuais, criamos uma classe de análise aspectual correspondente para encapsular as responsabilidades desse requisito nos casos de uso que ele deve afetar. Quando necessário, identificamos classes do tipo fronteira, entidade ou controle que precisam existir para a classe aspectual realizar suas responsabilidades. A Tabela 5.27 mostra as classes de análise que utilizamos para concretizar cada requisito aspectual.

Tabela 5.27 – Mapeamento entre requisitos aspectuais e classes de análise

REQUISITOS ASPECTUAIS	CLASSES DE ANÁLISE			
	ASPECTUAL	ENTIDADE	FRONTEIRA	CONTROLE
AR#01 - Cobrar tarifa por transação extra	CobrancaTransacaoExtra	Conta TransacoesConta	—	—
AR#02 - Gravar de informações da transação	GravacaoInfoTransacao	TransacoesConta	—	—
AR#03 - Cobrar CPMF	CobrancaCPMF	Conta CPMFAcumuladoConta	—	—
AR#04 - Oferecer bonificação	Bonificação	TransacoesConta NumerosSorteio	—	—
AR#05 - Checar senha Internet Banking	ChecagemSenhaIB	Conta	InterfaceEntradaSenhaIB	—
AR#06 - Checar dados pessoais do cliente	ChecagemDadosPessoais	Cliente	InterfaceEntradaDadoPessoal	—
AR#07 – Limitar valor movimentação	VerificacaoLimite	—	—	—

5.3.1.2 DESCREVER INTERAÇÕES ENTRE OBJETOS DE ANÁLISE

Nesta atividade analisamos como um requisito aspectual impactará a interação dos casos de uso que ele afeta. Para isso, inicialmente descrevemos as interações entre objetos de análise que concretizam os casos de uso. A seguir, para cada requisito aspectual, analisamos como o objeto de análise aspectual que o representa deverá ser inserido nas interações dos casos de uso afetados e qual é o papel desse objeto na interação; essas informações são especificadas em tabelas de composição nível 2. Por fim, a partir das informações contidas nas tabelas de composição nível 2, podemos gerar uma visão dos diagramas de interação dos casos de uso com a presença dos objetos aspectuais que afetam o caso de uso.

DESCREVER INTERAÇÕES ENTRE OBJETOS DE ANÁLISE DE CASOS DE USO

A partir das descrições dos casos de uso (Seção 5.2.5.2) e das classes de análise identificadas para cada um deles (Tabela 5.26), especificamos como os objetos de análise interagem para concretizar cada caso de uso.

A Figura 5.4 ilustra como modelamos a colaboração entre objetos de análise para realizar o cenário principal caso de uso *Visualizar Saldo* (especificado na Tabela 5.9).

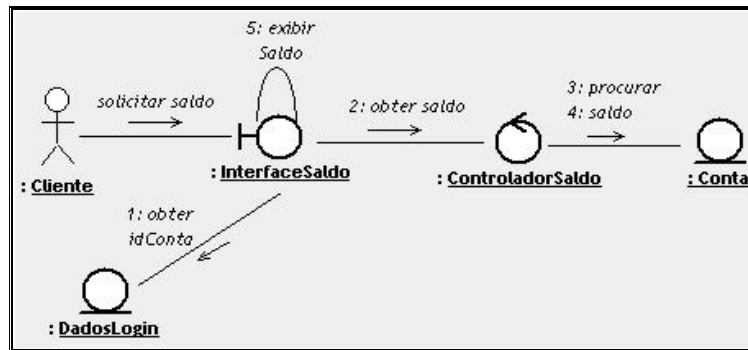


Figura 5.4 - Diagrama de colaboração para caso de uso *Visualizar saldo de conta*

A Figura 5.5 ilustra como modelamos a colaboração entre objetos de análise para realizar o cenário principal caso de uso *Visualizar extrato de conta* (especificado na Tabela 5.10).

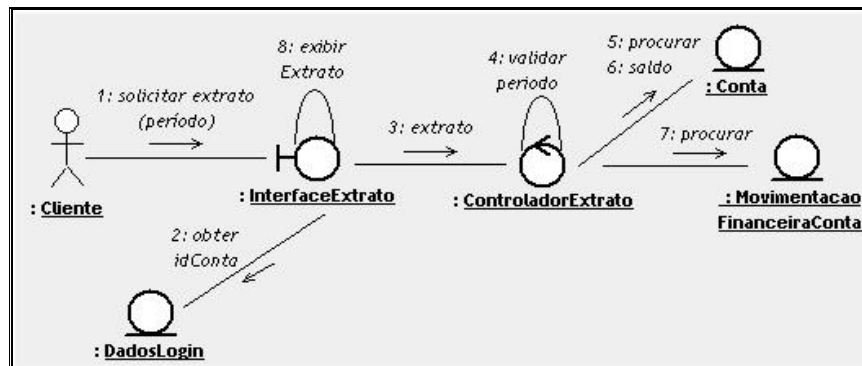


Figura 5.5 – Diagrama de colaboração para caso de uso *Visualizar extrato de conta*

O diagrama de colaboração apresentado na Figura 5.6 exibe o resultado da modelagem que realizamos para realizar o cenário principal do caso de uso *Realizar transferência* (especificado na Tabela 5.12).

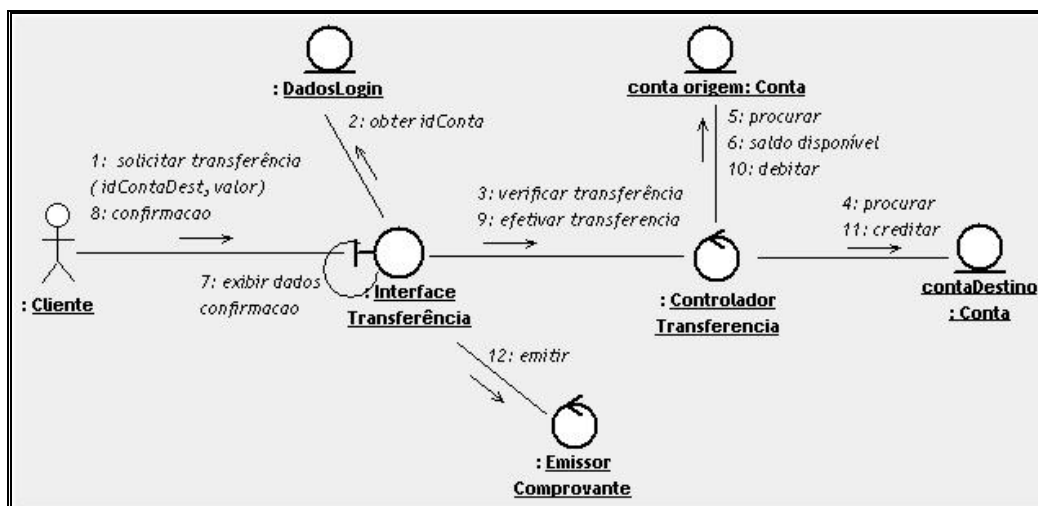


Figura 5.6 - Diagrama de colaboração para caso de uso *Realizar Transferência*

ESPECIFICAR TABELAS DE COMPOSIÇÃO NÍVEL 2

Nesta parte do processo, modelamos como os requisitos aspectuais — representados no fluxo de análise por meio de classes/objetos de análise aspectuais — devem ser inseridos nas interações dos casos de uso que eles afetam. Desta maneira, para cada classe de análise aspectual (*CobrancaTransacaoExtra*, *GravacaoInfoTransacao*, *CobrancaCPMF*, *Bonificação*, *ChecagemSenhaIB*, *ChecagemDadosPessoais* e *VerificacaoLimite*) preenchemos uma tabela de composição nível 2 seguindo o roteiro apresentado na Figura 4.12. Entradas importantes para esta atividade são as tabelas de composição nível 1 dos requisitos aspectuais (Seção 5.2.5.3) e os diagramas de interação obtidos anteriormente.

Como exemplo, descrevemos como preenchemos a tabela de composição da classe de análise aspectual *CobrancaCPMF*. O primeiro passo foi recorrer à tabela de composição nível 1 do requisito que originou essa classe de análise (Tabela 5.16). A primeira linha daquela tabela indica que o requisito aspectual *Cobrança CPMF* deve ser aplicado no caso de uso *Realizar transferência* (caso de uso afetado) substituindo (regra de composição *override*) a consulta do saldo disponível (ponto de junção) que agora passa a ser esse valor descontando o valor do CPMF acumulado (informações adicionais). A partir dessas informações, criamos uma linha na tabela de composição nível 2 (vide Tabela 5.28) e preenchemos as colunas “caso de uso afetado”, “condição”, “regra de composição” e “informações adicionais”. Para preencher as colunas “classe de

análise afetada” e “ponto de junção” deve-se analisar o diagrama de interação do caso de uso *Realizar transferência* (Figura 5.6) e verificar qual parte dessa interação corresponde ao ponto de junção indicado na tabela nível 1. Desta maneira, identificamos que o ponto de junção na interação será a mensagem “saldo disponível” enviada para *Conta*. A coluna “classe de análise afetada” será preenchida com a classe *ControladorTransferencia* porque queremos afetar apenas as chamadas a saldo disponível da classe *Conta* oriundas da classe *ControladorTransferencia*. Na coluna “responsabilidade do aspecto”, especificamos o papel do aspecto naquela interação, que, neste caso será *obter saldoDisponivelCPMF*. Finalizamos o preenchimento da tabela de composição nível 2 da classe de análise aspectual *CobrancaCPMF* após repetir esse procedimento para cada linha da Tabela 5.16. Ao final desse processo, obtivemos a Tabela 5.28.

Tabela 5.28 – Tabela de composição nível 2 para classe de análise aspectual *CobrancaCPMF*

CLASSE DE ANÁLISE ASPECTUAL: # CobrancaCPMF						
CASO DE USO AFETADO	CLASSE DE ANÁLISE AFETADA	COND. (OPC.)	RESPONSABILIDADE DO ASPECTO	REGRA DE COMPOSIÇÃO	PONTO JUNÇÃO (CLASSE/MENSAGEM)	INFORMAÇÕES ADICIONAIS
UC#04 - Realizar transferência	Controlador Transferencia	O dia da transação corresponder ao dia do débito periódico do CPMF	obter saldoDisponivelCPMF	override	Conta/ saldo disponível	saldoDisponivelCPMF = saldoDisponivel – CPMFfacumulado
UC#04 - Realizar transferência	Controlador Transferencia	-	acumular CPMF	overlap.after	Conta/ debitar	CPMFacumulado = CPMFfacumulado + (valorDebito*cpmf)
UC#05 - Pagar boleto bancário	Controlador Pagamento	O dia da transação corresponder ao dia do débito periódico do CPMF	obter saldoDisponivelCPMF	override	Conta/ saldo disponível	saldoDisponivelCPMF = saldoDisponivel – CPMFfacumulado
UC#05 - Pagar boleto bancário	Controlador Pagamento	-	acumular CPMF	overlap.after	Conta/ debitar	CPMFacumulado = CPMFfacumulado + (valorBoleto*cpmf)

O mesmo procedimento descrito acima foi realizado para preencher as tabelas de composição nível 2 das demais classes de análise aspectuais: *CobrancaTransacaoExtra*, *GravacaoInfoTransacao*, *Bonificacao*, *ChecagemSenhaIB*, *ChecagemDadosPessoais* e *VerificacaoLimite*. O resultado pode ser conferido respectivamente na Tabela 5.29, na Tabela 5.30, na Tabela 5.31, na Tabela 5.32, na Tabela 5.33 e na Tabela 5.34.

A Tabela 5.29 foi preenchida a partir da tabela de composição nível 1 do requisito aspectual *Cobrar tarifa por transação extra* (Tabela 5.14) e da análise dos

diagramas de colaboração dos casos de uso *Visualizar saldo de conta* (Figura 5.4) e *Visualizar extrato de conta* (Figura 5.5).

Tabela 5.29 – Tabela de composição nível 2 para classe de análise aspectual *CobrancaTransacaoExtra*

CLASSE DE ANÁLISE ASPECTUAL: # CobrancaTansacaoExtra						
CASO DE USO AFETADO	CLASSE DE ANÁLISE AFETADA	CONDIÇÃO (OPCIONAL)	RESPONSABILIDADE DO ASPECTO	REGRA DE COMPOSIÇÃO	PONTO JUNÇÃO (CLASSE/MENSAGEM)	INFO. ADIC.
UC#02 - Visualizar Saldo de conta	InterfaceSaldo	Quantidade máxima do tipo transação "visualizar saldo de conta" para a modalidade da conta naquele mês foi ultrapassada	cobrar tarifa	overlap.after	InterfaceSaldo / exibir saldo	-
UC#03 - Visualizar Extrato de conta	InterfaceExtrato	Quantidade máxima do tipo transação "visualizar extrato de conta" para a modalidade da conta naquele mês foi ultrapassada	cobrar tarifa	overlap.after	InterfaceExtrato/ exibir extrato	-

A Tabela 5.30 foi preenchida a partir da tabela de composição nível 1 do requisito aspectual *Gravar de informações da transação* (Tabela 5.15) e da análise dos diagramas de colaboração dos casos de uso *Visualizar saldo de conta* (Figura 5.4), *Visualizar extrato de conta* (Figura 5.5), *Realizar transferência* (Figura 5.6) e *Pagar boleto bancário*.

Tabela 5.30 – Tabela de composição nível 2 para classe de análise aspectual *GravacaoInfoTransacao*

CLASSE DE ANÁLISE ASPECTUAL: # GravacaoInfoTransacao						
CASO DE USO AFETADO	CLASSE DE ANÁLISE AFETADA	COND. (OPC.)	RESPONSABILIDADE DO ASPECTO	REGRA DE COMPOSIÇÃO	PONTO JUNÇÃO (CLASSE/ MENSAGEM)	INFO. ADIC.
UC#02 - Visualizar Saldo	InterfaceSaldo	-	registrar	overlap.after	InterfaceSaldo/ exibir saldo	-
UC#03 - Visualizar Extrato	InterfaceExtrato	-	registrar	overlap.after	InterfaceExtrato/ exibir extrato	-
UC#04 - Realizar Transferência	ControladorTransferencia	-	registrar	overlap.after	Conta / Creditar	-
UC#05 - Pagar boleto bancário	ControladorPagamento	-	registrar	overlap.after	ProcessadorConta/ processar	-

A Tabela 5.31 foi preenchida a partir da tabela de composição nível 1 do requisito aspectual *Oferecer bonificação* (Tabela 5.17) e da análise dos diagramas de colaboração dos casos de uso *Realizar transferência* (Figura 5.6) e *Pagar boleto bancário*.

Tabela 5.31 – Tabela de composição nível 2 para classe de análise aspectual *Bonificacao*

CLASSE DE ANÁLISE ASPECTUAL: # Bonificação						
CASO DE USO AFETADO	CLASSE DE ANÁLISE AFETADA	CONDIÇÃO (OPCIONAL)	RESPONSABILIDADE DO ASPECTO	REGRA DE COMPOSIÇÃO	PONTO JUNÇÃO (CLASSE/ MENSAGEM)	INFO. ADIC.
UC#04 - Realizar Transferência	Controlador Transferencia	A cada nova dezena de transações financeiras realizadas no mês.	cadastrar	overlap.after	Conta / Creditar	-
UC#05 - Pagar boleto bancário	Controlador Pagamento	A cada nova dezena de transações financeiras realizadas no mês.	cadastrar	overlap.after	ProcessadorConta/ processar	-

A Tabela 5.32 foi preenchida a partir da tabela de composição nível 1 do requisito aspectual *Checar senha Internet Banking* (Tabela 5.18) e da análise dos diagramas de colaboração dos casos de uso *Visualizar saldo de conta* (Figura 5.4), *Visualizar extrato de conta* (Figura 5.5), *Realizar transferência* (Figura 5.6) e *Pagar boleto bancário*.

Tabela 5.32 – Tabela de composição nível 2 para classe de análise aspectual *ChecagemSenhaIB*

CLASSE DE ANÁLISE ASPECTUAL: ChecagemSenhaIB						
CASO DE USO AFETADO	CLASSE DE ANÁLISE AFETADA	COND. (OPC.)	RESPONSABILIDADE DO ASPECTO	REGRA DE COMPOSIÇÃO	PONTO JUNÇÃO (CLASSE/ MENSAGEM)	INFORMAÇÕES ADICIONAIS
UC#02 - Visualizar Saldo	InterfaceSaldo	-	cheçar	wrap	ControladorSaldo/ solicitar saldo	Se senhaOk então execute(pontojunção) Senão exibe(msgErro)
UC#03 - Visualizar Extrato	InterfaceExtrato	-	cheçar	wrap	ControladorExtrato/ solicitar extrato	Se senhaOk então execute(pontojunção) Senão exibe(msgErro)
UC#04 - Realizar Transferência	Interface Transferencia	-	cheçar	wrap	ControladorTransferencia/ solicitar transferência	Se senhaOk então execute(pontojunção) Senão exibe(msgErro)
UC#05 - Pagar boleto bancário	Interface Pagamento	-	cheçar	wrap	ControladorPagamento/ solicitar pagamento	Se senhaOk então execute(pontojunção) Senão exibe(msgErro)

A Tabela 5.33 foi preenchida a partir da tabela de composição nível 1 do requisito aspectual *Checar dados pessoais do cliente* (Tabela 5.19) e da análise dos diagramas de colaboração dos casos de uso *Realizar transferência* (Figura 5.6) e *Pagar boleto bancário*.

Tabela 5.33 – Tabela de composição nível 2 para classe de análise aspectual
ChecagemDadosPessoais

CLASSE DE ANÁLISE ASPECTUAL: <i>ChecagemDadosPessoais</i>						
CASO DE USO AFETADO	CLASSE DE ANÁLISE AFETADA	COND. (OPC.)	RESPONSABILIDADE DO ASPECTO	REGRA DE COMPOSIÇÃO	PONTO JUNÇÃO (CLASSE/ MENSAGEM)	INFORMAÇÕES ADICIONAIS
UC#04 - Realizar Transferência	InterfaceTransferencia	-	chechar	wrap	InterfaceTransferencia/ exibir dados confirmação	Se dadosOk então execute(pontojunção) Senão exibe(msgErro)
UC#05 - Pagar boleto bancário	InterfacePagamento	-	chechar	wrap	InterfacePagamento/ exibir dados confirmação	Se dadosOk então execute(pontojunção) Senão exibe(msgErro)

A Tabela 5.34 foi preenchida a partir da tabela de composição nível 1 do requisito aspectual *Limitar valor de movimentação* (Tabela 5.20) e da análise dos diagramas de colaboração dos casos de uso *Realizar transferência* (Figura 5.6) e *Pagar boleto bancário*.

Tabela 5.34 – Tabela de composição nível 2 para classe de análise aspectual *VerificacaoLimite*

CLASSE DE ANÁLISE ASPECTUAL: <i>VerificacaoLimite</i>						
CASO DE USO AFETADO	CLASSE DE ANÁLISE AFETADA	COND. (OPC.)	RESPONSABILIDADE DO ASPECTO	REGRA DE COMPOSIÇÃO	PONTO JUNÇÃO (CLASSE/ MENSAGEM)	INFORMAÇÕES ADICIONAIS
UC#04 - Realizar Transferência	Interface Transferencia	-	verificar	wrap	ControladorTransferencia/ solicitar transferência	Se limiteOk então execute(pontoJunção) Senão exibe(msgErro)
UC#05 - Pagar boleto bancário	Interface Pagamento	-	verificar	wrap	ControladorPagamento/ solicitar pagamento	Se limiteOk então execute(pontoJunção) Senão exibe(msgErro)

VISÃO DOS DIAGRAMAS DE COLABORAÇÃO COM A PRESENÇA DE OBJETOS DE ANÁLISE ASPECTUAIS

Com as tabelas de composição nível 2 preenchidas, é possível gerar uma visão do diagrama de colaboração do caso de uso no qual os objetos de análise aspectuais que afetam a interação estão presentes.

A Figura 5.7, por exemplo, reflete a visão do diagrama de colaboração do caso de uso *Visualizar saldo de conta* com a presença dos objetos de análise aspectuais que afetam a colaboração. Essa visão foi gerada a partir do diagrama da Figura 5.4 e das tabelas de composição nível 2 das classes de análise aspectuais que afetam o caso de uso *Visualizar saldo de conta* (Tabela 5.29, Tabela 5.30 e Tabela 5.32).

É importante frisar que, essa visão dá uma idéia de como essa colaboração seria modelada caso não estivéssemos utilizando nossa abordagem para proporcionar a

separação explícita de preocupações transversais. Comparando esse diagrama com o diagrama da Figura 5.4, que enfatiza apenas a essência do caso de uso *Visualizar saldo de conta*, percebe-se como a colaboração do diagrama da Figura 5.7 é entrelaçada com objetos que não contribuem para realização do objetivo essencial do caso de uso *Visualizar saldo de conta*. Além de inserir complexidade extra nesse diagrama, esse entrelaçamento dificulta o reuso e a manutenção dessa colaboração, conforme já discutido na Seção 5.2.5.2.

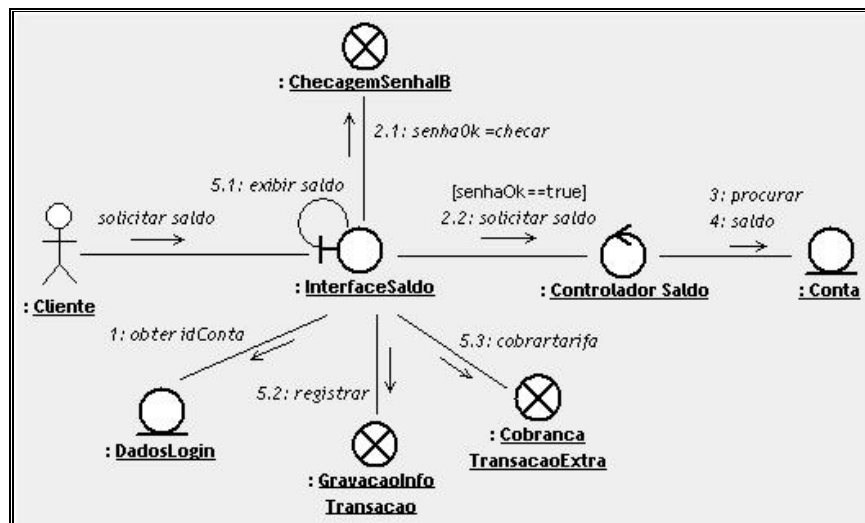


Figura 5.7 - Visão do diagrama de colaboração do caso de uso *Visualizar saldo de conta* COM a presença dos objetos de análise aspectuais

A Figura 5.8 apresenta a visão do diagrama de colaboração do caso de uso *Visualizar extrato de conta* com a presença dos objetos de análise aspectuais que afetam esse caso de uso. Essa visão foi gerada a partir do diagrama da Figura 5.5 e das tabelas de composição nível 2 das classes de análise aspectuais que afetam o caso de uso *Visualizar extrato de conta* (Tabela 5.29, Tabela 5.30 e Tabela 5.32).

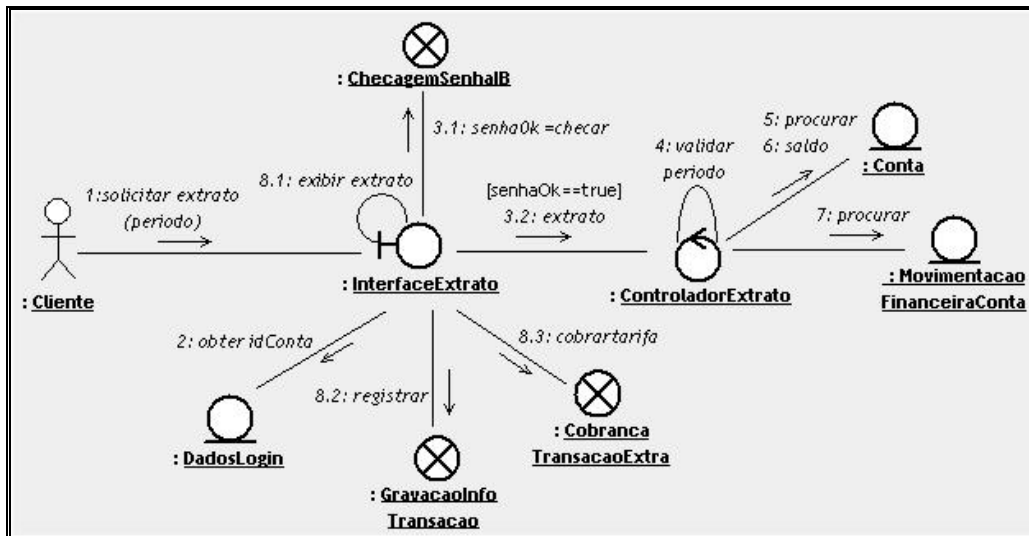


Figura 5.8 - Visão do diagrama de colaboração do caso de uso *Visualizar extrato de conta* COM a presença dos objetos de análise aspectuais

Quanto maior o número de preocupações transversais que precisam ser inseridas num caso de uso, mais crítico será o impacto de não separar preocupações transversais. Comparando o diagrama da Figura 5.6, que enfatiza apenas a essência do caso de uso *Realizar transferência*, com a visão desse diagrama incluindo os objetos de análise aspectuais (Figura 5.9), a qual dá uma idéia de como essas preocupações seriam tratadas sem nossa abordagem, pode-se perceber que esse último diagrama é muito mais difícil de compreender, reusar e manter do que o da Figura 5.6.

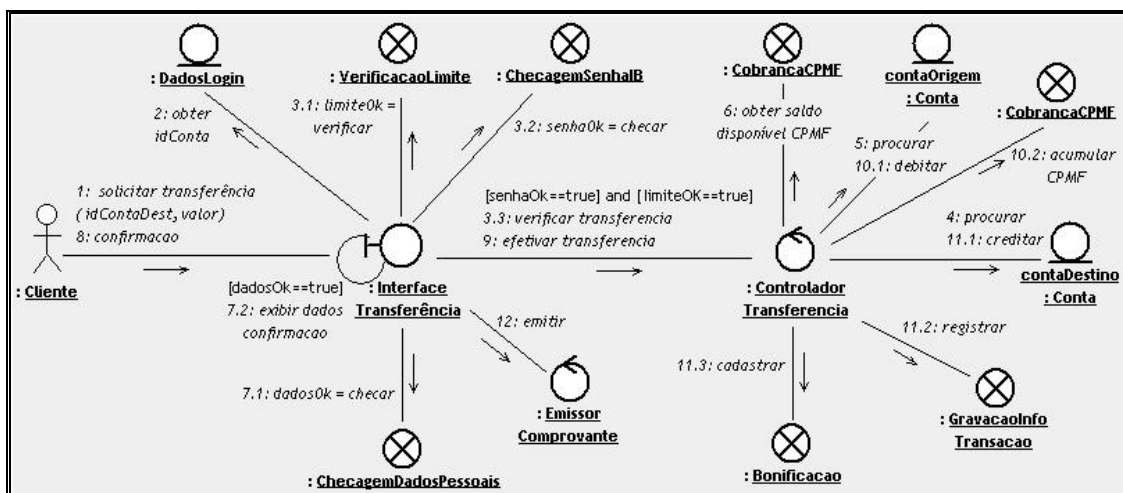


Figura 5.9 - Visão do diagrama de colaboração do caso de uso *Realizar transferência* COM a presença dos objetos de análise aspectuais

5.3.2 Analisar classes

Nesta atividade, analisamos cada classe de análise aspectual em mais detalhes, verificando quais responsabilidades elas agregam e como elas realizam essas responsabilidades. No nosso estudo de caso, as classes de análise aspectuais identificadas incluem (vide Tabela 5.27): *CobrancaTransacaoExtra*, *GravacaoInfoTransacao*, *CobrancaCPMF*, *Bonificação*, *ChecagemSenhaIB*, *ChecagemDadosPessoais* e *VerificacaoLimite*.

Iniciando pela classe de análise aspectual *CobrancaTransacaoExtra*, a partir da Tabela 5.29, identificamos que, em todos os casos de uso em que deve ser inserida, essa classe de análise possui a responsabilidade de “cobrar tarifa”. Para realizar essa responsabilidade, um objeto da classe *CobrancaTransacaoExtra* precisará interagir com um objeto da classe de *TransacoesConta* para verificar o número de transações de determinado tipo que foram realizadas no mês; caso esse número ultrapasse a quantidade permitida para modalidade da conta, o objeto de análise aspectual disparará uma mensagem para que seja debitada a tarifa da conta. Essa interação está representada na Figura 5.10.

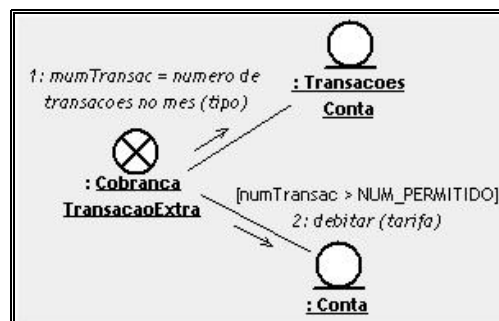


Figura 5.10 - Realização da responsabilidade "cobrar tarifa" da classe de análise aspectual *CobrancaTransacaoExtra*

A única classe de análise aspectual com mais de uma responsabilidade que foi identificada nesse estudo de caso foi a classe *CobrancaCPMF*. De acordo com a tabela de nível 2 dessa classe (Tabela 5.28), ela possui duas responsabilidades nas interações em que deve ser inserida: “acumular CPMF” e “obter saldo disponível CPMF”. Para realizar a primeira responsabilidade, um objeto da classe *CobrancaCPMF* precisa calcular o CPMF da transação e, em seguida, acumular esse valor num registro de CPMF acumulado para cada conta (Figura 5.11). Por sua vez, para realizar a

responsabilidade “obter saldo disponível CPMF” um objeto da classe *CobrancaCPMF* deve consultar o saldo disponível na conta e o saldo de CPMF acumulado e, por fim, calcular o saldo disponível na conta descontando o CPMF acumulado (Figura 5.12).

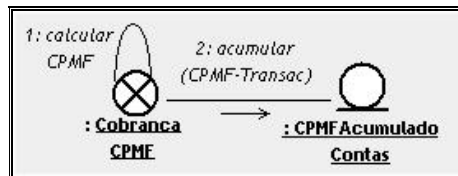


Figura 5.11 - Realização da responsabilidade "cobrar CPMF" da classe de análise aspectual *CobrancaCPMF*

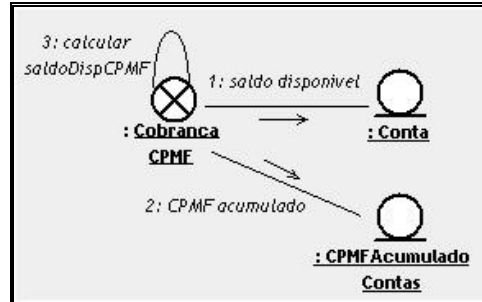


Figura 5.12 - Realização da responsabilidade "obter saldo disponível CPMF" da classe de análise aspectual *CobrancaCPMF*

A responsabilidade da classe de análise aspectual *VerificacaoLimite* corresponde apenas a um teste para verificar se o valor da transação supera o valor permitido. Por sua simplicidade, a responsabilidade da classe de análise aspectual *VerificacaoLimite* foi a única responsabilidade cuja realização não exigiu interação com outras classes de análise. As interações necessárias para realizar as responsabilidades das demais classes de análise são ilustradas na Figura 5.13, na Figura 5.14, na Figura 5.15 e na Figura 5.16.

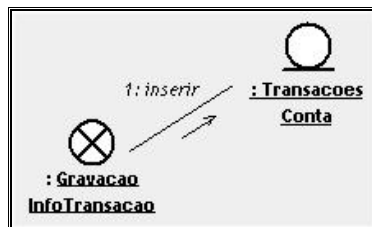


Figura 5.13 - Realização da responsabilidade "registrar" da classe de análise aspectual *GravacaoInfoTransacao*

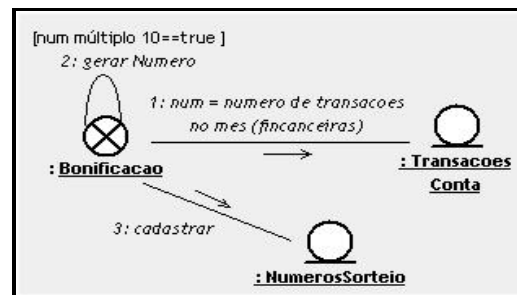


Figura 5.14 - Realização da responsabilidade "cadastrar" da classe de análise aspectual *Bonificacao*

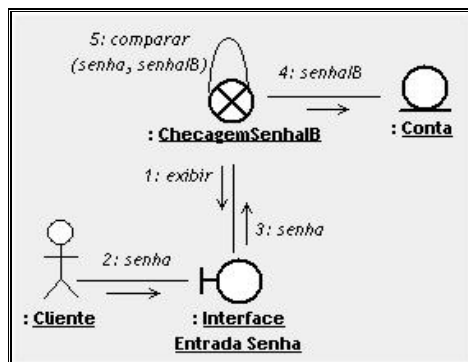


Figura 5.15 - Realização da responsabilidade "chechar" da classe de análise aspectual *ChecagemSenhaIB*

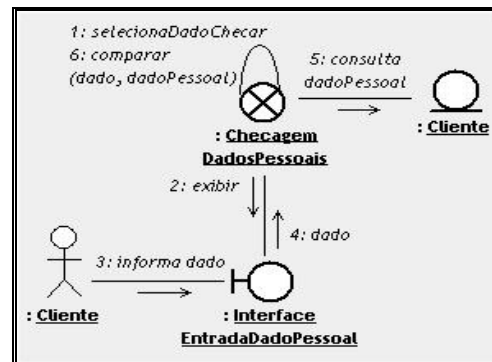


Figura 5.16 - Realização da responsabilidade "chechar" da classe de análise aspectual *ChecagemDadosPessoais*

5.4 Fluxo de projeto

Tendo concluído os fluxos de requisitos e análise, neste fluxo refinamos o modelo de análise, considerando que (i) iremos utilizar a tecnologia Java [GOSLING et al., 2000] e interface baseada em páginas HTML para implementar as entidades de projeto; e (ii) o sistema será projetado de acordo com o padrão arquitetural PDC [MASSONI et al., 2001].

A princípio, definimos como os casos de uso e requisitos aspectuais serão concretizados em termos de interações entre classes de projeto e/ou subsistemas (Seção 5.4.1). O passo seguinte foi identificar operações, atributos e relacionamentos para cada classe de projeto aspectual identificada (5.4.2).

5.4.1 Projetar casos de uso

Esta atividade é subdividida em duas partes. Na primeira, identificamos as entidades de projeto que serão utilizadas na realização de cada caso de uso e cada requisito aspectual (Seção 5.4.1.1). Na segunda parte, descrevemos como os objetos dessas classes interagem para realizar o caso de uso e como os objetos das classes de projeto aspectuais devem ser inseridos nessas interações (Seção 5.4.1.2).

5.4.1.1 IDENTIFICAR E ESPECIFICAR ENTIDADES DE PROJETO

Nesta parte do processo, mapeamos as classes de análise necessárias para realizar os casos de uso e os requisitos aspectuais em elementos de projeto, considerando a tecnologia adotada. A Tabela 5.35 exibe o resultado do mapeamento que adotamos. As células em destaque representam classes de análise/projeto aspectuais. Em resumo, a estratégia que seguimos para fazer esse mapeamento foi a seguinte:

- conforme sugerido pelo padrão arquitetural PDC [MASSONI et al., 2001]:
 - ✓ criamos uma classe para ser o ponto de acesso aos serviços do sistema, denominada de *FachadaBanco*;
 - ✓ cada uma das classes de análise do tipo entidade (tais como: *Conta*, *Cliente*, *CPMFAcumuladoConta* e *NumerosSorteio*) foram mapeadas em pelo menos quatro entidades de projeto: (i) uma classe básica com atributos e operações (ex: *Conta*) (ii) uma classe que conterá métodos para inserir/consultar/atualizar/excluir objetos, com verificação de regras de integridade (ex: *CadastroConta*); (iii) uma classe responsável por manipular uma coleção de objetos persistentes (ex: *RepositorioContaBDR*); (iv) uma interface entre a classe de cadastro e o repositório (ex: *IREpositorioConta*);
- classes de análise do tipo fronteira e controle foram subdivididas/fundidas em classes de projeto de acordo com as suas responsabilidades;
- classes de análise aspectuais (*CobrancaTransacaoExtra*, *GravacaoInfoTransacao*, *CobrancaCPMF*, *Bonificação*, *ChecagemSenhaIB*, *ChecagemDadosPessoais* e *VerificacaoLimite*) foram mapeadas em classes de projeto aspectuais.

Tabela 5.35 – Mapeamento entre classes de análise e classes de projeto

	CLASSES DE ANÁLISE	ENTIDADES DE PROJETO
CLASSES DE PROJETO QUE CONTRIBUEM PARA REALIZAÇÃO DE CASOS DE USO	-	FachadaBanco
	Conta	CadastroConta IRepositorioConta RepositorioContaBDR Conta
	ControladorExtrato	ControladorTransacao
	ControladorPagamento	
	ControladorSaldo	
	ControladorTransferencia	
	DadosLogin	SessaoUsuario
	EmissorComprovante	EmissorComprovante
	InterfaceExtrato	FormularioEntradaExtrato ServletExibirExtrato
	InterfacePagamento	FormularioEntradaPgto ServletConfirmDadosPgto
	InterfaceSaldo	ServletExibirSaldo
	InterfaceTransferencia	FormularioEntradaTransf ServletConfirmDadosTransf
	MovimentacaoFinanceiraConta	ListaMovFinConta
	ProcessadorConta	ProcessadorConta
CLASSES DE PROJETO QUE CONTRIBUEM PARA REALIZAÇÃO DE REQUISITOS ASPECTUAIS	Bonificacao	Bonificacao
	ChecagemDadosPessoais	ChecagemDadosPessoais
	ChecagemSenhaIB	ChecagemSenhaIB
	Cliente	CadastroCliente IRepositorioCliente RepositorioClienteBDR Cliente
	CobrancaCPMF	CobrancaCPMF
	CobrancaTransacaoExtra	CobrancaTransacaoExtra
	Conta	CadastroConta IRepositorioConta RepositorioContaBDR Conta
	CPMFAcumuladoConta	CadastroCPMFAcumuladoConta IRepositorioCPMFAcumuladoConta RepositorioCPMFAcumuladoContaBDR CPMFAcumuladoConta
	GravacaoInfoTransacao	GravacaoInfoTransacao
	InterfaceEntradaDadoPessoal	FormularioDadoPessoal
	InterfaceEntradaSenhaIB	FormularioSenhaIB
	NumerosSorteio	CadastroNumeroSorteio IRepositorioNumeroSorteio RepositorioNumeroSorteioBDR NumeroSorteio
	TransacoesConta	CadastroTransacao IRepositorioTransacao RepositorioTransacaoBDR ITransacao TransacaoConsulta TransacaoFinanceira TransacaoSaldo TransacaoExtrato TransacaoTransferencia TransacaoPagamento
	VerificacaoLimite	VerificacaoLimite

5.4.1.2 DESCREVER INTERAÇÕES ENTRE ENTIDADES DE PROJETO

Essa atividade visa analisar como um requisito aspectual (neste ponto do processo representado por uma classe de projeto aspectual) impactará a interação dos casos de uso que ele afeta em termos de entidades de projeto. Para isso, inicialmente descrevemos as interações entre objetos de projeto que concretizam os casos de uso. A seguir, para cada requisito aspectual, analisamos como a instância da classe de projeto aspectual que representa esse requisito deverá ser inserida nas interações dos casos de uso afetados e, se isso exige alguma adição estrutural em alguma classe que participa da interação; essas informações são especificadas em tabelas de composição nível 3. Por fim, a partir das informações contidas nas tabelas de composição nível 3, podemos gerar uma visão dos diagramas de seqüência dos casos de uso com a presença dos objetos aspectuais que afetam o caso de uso.

DESCREVER INTERAÇÕES ENTRE ENTIDADES DE PROJETO PARA REALIZAÇÃO DE CASOS DE USO

Tendo conhecimento das classes de projeto necessárias para cada caso de uso (Tabela 5.35), descrevemos em diagramas de seqüência como será a interação das instâncias de entidades de projeto para realizar cada caso de uso. Para auxiliar nesse processo, usamos como ponto de referência as especificações dos casos de uso obtidas no fluxo de requisitos (Seção 5.2.5.2), os diagramas de colaboração obtidos no fluxo de análise (Seção 5.3.1.2) e o mapeamento entre classes de análise e entidades de projeto (Seção 5.4.1.1, Tabela 5.35).

A Figura 5.17 apresenta o diagrama de seqüência para o caso de uso *Visualizar saldo*

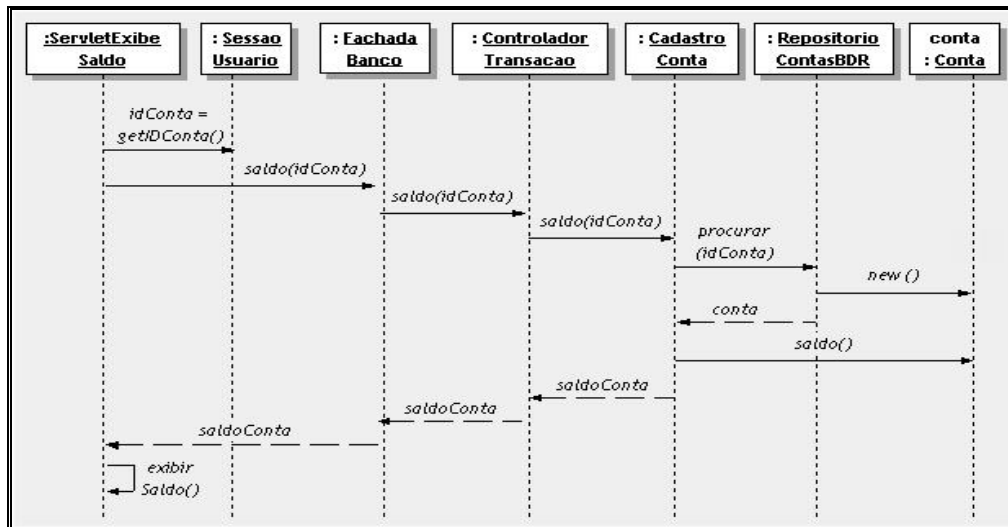


Figura 5.17 – Diagrama de sequência do caso de uso *Visualizar saldo de conta*

As interações entre instâncias de entidades de projeto para realizar o caso de uso *Visualizar extrato de conta* são apresentadas no diagrama de sequência da Figura 5.18.

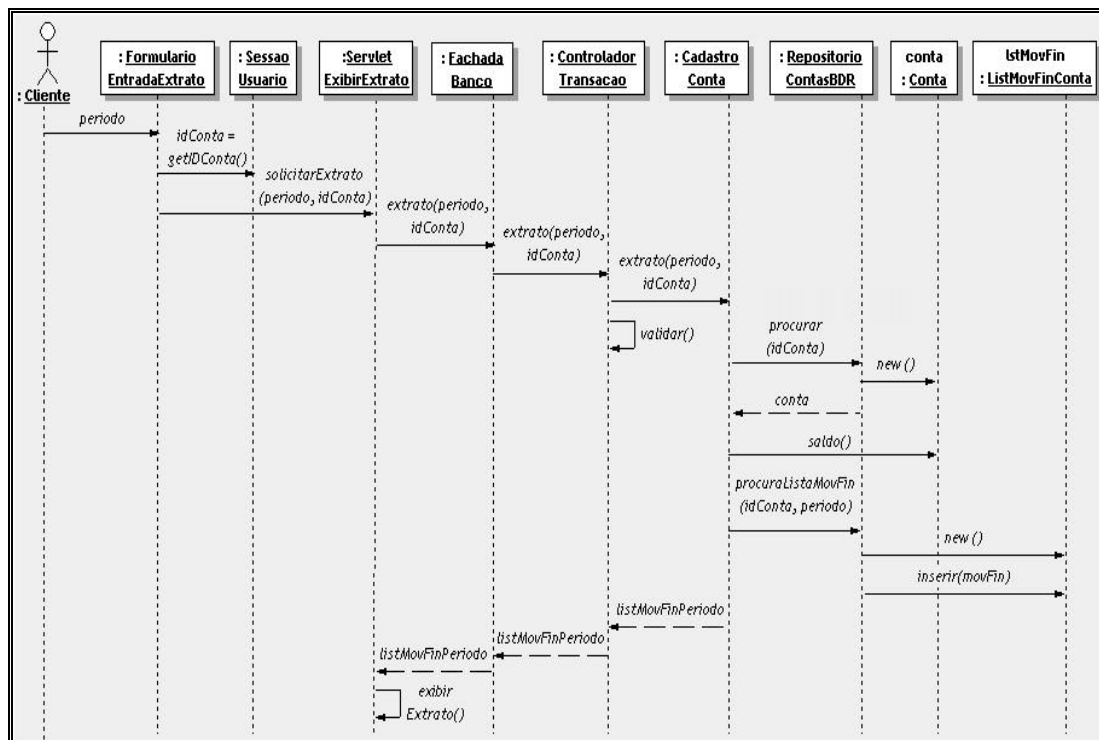


Figura 5.18 - Diagrama de sequência do caso de uso *Visualizar extrato de conta*

A realização do caso de uso *Realizar transferência* abrange um grande número de interações entre objetos. Para simplificar a visualização dessas interações, dividimos

o cenário principal desse caso de uso em duas partes. A primeira parte vai desde o início da execução do caso de uso até a exibição de dados para confirmação (Figura 5.19). A segunda parte modela desde a confirmação enviada pelo cliente, até o final da execução do caso de uso (Figura 5.20).

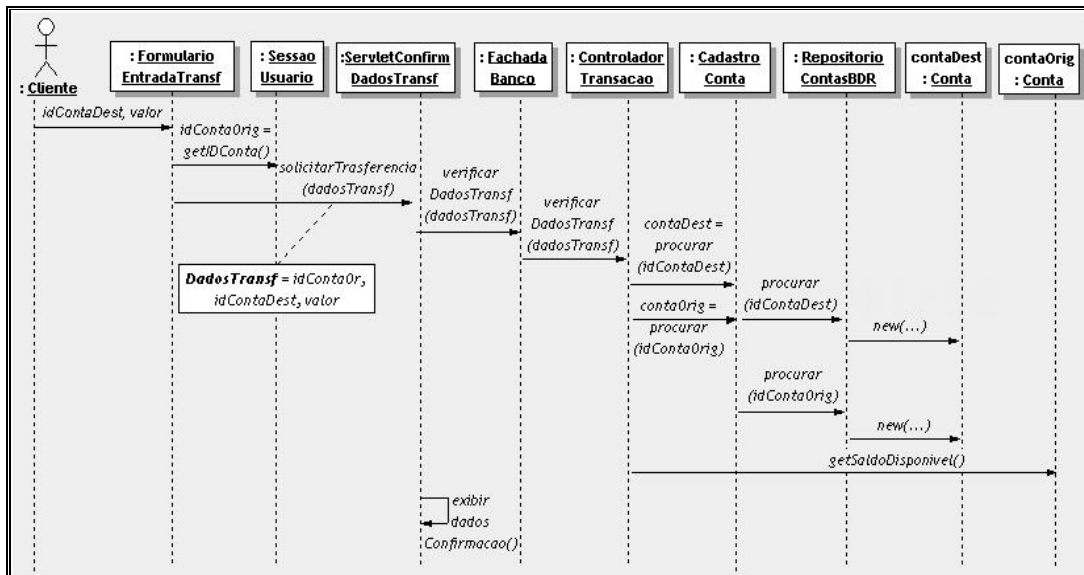


Figura 5.19 – Diagrama de sequência para a primeira parte do caso de uso *Realizar transferência*

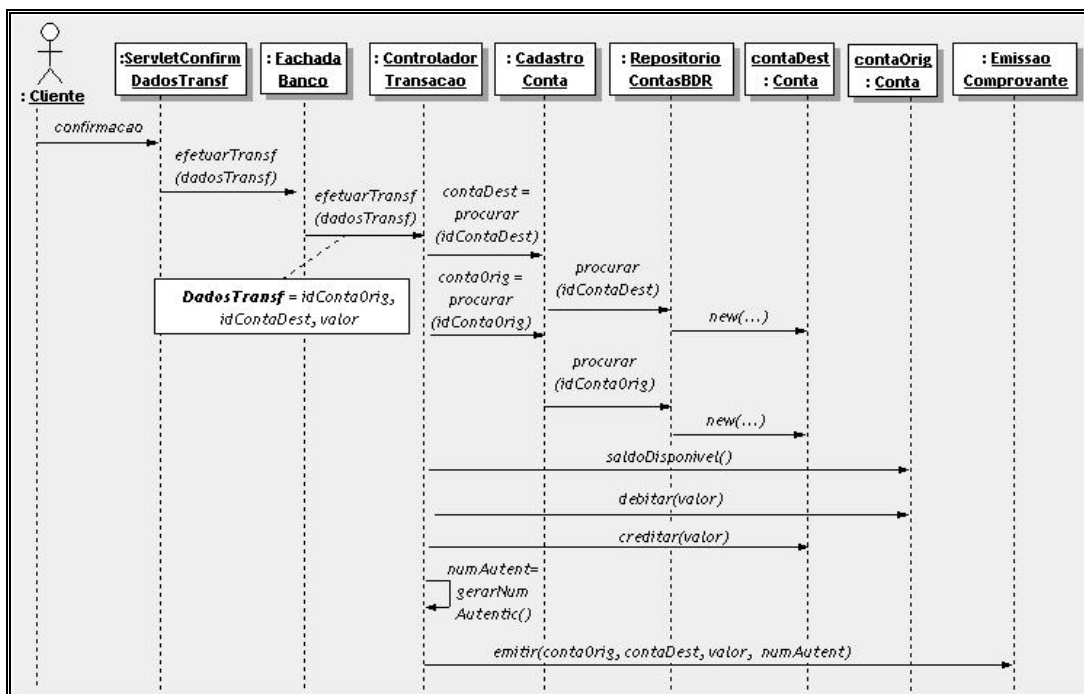


Figura 5.20 - Diagrama de sequência para a segunda parte do caso de uso *Realizar transferência*

ESPECIFICAR TABELAS DE COMPOSIÇÃO NÍVEL 3

Para definir como as instâncias de classes de projeto aspectuais deveriam ser inseridas nas interações entre entidades de projeto, preenchemos uma tabela de composição nível 3 para cada classe de projeto aspectual (*CobrancaTransacaoExtra*, *GravacaoInfoTransacao*, *CobrancaCPMF*, *Bonificação*, *ChecagemSenhaIB*, *ChecagemDadosPessoais* e *VerificacaoLimite*).

O preenchimento das tabelas de composição nível 3 seguiu o roteiro apresentado na Figura 4.17 e teve como principais pontos de referência: (i) as tabelas de composição nível 2 (Seção 5.3.1.2); (ii) o mapeamento entre classes de análise e entidades de projeto (Tabela 5.35); (iii) os diagramas de interação (seqüência) entre entidades de projeto dos casos de uso obtidos no passo imediatamente anterior a esse.

Para preencher a tabela de composição nível 3 da classe de projeto aspectual *CobrancaCPMF*, inicialmente recorreremos às informações tabela de nível 2 correspondente (Tabela 5.28). A primeira linha dessa tabela indica que uma instância da classe *CobrancaCPMF* deve ser inserida na interação do caso de uso *Realizar transferência* (caso de uso afetado) para que sempre que o *ControladorTransferencia* (classe de análise afetada) enviar uma mensagem para *Conta* para consultar o saldo disponível (ponto de junção), essa consulta seja substituída (regra de composição *override*) e o resultado passa a ser o valor do saldo disponível, descontando o valor do CPMF acumulado. A partir dessas informações, criamos uma linha na tabela de composição nível 3 (vide Tabela 5.36) e preenchemos as colunas “caso de uso afetado”, “condição”, “regra de composição” e “informações adicionais”. Analisando o mapeamento entre classes de análise e classes de projeto (Tabela 5.35), preenchemos a coluna “classe de projeto afetada” com a classe de *ControladorTransacao*. Definimos que a operação `saldoDisponivelCPMF:double` concretizará a responsabilidade indicada na tabela de nível 2; por isso, esse será o conteúdo da coluna “comportamento transversal”. A coluna “ponto de junção” na tabela de nível 3 será `Conta.saldoDisponivel()`, pois está é mensagem no diagrama de seqüência que corresponde ao ponto de junção especificado na tabela de nível 2. A necessidade de haver alguma adição estrutural será avaliada na atividade “Projetar classes” e, caso isso seja necessário, o preenchimento dessa tabela será completado nessa atividade.

Finalizamos o preenchimento da tabela de composição nível 3 da classe de projeto aspectual *CobrancaCPMF* após repetir esse procedimento para cada linha da Tabela 5.28. Ao final desse processo, obtivemos a Tabela 5.36.

Tabela 5.36 – Tabela de composição nível 3 para classe de projeto aspectual *CobrancaCPMF*

CLASSE DE PROJETO ASPECTUAL: CobrançaCPMF							
CASO DE USO AFETADO	CLASSE PROJETO AFETADA	CONDIÇÃO	ADIÇÕES PROVIDAS PELA CLASSE DE PROJETO ASPECTUAL				INFORMAÇÕES ADICIONAIS
			ADIÇÃO COMPORTAMENTAL			ADIC. ESTR.	
			COMPORTAMENTO TRANSVERSAL	REGRA COMPOSIÇÃO	PONTO DE JUNÇÃO		
UC#04 - Realizar transferência	Controlador Transacao	O dia da transação corresponder ao dia do débito periódico do CPMF	saldoDisponivelCPMF (conta: Conta) : double	override	Conta. saldoDisponivel()	-	saldoDisponivelCPMF = saldoDisponivel - CPMF acumulado
UC#04 - Realizar transferência	Controlador Transacao	-	acumularCPMF (valorTransac: double)	overlap.after	Conta. debitar(valor: double)	-	CPMFacumulado = CPMFacumulado + (valorDebito*cpmf)
UC#05 - Pagar boleto bancário	Controlador Transacao	O dia da transação corresponder ao dia do débito periódico do CPMF	saldoDisponivelCPMF (conta: Conta) : double	override	Conta. saldoDisponivel()	-	saldoDisponivelCPMF = saldoDisponivel - CPMF acumulado
UC#05 - Pagar boleto bancário	Controlador Transacao	-	acumularCPMF (idConta: string; valorTransac: double)	overlap.after	Conta.debitar (valor: double)	-	CPMFacumulado = CPMFacumulado + (valorBoleto*cpmf)

O mesmo procedimento descrito acima foi realizado para preencher as tabelas de composição nível 3 das demais classes de análise aspectuais: *CobrancaTransacaoExtra*, *GravacaoInfoTransacao*, *Bonificacao*, *ChecagemSenhaIB*, *ChecagemDadosPessoais* e *VerificacaoLimite*. O resultado pode ser conferido respectivamente na Tabela 5.37, Tabela 5.38, na Tabela 5.39, na Tabela 5.40, na Tabela 5.41 e na Tabela 5.42.

A Tabela 5.37 foi preenchida a partir da tabela de composição nível 2 da classe de análise aspectual *CobrancaTransacaoExtra* (Tabela 5.29) e da análise dos diagramas de sequência dos casos de uso *Visualizar saldo de conta* (Figura 5.17) e *Visualizar extrato de conta* (Figura 5.18).

Tabela 5.37 -Tabela de composição nível 3 para classe de projeto aspectual *CobrancaTransacaoExtra*

CLASSE DE PROJETO ASPECTUAL: CobrancaTransacaoExtra							
CASO DE USO AFETADO	CLASSE PROJETO AFETADA	COND.	ADIÇÕES PROVIDAS PELA CLASSE DE PROJETO ASPECTUAL				INFO ADIC.
			ADIÇÃO COMPORTAMENTAL			ADIC. ESTRUT.	
			COMPORTAMENTO TRANSVERSAL	REGRA COMPOSIÇÃO	PONTO DE JUNÇÃO		
UC#02 - Visualizar Saldo de conta	ServletExibe Saldo	Quantidade máxima do tipo transação "visualizar saldo de conta" para a modalidade da conta naquele mês foi ultrapassada	cobrarTarifa (idConta: string)	overlap.after	ServletExibeSaldo. exibirSaldo()	-	-
UC#03 - Visualizar Extrato de conta	ServletExibe Extrato	Quantidade máxima do tipo transação "visualizar extrato de conta" para a modalidade da conta naquele mês foi ultrapassada	cobrarTarifa (idConta: string)	overlap.after	ServletExibeExtrato. exibirExtrato()	-	-

A Tabela 5.38 foi preenchida a partir da tabela de composição nível 2 da classe de análise aspectual *GravacaoInfoTransacao* (Tabela 5.30) e da análise dos diagramas de sequência dos casos de uso *Visualizar saldo de conta* (Figura 5.17), *Visualizar extrato de conta* (Figura 5.18), *Realizar transferência* (Figura 5.20) e *Pagar boleto bancário*.

Tabela 5.38 - Tabela de composição nível 3 para classe de projeto aspectual *GravacaoInfoTransacao*

CLASSE DE PROJETO ASPECTUAL: GravaçãoInfoTransação							
CASO DE Uso AFETADO	CLASSE PROJETO AFETADA	COND.	ADIÇÕES PROVIDAS PELA CLASSE DE PROJETO ASPECTUAL				INFO ADIC.
			ADIÇÃO COMPORTAMENTAL			ADIC. ESTRUT.	
			COMPORTAMENTO TRANSVERSAL	REGRA COMPOSIÇÃO	PONTO DE JUNÇÃO		
UC#02 - Visualizar Saldo	ServletExibeSaldo	-	registrar (info:TransacSaldo)	overlap.after	ServletExibeSaldo. exibirSaldo()	-	-
UC#03 - Visualizar Extrato	ServletExibeExtrato	-	registrar (info:TransacExtrato)	overlap.after	ServletExibeExtrato. exibirExtrato()	-	-
UC#04 - Realizar Transferência	ControladorTransacao	-	registrar (info:TransacTransf)	overlap.after	Conta. creditar(valor: double)	-	-
UC#05 - Pagar boleto bancário	ControladorTransacao	-	registrar (info:TransacPgto)	overlap.after	ProcessamentoBoleto. processar()	-	-

A Tabela 5.39 foi preenchida a partir da tabela de composição nível 2 da classe de análise aspectual *Bonificacao* (Tabela 5.31) e da análise dos diagramas de sequência dos casos de uso *Realizar transferência* (Figura 5.20) e *Pagar boleto bancário*.

Tabela 5.39 – Tabela de composição nível 3 para classe de projeto aspectual *Bonificacao*

CLASSE DE PROJETO ASPECTUAL: Bonificacao							
CASO DE Uso AFETADO	CLASSE PROJETO AFETADA	COND.	ADIÇÕES PROVIDAS PELA CLASSE DE PROJETO ASPECTUAL				INFO ADIC.
			ADIÇÃO COMPORTAMENTAL			ADIC. ESTRUT.	
			COMPORTAMENTO TRANSVERSAL	REGRA COMPOSIÇÃO	PONTO DE JUNÇÃO		
UC#04 - Realizar Transferência	ControladorTransacao	A cada nova dezena de transações financeiras realizadas no mês.	cadastrarBonus (idConta: string)	overlap.after	Conta.creditar (valor: double)	-	-
UC#05 - Pagar boleto bancário	ControladorTransacao	A cada nova dezena de transações financeiras realizadas no mês.	cadastrarBonus (idConta: string)	overlap.after	Processamento Boleto.processar()	-	-

A Tabela 5.40 foi preenchida a partir da tabela de composição nível 2 da classe de análise aspectual *ChecagemSenhaIB* (Tabela 5.32) e da análise dos diagramas de sequência dos casos de uso *Visualizar saldo de conta* (Figura 5.17), *Visualizar extrato de conta* (Figura 5.18), *Realizar transferência* (Figura 5.19) e *Pagar boleto bancário*.

Tabela 5.40 – Tabela de composição nível 3 para classe de projeto aspectual *ChecagemSenhaIB*

CLASSE DE PROJETO ASPECTUAL: ChecagemSenhaIB							
CASO DE Uso AFETADO	CLASSE PROJETO AFETADA	COND.	ADIÇÕES PROVIDAS PELA CLASSE DE PROJETO ASPECTUAL				INFO ADICIONAL
			ADIÇÃO COMPORTAMENTAL			ADIC. ESTRUT.	
			COMPORTAMENTO TRANSVERSAL	REGRA COMPOSIÇÃO	PONTO DE JUNÇÃO		
UC#02 - Visualizar Saldo	ServletExibe Saldo	-	verificarSenha (idConta: string) : boolean	wrap	FachadaBanco.saldo (idConta: string)	-	Se senhaOk então execute(pontojunção) Senão exibe(msgErro)
UC#03 - Visualizar Extrato	ServletExibe Extrato	-	verificarSenha (idConta: string) : boolean	wrap	FachadaBanco.extrato (dataI: Date; idConta: string)	-	Se senhaOk então execute(pontojunção) Senão exibe(msgErro)
UC#04 - Realizar Transferência	ServletConfirm DadosTransf	-	verificarSenha (idConta: string) : boolean	wrap	FachadaBanco. verificarDadosTransf (idContaOr: string; idContaDest:string; valor: double)	-	Se senhaOk então execute(pontojunção) Senão exibe(msgErro)
UC#05 - Pagar boleto bancário	ServletConfirm DadosPgto	-	verificarSenha (idConta: string) : boolean	wrap	FachadaBanco. verificarDadosPgto (idConta: string; codBarras: string; data: Date; valor: double)	-	Se senhaOk então execute(pontojunção) Senão exibe(msgErro)

A Tabela 5.41 foi preenchida a partir da tabela de composição nível 2 da classe de análise aspectual *ChecagemDadosPessoais* (Tabela 5.33) e da análise dos diagramas de sequência dos casos de uso *Realizar transferência* (Figura 5.19) e *Pagar boleto bancário*.

Tabela 5.41 – Tabela de composição nível 3 para classe de projeto aspectual
ChecagemDadosPessoais

CLASSE DE PROJETO ASPECTUAL: ChecagemDadosPessoais							
CASO DE USO AFETADO	CLASSE PROJETO AFETADA	COND.	ADIÇÕES PROVIDAS PELA CLASSE DE PROJETO ASPECTUAL				INFORMAÇÕES ADICIONAIS
			ADIÇÃO COMPORTAMENTAL			ADIC. ESTR.	
			COMPORTAMENTO TRANSVERSAL	REGR COMPOSIÇÃO	PONTO DE JUNÇÃO		
UC#04 - Realizar Transferênci a	ServletConfirm DadosTransf	-	checarDadPessoais (idConta: string): boolean	wrap	ServletConfirmDados Transf. exibirDadosConfirm()	-	Se dadosOk então execute(pontojunçã o) Senão exibe(msgErro)
UC#05 - Pagar boleto bancário	ServletConfirm DadosPgto	-	checarDadPessoais (idConta: string): boolean	wrap	ServletConfirmDados Pgto exibirDadosConfirm()	-	Se dadosOk então execute(pontojunçã o) Senão exibe(msgErro)

A Tabela 5.42 foi preenchida a partir da tabela de composição nível 2 da classe de análise aspectual *VerificacaoLimite* (Tabela 5.34) e da análise dos diagramas de sequência dos casos de uso *Realizar transferência* (Figura 5.19) e *Pagar boleto bancário*.

Tabela 5.42 – Tabela de composição nível 3 para classe de análise aspectual *VerificacaoLimite*

CLASSE DE PROJETO ASPECTUAL: VerificacaoLimite							
CASO DE USO AFETADO	CLASSE PROJETO AFETADA	COND.	ADIÇÕES PROVIDAS PELA CLASSE DE PROJETO ASPECTUAL				INFORMAÇÕES ADICIONAIS
			ADIÇÃO COMPORTAMENTAL			ADIC. ESTR.	
			COMPORTAMENTO TRANSVERSAL	REGRA COMPOSIÇÃO	PONTO DE JUNÇÃO		
UC#04 - Realizar Transferência	ServletConfirm DadosTransf	-	verificarLimite (valorTransac:double) :boolean	wrap	FachadaBanco. verificarDadosTransf (idContaOr: string; idContaDest:string; valor: double)	-	Se limiteOk então execute(pontoJunção) Senão exibe(msgErro)
UC#05 - Pagar boleto bancário	ServletConfirm DadosPgto	-	verificarLimite (valorTransac:double) :boolean	wrap	FachadaBanco. verificarDadosPgto (idConta: string; codBarras: string; data: Date:valor: double)	-	Se limiteOk então execute(ponto junção) Senão exibe(msgErro)

VISÃO DOS DIAGRAMAS DE SEQUÊNCIA COM A PRESENÇA DE OBJETOS DE PROJETO ASPECTUAIS

Com as tabelas de composição nível 3 preenchidas, podemos gerar uma visão dos diagramas de sequência dos casos de uso nos quais os objetos de projeto aspectuais que afetam a interação estão presentes.

A Figura 5.21 exibe a visão do diagrama de sequência do caso de uso *Visualizar saldo de conta* com a presença dos objetos de análise aspectuais que afetam a interação. Esse diagrama foi gerado tomando como base o diagrama de sequência da Figura 5.17 e usando as informações obtidas nas tabelas de composição nível 3 das

classes de projeto aspectuais que devem ser inseridas na interação do caso de uso *Visualizar saldo de conta* (Tabela 5.37, Tabela 5.38 e Tabela 5.40).

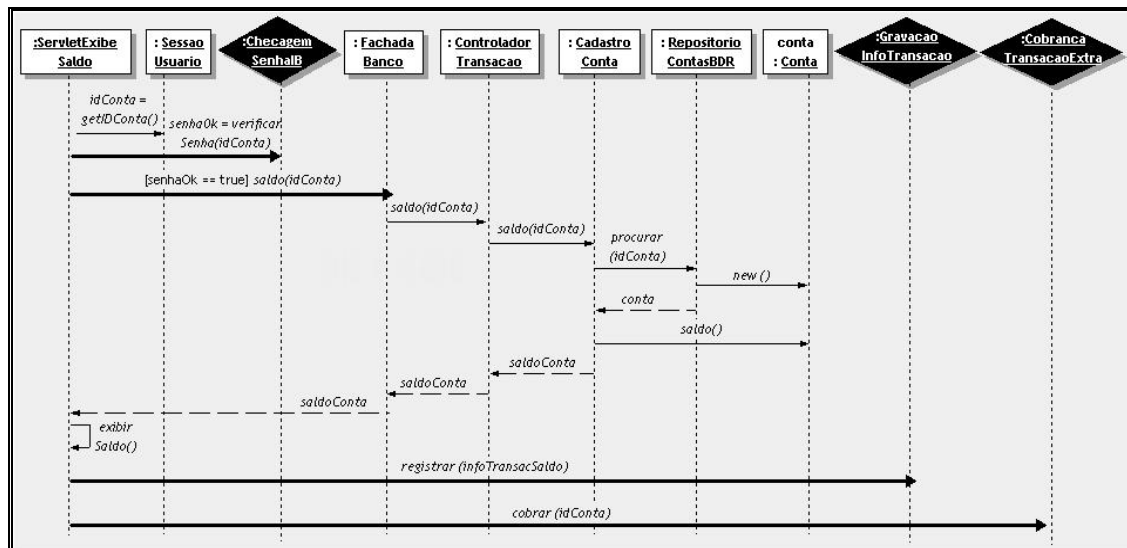


Figura 5.21- Visão do diagrama de sequência do caso de uso *Visualizar saldo de conta* COM a presença dos objeto de projeto aspectuais que afetam a interação

5.4.2 Projetar classes

Neste ponto do processo, analisamos cada classe de projeto aspectual individualmente, a fim de definir sua estrutura (atributos, operações), seus relacionamentos e como suas operações serão implementadas.

Nesta seção, apresentamos como projetamos a classe de projeto aspectual *CobrancaCPMF*. Primeiramente analisamos como suas operações seriam realizadas. Em seguida, definimos a estrutura dessa classe e seus relacionamentos. A principal entrada para realizar o projeto dessa classe de projeto aspectual foi sua tabela de composição nível 3 (Tabela 5.36).

Conforme apresentado na Figura 5.22, para realizar a operação *saldoDisponívelCPMF*, seria necessário sempre fazer uma consulta ao cadastro que manipula com o CPMF acumulado das contas para saber qual o valor do CPMF acumulado. Entretanto, essa solução não é eficiente caso seja preciso consultar várias vezes o valor do CPMF acumulado.

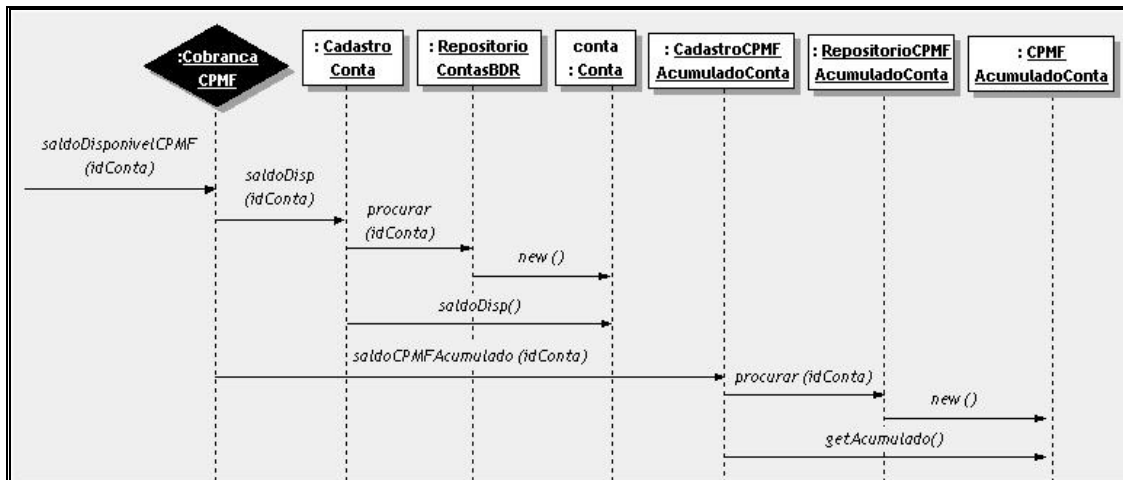


Figura 5.22 - Diagrama de sequência para realização da operação *saldoDisponivelCPMF* da classe de projeto aspectual *CobrancaCPMF*

Por isso, decidimos utilizar uma outra solução: fazer com que esse valor seja atributo da classe *Conta*, sendo carregado e atualizado, respectivamente, na criação e atualização de objetos dessa classe. Dessa maneira, teria que ser feita uma adição estrutural na classe *Conta*, para que ela possua o atributo *valorAcumuladoCPMF*. Além disso, serão necessários os seguintes métodos de acesso a esse atributo: *getAcumuladoCPMF()* e *setAcumuladoCPMF()*.

Conforme o passo 2 do roteiro descrito na Figura 4.17, cada uma dessas adições estruturais será representada numa linha da tabela de composição nível 3 e o conteúdo da coluna classe de projeto afetada será a classe *Conta*. Para fazer com que esse atributo seja manipulado na classe *Conta*, serão necessárias ainda mudanças comportamentais no *RepositorioContasBDR* para que, ao executar os métodos “consultar”, “inserir”, “excluir” e “atualizar”, também sejam feitas as operações correspondentes na classe *RepositorioCPMFAcumuladoContaBDR*. Todas essas informações foram colocadas na tabela de composição nível 3 dessa classe, conforme ilustra a Tabela 5.43.

Tabela 5.43 – Tabela de composição nível 3 para classe de projeto aspectual CobrançaCPMF (incluindo adição estrutural)

CLASSE DE PROJETO ASPECTUAL: CobrançaCPMF							
CASO DE USO AFETADO	CLASSE PROJETO AFETADA	COND.	ADIÇÕES PROVIDAS PELA CLASSE DE PROJETO ASPECTUAL				INFORMAÇÕES ADICIONAIS
			ADIÇÃO COMPORTAMENTAL			ADIÇÃO ESTRUTURAL	
			COMPORTAMENTO TRANSVERSAL	REGRA COMPOSIÇÃO	PONTO DE JUNÇÃO		
UC#04 - Realizar transferência	ControladorTransacao	O dia da transação corresponder ao dia do débito periódico do CPMF	saldoDisponivelCPMF (conta: Conta) : double	override	Conta. saldoDisponivel()	-	saldoDisponivelCPMF = Conta.saldoDisponivel() – Conta.getAcumuladoCPMF()
UC#04 - Realizar transferência	ControladorTransacao	-	acumularCPMF (conta: Conta; valorTransac: double)	overlap.after	Conta. debitar(valor: double)	-	Conta.adicionarCPMF (valorTransac: double)
UC#05 - Pagar boleto bancário	ControladorTransacao	O dia da transação corresponder ao dia do débito periódico do CPMF	saldoDisponivelCPMF (conta: Conta) : double	override	Conta. saldoDisponivel()	-	saldoDisponivelCPMF = Conta.saldoDisponivel() – Conta.getAcumuladoCPMF()
UC#05 - Pagar boleto bancário	ControladorTransacao	-	acumularCPMF (conta: Conta; valorTransac: double)	overlap.after	Conta.debitar (valor: double)	-	Conta.adicionarCPMF (valorTransac: double)
	Conta					valorAcumuladoCPMF: double	-
	Conta					getAcumuladoCPMF(): double	-
	Conta					setAcumuladoCPMF (valor:double)	-
	Conta					adicionarCPMF (valorTransac; double)	CPMFacumulado = CPMFacumulado + (valorTransac*CPMF)
	RepositorioContasBDR	-	atualizarValorCPMF	overlap.after	RepositorioContasBDR.atualizar(...)	-	Adições comportamentais em consequência das adições estruturais realizadas
	RepositorioContasBDR	-	inserirValorCPMF	overlap.after	RepositorioContasBDR.Inserir(...)	-	
	RepositorioContasBDR	-	excluirValorCPMF	overlap.after	RepositorioContasBDR.excluir(...)	-	
	RepositorioContasBDR	-	carregarValorCPMF	overlap.after	RepositorioContasBDR.consultar(...)	-	

A partir da Tabela 5.43, temos informações suficientes para identificar os atributos e operações da classe de projeto aspectual *CobrancaCPMF*.

A partir da coluna “comportamento transversal”, identificamos as operações que deverão ser invocadas pelas classes afetadas. Na definição da estrutura da classe de projeto aspectual, essas operações são antecedidas do estereótipo <<weave>>. A partir da coluna “Adição estrutural” identificamos atributos e operações necessários para realização do comportamento do aspecto que deverão ser inseridos em outras classes. Esses atributos e operações são antecedidos do estereótipo <<introduction>> na definição da estrutura da classe. A Figura 5.23 ilustra a estrutura da classe de projeto aspectual *CobrancaCPMF*.

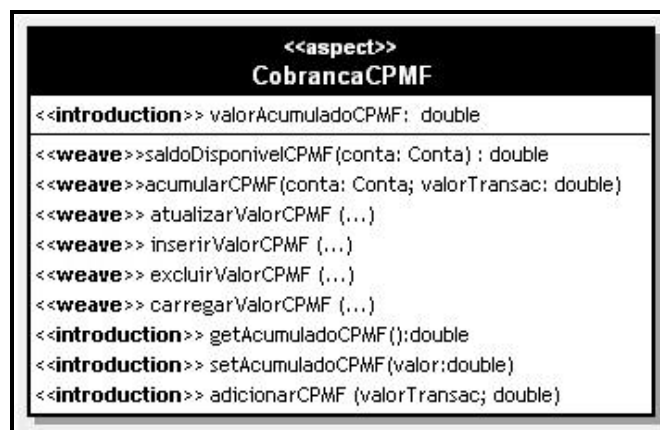


Figura 5.23 – Estrutura da classe de projeto aspectual *CobrancaCPMF*

O relacionamento entre a classe de projeto aspectual *CobrancaCPMF* e as classes afetadas por ela (identificadas a partir da coluna “classe de projeto afetada” da Tabela 5.43) é ilustrado na Figura 5.24.

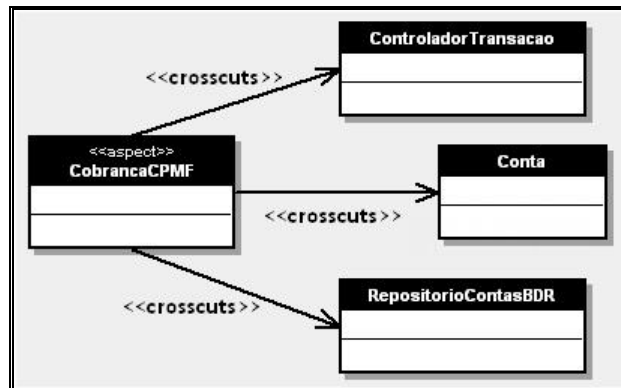


Figura 5.24 – Diagrama de classe enfatizando relacionamento da classe de projeto aspectual *CobrancaCPMF* e as classes afetadas por ela

A estrutura e o relacionamento das demais classes de projeto aspectuais (*CobrancaTransacaoExtra*, *GravacaoInfoTransacao*, *Bonificação*, *ChecagemSenhaIB*, *ChecagemDadosPessoais* e *VerificacaoLimite*) foi realizada de maneira similar.

5.5 Considerações finais

Embora estudos de caso não possam alcançar o rigor científico de experimentos formais, eles podem fornecer informações suficientes para ajudar a julgar se uma determinada abordagem irá prover benefícios durante um projeto de desenvolvimento de software [KITCHENHAM et al., 1994].

Ao longo desse capítulo mostramos como nossa abordagem poderia ser utilizada para separar preocupações transversais que devem ser manipuladas no contexto de um sistema de *Internet Banking*.

Utilizando as práticas adotadas pelo Processo Unificado e pelos métodos orientados a objetos tradicionais, preocupações como cobrança de CPMF, checagem de senha, cobrança de tarifa por transação extra e verificação de limite de transação ficariam misturadas nos cenários dos casos de uso onde elas deveriam ser aplicadas e ainda nos respectivos diagramas de seqüência e colaboração desses casos de uso. Esse tipo de prática traz algumas dificuldades. Primeiro, complexidade extra seria inserida nessas especificações, prejudicando o entendimento do real objetivo desses artefatos. Segundo, como as especificações ficariam misturadas com preocupações adicionais que não tem relação com o objetivo essencial do caso de uso, o reuso integral desses

artefatos em sistemas de outros bancos que não adotem alguma dessas preocupações seria dificultado. Outra decorrência negativa é a sobrecarga para o engenheiro, visto que ele deveria lembrar de fazer referência a essas preocupações em todos os casos de uso e ainda nos respectivos diagramas de seqüência e colaboração onde essas preocupações devem ser consideradas.

Do contrário, conforme exibimos neste capítulo, usando nossa abordagem as preocupações transversais mencionadas acima não precisaram ficar espalhadas e misturadas em vários artefatos, elas foram representadas em unidades à parte e sua composição nos casos de uso e diagramas de seqüência e colaboração foram modeladas em tabelas de composição.

Comparando os artefatos gerados ao longo do processo utilizando nossa abordagem com aqueles que seriam gerados caso estivéssemos utilizando a abordagem tradicional, pudemos perceber que os artefatos que obtivemos possuem melhor potencial de compreensibilidade, manutenibilidade e reusabilidade.

Apesar de estudos de caso serem mais difíceis de generalizar do que experimentos [KITCHENHAM et al., 1994], os resultados que obtivemos nos artefatos do sistema de *Internet Banking* nos motivam a acreditar que benefícios semelhantes podem ser obtidos em outros sistemas. Entretanto, para confirmar isso, precisamos realizar mais estudos de casos.

6

CONCLUSÃO

*“Ando devagar porque já tive pressa. E levo esse sorriso porque já chorei demais.
Hoje me sinto mais forte, mais feliz quem sabe. Só levo a certeza de que muito
pouco eu sei. Eu nada sei... Todo mundo ama um dia, todo mundo chora.
Um dia a gente chega, no outro vai embora. Cada um de nós compõe a sua
história, e cada ser em si, carrega o dom de ser capaz, e ser feliz”.*

Almir Sater

Este capítulo apresenta de maneira sintética os resultados da pesquisa. Primeiramente, discutimos o estágio atual da orientação a aspectos. Em seguida, salientamos as contribuições da nossa abordagem e a comparamos com trabalhos relacionados. Depois, sugerimos alguns tópicos para trabalhos futuros, baseando-nos nas limitações da abordagem. Por fim, avaliamos se os objetivos estabelecidos foram alcançados e apresentamos nossas considerações finais.

6.1 Discussão

A separação de preocupações é um princípio natural que o ser humano aplica ao resolver problemas. No desenvolvimento de software não é diferente: os paradigmas de desenvolvimento fornecem mecanismos e práticas para decompor um problema em unidades menores, possibilitando manipular com uma preocupação do sistema de cada vez.

A orientação a aspectos é uma nova abordagem para modularizar sistemas complexos¹⁴. Como uma extensão dos paradigmas atuais, o paradigma orientado a aspectos tem o objetivo de capturar e modularizar preocupações que normalmente ficam espalhadas e misturadas em várias unidades de software, denominadas de preocupações transversais ou aspectos. Dessa maneira, esse novo paradigma pretende melhorar as potencialidades de reusabilidade, manutenibilidade e compreensibilidade dos artefatos, qualidades que ficavam comprometidas devido ao espalhamento e entrelaçamento de preocupações transversais nas especificações.

O paradigma orientado a aspectos ainda está em processo de amadurecimento e muito ainda precisa ser feito para que esse seus fundamentos, práticas e mecanismos se estabilizem e esse paradigma venha a ser largamente adotado. Atualmente, ele tem tido foco, principalmente, na pesquisa acadêmica e os avanços mais significativos nessa área concentraram-se em linguagens orientadas a aspectos. Com a crescente aceitação dessas linguagens, surgiu o interesse da comunidade em aplicar a orientação a aspectos em vários estágios do ciclo de vida do software. No entanto, faltam métodos, processos e técnicas que propiciem a sua utilização nas atividades do ciclo de desenvolvimento. E, somente com a utilização dos fundamentos desse paradigma em grandes projetos e com a realização de experimentos que avaliem seu impacto no processo de desenvolvimento é que será possível saber se esse paradigma perdurará ou não.

¹⁴ Software complexos são aqueles constituídos de um grande número de partes que interagem de uma maneira não-trivial, onde o todo não se limita à soma das partes [SIMON, 1981].

6.2 Contribuições

Em seu artigo seminal sobre a orientação a aspectos, KICZALES et al. (1997) apresenta vários pontos em que a pesquisa nessa área poderia avançar, entre eles estão: (i) como antecipar a modelagem de aspectos para as atividades de análise e projeto; e (ii) como integrar a orientação a aspectos com métodos, ferramentas e processos de desenvolvimento existentes. Este trabalho oferece contribuições nessas duas direções. Primeiro, porque nossa abordagem permite antecipar a modelagem de aspectos não apenas nas atividades de análise e projeto, mas também oferece suporte para que desde o fluxo de requisitos, seja possível analisar as partes do sistema que serão afetadas por uma preocupação transversal. Segundo, porque neste trabalho conseguimos realizar isso através da adaptação do Processo de Desenvolvimento Unificado [JACOBSON et al., 1999].

Este trabalho também avança em questões atuais de interesse da área de Engenharia de Requisitos Orientada a Aspectos [WORKSHOP ON EARLY ASPECTS, 2003]: a identificação de aspectos no nível de requisitos e como rastreá-los nas atividades seguintes de desenvolvimento. Nesta proposta, apresentamos algumas heurísticas para identificar aspectos que devem ser aplicados em casos de uso, permitindo a identificação tanto de aspectos derivados de preocupações não-funcionais como daqueles oriundos de preocupações funcionais e regras de negócio. Desde que são identificados no fluxo de requisitos e mesmo nos fluxos de análise e projeto, esses aspectos são vinculados aos casos de uso nos quais eles devem ser aplicados, facilitando o rastreamento de aspectos ao longo do processo.

No entanto, a principal contribuição deste trabalho é possibilitar a separação de preocupações transversais (ou aspectos) desde atividades iniciais do ciclo de desenvolvimento e, conseqüentemente, melhorar o potencial de reuso, manutenção e compreensão nos artefatos gerados no processo de desenvolvimento.

O reuso é facilitado porque, separando as preocupações transversais dos artefatos de software, desenvolvedores podem reusar exatamente o que eles precisam, não precisando retirar desses artefatos partes que são incompatíveis com o contexto de reuso, tais como regras de negócio e preocupações não-funcionais.

A manutenção dos artefatos é simplificada visto que preocupações transversais não ficam mais espalhadas em vários artefatos e, por isso, o número de artefatos que precisa ser alterado para adicionar, remover ou modificar uma preocupação transversal é reduzido.

A compreensão é melhorada em dois sentidos: (i) a especificação de artefatos como cenários de casos de uso e diagramas de sequência e colaboração foca-se apenas no objetivo do caso de uso que eles estão modelando; e (ii) não é necessário analisar todos os artefatos para entender o impacto de uma preocupação transversal no sistema, basta examinar a tabela de composição do aspecto.

6.3 Comparação com trabalhos relacionados

Comparando a parte de requisitos da nossa abordagem com os trabalhos apresentados na Seção 3.4.1.1, identificamos um ponto comum com o modelo de processo AORE [RASHID et. al, 2002 e RASHID et. al, 2003]: a identificação e análise de preocupações funcionais e não-funcionais em atividades separadas. Por outro lado, existem três diferenças significativas entre nossa proposta e os trabalhos pesquisados na área de engenharia de requisitos orientada a aspectos: (i) realizamos a composição de preocupações não-funcionais em termos de operacionalizações dessas preocupações, ao contrário dos trabalhos pesquisados que, utilizam declarações abstratas, vagas e subjetivas (ex: segurança, performance); (ii) oferecemos heurísticas para identificar não apenas aspectos oriundos de preocupações não-funcionais, mas também aqueles oriundos de preocupações funcionais e regras de negócio; (iii) realizamos a composição de maneira não-invasiva em tabelas de composição.

Por sua vez, em comparação com os trabalhos apresentados na Seção 3.4.2, a parte de projeto da nossa abordagem assemelha-se com as proposta de ZAKARIA et al. (2002) e PAWLAK et al. (2002) no que se refere ao uso do estereótipo <<aspect>> do elemento *class* para representar um aspecto; e é semelhante a proposta de SUZUKI & YAMAMOTO (1999) no que se refere ao uso de estereótipos antes de membros do aspecto que deverão ser adicionados em outras classes. No entanto, divergimos de todas as propostas analisadas em relação ao modo como a composição entre aspectos e classes é especificada: para isso utilizamos tabelas de composição. Ademais, as propostas de

projeto analisadas focam apenas em notações para representar aspectos e sua composição, enquanto a nossa oferece ainda diretrizes para identificar pontos de junção e classes afetadas.

Conforme vimos na Seção 3.4.1.3 e na Seção 3.4.2.1, a maioria das propostas atuais em orientação a aspectos sugere que se coloque na especificação do aspecto quais unidades do sistema ele deve afetar e como deve afetar cada uma delas, o que caracteriza uma composição dirigida ao componente (Seção 3.3.1). Isso promove acoplamento entre o aspecto e as unidades base, comprometendo o reuso do aspecto em outros sistemas. Por isso, acreditamos que a utilização de tabelas de composição, tanto no fluxo de requisitos como nos de análise e projeto, avança em relação às propostas atuais já que permite modelar a composição de aspectos de maneira fechada (Seção 3.3.1), ou seja, sem que existam referências na(s) unidade(s) base ou no aspecto sobre a relação entre eles.

Por fim, um trabalho que não tem relação direta com a área de orientação a aspectos, mas que possui algumas semelhanças com a nossa abordagem no nível de requisitos é o trabalho de CYSNEIROS et al. (2001). Esse trabalho propõe uma estratégia para integrar requisitos não-funcionais em modelos conceituais. Nessa estratégia, o fluxo de requisitos é realizado através de dois ciclos independentes, um considerando os requisitos funcionais do sistema e o outro os requisitos não-funcionais. A integração dessas visões é realizada através da inserção das operacionalizações obtidas com o *framework* NFR [MYLOPOULOS et al., 1992; CHUNG et al., 2000; MYLOPOULOS et al., 2001] nos diagramas de entidade-relacionamento, classe, seqüência e colaboração. Uma semelhança dessa abordagem com a nossa é que ambas lidam com requisitos funcionais e não-funcionais separadamente para depois integrá-los. No entanto, a abordagem de CYSNEIROS et al. (2001) não tem a intenção de separar preocupações e, por isso, operacionalizações de requisitos não-funcionais ficam misturadas e espalhadas nos modelos onde elas devem ser consideradas.

6.4 Limitações e Trabalhos futuros

Este trabalho trata do desenvolvimento de software orientado a aspectos nas atividades de requisitos, análise e projeto, mas não chega a analisar como seria a

transição para a implementação do sistema. Uma perspectiva de trabalho futuro seria, portanto, implementar um sistema com diferentes linguagens orientadas a aspectos (ex: AspectJ [KICZALES et al., 2001] e Hyper/J [OSSHER & TARR, 2000]) e também da maneira tradicional com uma linguagem orientada a objetos, partindo dos artefatos gerados com nossa abordagem. A partir das dificuldades encontradas nessa transição seria possível propor melhorias na abordagem.

A abordagem de padrões de composição [CLARKE & WALKER, 2001] (Seção 3.4.2)) possibilita que o modelo de projeto do aspecto seja extensível e reusável pois modela um aspecto de forma parametrizável, não sendo feitas referências às classes que ele afeta e sim utilizando parâmetros para representá-las. Essa é uma abordagem promissora e poderia ter sido aplicada nos fluxos de análise e projeto da nossa proposta. Neste trabalho, porém, preferimos manter a uniformidade da abordagem ao longo dos fluxos de requisitos, análise e projeto: primeiramente separando as preocupações em entidades à parte e depois especificando o modo como essas preocupações devem ser inseridas nas unidades que elas afetam através de tabelas de composição. Sendo assim, uma outra possibilidade de trabalho futuro seria a integração da abordagem de CLARKE & WALKER, 2001 com o nosso trabalho.

Melhorias também poderiam ser propostas a partir da realização de mais estudos de caso. Com a experiência adquirida, também seria possível fornecer mais heurísticas e diretrizes para os desenvolvedores.

Neste trabalho não refletimos sobre o impacto das adaptações realizadas em relação a questões arquiteturais e em relação à iteratividade e incrementalidade do Processo Unificado. Essa poderia ser outra área de pesquisa futura.

Também seria interessante ter ferramentas que gerassem automaticamente, a partir das tabelas de composição, a perspectiva de especificações de casos de uso e diagramas de interação com a presença de aspectos que afetam esses artefatos; bem como ferramentas para gerenciar as dependências das informações nas tabelas de composição.

Pelo fato de oferecer suporte à separação de preocupações transversais, entre elas requisitos não-funcionais e regras de negócio, tipicamente requisitos que variam com mais frequência dependendo do contexto onde o sistema vai ser inserido, um

possível trabalho futuro seria investigar o uso da nossa abordagem na área de linhas de produto [CLEMENTS & NORTHROP, 2001].

6.5 Considerações finais

Conforme apresentamos no Capítulo 4, conseguimos realizar o objetivo inicialmente proposto de estabelecer uma abordagem simples para possibilitar a separação de preocupações transversais desde atividades iniciais do ciclo de desenvolvimento. Alcançamos este objetivo inserindo os fundamentos do paradigma orientado a aspectos nos fluxos de requisitos, análise e projeto do Processo Unificado, mas sem ocasionar tanto impacto em como o processo é conduzido, já que as alterações concentram-se mais na maneira como as unidades serão estruturadas.

Além disso, pelo fato das adaptações não se prenderem aos mecanismos de uma abordagem ou de uma linguagem orientada a aspectos específica e sim aos conceitos fundamentais do paradigma orientado a aspectos, a nossa abordagem dá suporte tanto a uma implementação orientada a aspectos como a uma implementação tradicional. Vale a pena frisar, contudo, que o paradigma orientado a aspectos está em processo de evolução e amadurecimento e os fundamentos da orientação a aspectos utilizados em nossa abordagem refletem o estágio atual dessa área.

Outra característica positiva da nossa abordagem é que as adaptações seguem a mesma linha de raciocínio ao longo dos fluxos de requisitos, análise e projeto: primeiramente separando as preocupações em entidades à parte e depois especificando o modo como essas preocupações devem ser inseridas nas unidades que elas afetam através de tabelas de composição.

Pelo fato de possibilitar a separar preocupações transversais, os artefatos que obtivemos a partir do estudo de caso realizado possuem melhor potencial de reuso, manutenção e compreensão, quando comparados com aqueles que seriam obtidos seguindo uma abordagem tradicional, na qual as preocupações transversais, normalmente, ficariam espalhadas e misturadas nos artefatos.

Apesar de mais estudos de casos e experimentos serem necessários para avaliar e aprimorar nossa abordagem, este trabalho representa o primeiro passo para que o

Processo Unificado possa ser utilizado no desenvolvimento de software orientado a aspectos.

REFERÊNCIAS BIBLIOGRÁFICAS

- [AKSIT et al., 2001] AKSIT, M.; TEKINERDOGAN, B. and BERGMANS, L. "The Six Concerns for Separation of Concerns", in Proceedings of ECOOP 2001 Workshop on Advanced Separation of Concerns, Budapest, Hungary, June 18-22, 2001.
- [ALDAWUD et al., 2001] ALDAWUD, O.; ELRAD, T. and BADER, A. "A UML Profile for Aspect Oriented Modeling". OOPSLA 2001 Workshop on Aspect Oriented Programming, 2001.
- [AMBLER, 2001] AMBLER, S. "The Object Primer: the Application Developer's Guide to Object Orientation and the UML", Cambridge University Press, 2001.
- [AOSD, 2003] AOSD Mailing Discussion List. Disponível em: <http://aosd.net>. Acesso em novembro de 2003.
- [ARAUJO et al., 2002] ARAÚJO, J.; MOREIRA, A.; BRITO, I. and RASHID, A. "Aspect-Oriented Requirements with UML", Workshop: Aspect-oriented Modeling with UML, UML 2002, Dresden, Germany, 2002.
- [BANIASSAD et al., 2002] BANIASSAD, E.; MURPHY, G.; SCHWANNINGER, C. and KIRCHER, M. "Managing Crosscutting Concerns During Software Evolution Tasks: an Inquisitive Study", Proceedings of the 1st International Conference on Aspect-Oriented Software Development, Enschede, The Netherlands, April 22-26, 2002.
- [BERGMANS & AKSIT, 2000] BERGMANS, L. and AKSIT, M. "Composing Software from Multiple Concerns: A Model and Composition Anomalies", Multi Dimensional Separation of Concerns in Software Engineering Workshop, ICSE 2000, Limerick, Ireland, 2000.
- [BERGMANS & AKSIT, 2001] BERGMANS, L. and AKSIT, M. "Composing Crosscutting Concerns Using Composition Filters", Communications of the ACM, Vol. 44, No. 10, pp. 51-57, October 2001.
- [BERGMANS et al., 2001] BERGMANS, L.; AKSIT, M. and TEKINERDOGAN, B. "Aspect Composition Using Composition Filters", In Software Architectures and Component Technology: The State of the Art in Research and Practice, M. Aksit (Ed.), Kluwer Academic Publishers, pp. 357 - 382, October 2001. (ISBN 0-7923-7576-9)
- [BOOCH et al., 1999] BOOCH, G.; RUMBAUGH, J. and JACOBSON, I. "The Unified Modeling Language User Guide". Addison-Wesley, 1999.

- [BORGIDA, 1985] BORGIDA, A. "Features of Languages for the Development of Information Systems at the Conceptual Level", IEEE Software, Jan 1985.
- [BRITO & MOREIRA, 2003] BRITO, I. and MOREIRA, A. "Towards a Composition Process for Aspect-Oriented Requirements". Workshop on Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design, March 17 - Boston, USA, 2003.
- [BROOKS, 1987] BROOKS, F., "No Silver Bullet: Essence and Accidents of Software Engineering". Computer, Vol. 20, No. 4, pp. 10-19, April 1987.
- [CHUNG et al., 2000] CHUNG, L; NIXON, B.; YU, E. and MYLOPOULOS, J. "Non-Functional Requirements in Software Engineering". Boston Kluwer Academic Publishers, 2000.
- [CLARKE & WALKER, 2001] CLARKE, S. and WALKER, J. "Composition patterns: An Approach to Designing Reusable Aspects". In Proceedings of the 23rd International Conference on Software Engineering, Toronto, Canada, 12-19 May 2001.
- [CLARKE & WALKER, 2002] CLARKE, S. and WALKER, J. "Towards a Standard Design Language for AOSD". In proceedings of the 1st International Conference on Aspect-Oriented Software Development, Enschede, The Netherlands, April 2002
- [CLARKE et al., 1999] CLARKE, S.; HARRISON, W.; OSSHER, H. and TARR, P. "Subject-Oriented Design: Towards Improved Alignment of Requirements, Design and Code". In proceedings of Object-Oriented Programming, Systems, Languages and Applications (OOPSLA) Denver, Colorado U.S., November 1999.
- [CLARKE, 2002] CLARKE, S. "Extending standard UML with model composition semantics" in Science of Computer Programming, Volume 44, Issue 1, pp. 71-100. Elsevier Science, July 2002
- [CLEMENTS & NORTHROP, 2001] CLEMENTS, P. and NORTHROP, L. Software Product Lines: Practices and Patterns. Addison-Wesley, 2001.
- [COADY et al., 2001] COADY, Y.; KICZALES, G.; FEELEY, M. and SMOLYN, G. "Using AspectC to Improve the Modularity of Path-Specific Customization in Operating System Code", FSE 2001.
- [COCKBURN, 2000] COCKBURN, A. "Writing Effective Use Cases". Addison-Wesley, 2000.
- [CONSTANTINIDES & HASSOUN, 2002] CONSTANTINIDES, C. and HASSOUN, Y. "Beyond objects: Improving the modularity of complex software". Workshop on Grand Challenges for Computing Research. Edinburgh, Scotland, November 24-26, 2002.

- [CYSNEIROS & LEITE, 1999] CYSNEIROS, L. e LEITE, J. "Integrating Non-Functional Requirements into Data Modeling". 4th International Symposium on Requirements Engineering. Limerick, Ireland. Jun. 1999.
- [CYSNEIROS et al., 2001] CYSNEIROS, L.; LEITE, J. and NETO, J. "A Framework for Integrating Non-Functional Requirements into Conceptual Models". Requirements Engineering Journal – Vol. 6, Issue 2, pp: 97-115, April 2001.
- [DIJKSTRA, 1976] DIJKSTRA, E. "A Discipline of Programming". Prentice-Hall, 1976.
- [ELRAD et al., 2001a] ELRAD, T., FILMAN, R. and BADER, A. "Aspect-Oriented Programming: Introduction", Communications of the ACM, v.44 n.10, p.29-32, Oct. 2001
- [ELRAD et al., 2001b] ELRAD, T.; AKSIT, M.; KICZALES, G.; LIEBERHERR, K. and OSSHER, H. "Discussing Aspects of AOP". Communications of the ACM, 44(10):33–38, October 2001.
- [FENG et al., 2001] FENG, L.; MARCUS, A. and SCHAFFER, K. "An Overview of Aspect Oriented Programming", Term Report, Kent State University, Department of Computer Science, 2001.
- [FILMAN, 2001] FILMAN, R. "What is Aspect-Oriented Programming, Revisited" Workshop on Advanced Separation of Concerns, 15th European Conference on Object-Oriented Programming, Budapest, June 2001.
- [FINKELSTEIN & SOMMERVILLE, 1996] FINKELSTEIN, A. and SOMMERVILLE, I. "The Viewpoints FAQ". BCS/IEEE Software Engineering Journal, Vol. 11, No. 1, 1996.
- [GHEZZI et al., 1991] GHEZZI, C.; JAZAYERI, M. and MANDRIOLI, D. "Fundamentals of Software Engineering". Prentice Hall, 1991.
- [GOSLING et al., 2000] GOSLING, J.; JOY, B.; STEELE, G. and BRACHA, G. "The Java Language Specification". Addison-Wesley, 2nd Edition, 2000.
- [HARRISON & OSSHER, 1993] HARRISON, W. and OSSHER, H. "Subject-Oriented Programming (a Critique of Pure Objects). In Proceedings of Conference on Object Oriented Programming: Systems, Languages, and Applications, pages 411-428, 1993.
- [HURSCH & LOPES, 1995] HURSCH, W. and LOPES, C. "Separation of Concerns". Technical Report NU CCS-95-03, Northeastern University, 1995.

- [IEEE-STD. '610', 1990] IEEE-STD. '610' "IEEE Standard Glossary of Software Engineering Terminology". Institute of Electrical and Electronics Engineers, New York, 1990.
- [JACOBSON & CHRISTERSON, 1995] JACOBSON, I., and CHRISTERSON, M., "A Growing Consensus on Use Cases", Journal of Object-Oriented Programming, vol. 8, pp. 15-19, March- April, 1995.
- [JACOBSON et al., 1992] JACOBSON, I; CHRISTERSON, M; JONSSON, P. and OVERGAARD, G. "Object-Oriented Software Engineering: A Use Case Driven Approach". Addison Wesley, 1992.
- [JACOBSON et al., 1999] JACOBSON, I.; BOOCH, G.; and RUMBAUGH, J. "The Unified Software Development Process", Addison-Wesley, 1999.
- [JACOBSON, 1986] JACOBSON, I. "FDL: A Language for Designing Large Real Time Systems", Proceedings of IFIP 86, Sept. 1986.
- [JACOBSON, 1987] JACOBSON, I. "Object-oriented Development in an Industrial Environment". Proceedings of OOPSLA'87. SIGPLAN Notices, 22(12), pp. 183-191, 1987.
- [JACOBSON, 2003a] JACOBSON, I. "Use Cases - Yesterday, Today, and Tomorrow". The Rational Edge, March 2003.
- [JACOBSON, 2003b] JACOBSON, I. "Use Cases and Aspects - Working Seamlessly Together", In Journal of Object Technology, Vol. 2, No. 4, July-August 2003, pp. 7- 28.
- [KERSTEN & MURPHY, 1999] KERSTEN, M. and MURPHY, G. "Atlas: A Case Study in Building a Web-Based Learning Environment using Aspect-oriented Programming". In OOPSLA'1999. 1999.
- [KICZALES et al., 1997] KICZALES, G.; LAMPING, J.; MENDHEKAR, A.; MAEDA, C.; LOPES, C.; LOINGTIER, J. and IRWIN, J. "Aspect-Oriented Programming". In M. Aksit and S. Mat-suoka, editors, European Conference on Object-Oriented Programming, volume 1241 of LNCS, pages 220-242. Springer Verlag, 1997.
- [KICZALES et al., 2001] KICZALES, G.; HILSDALE, E.; HUGUNIN, J.; KERSTEN, M.; PALM, J. and GRISWOLD, W. "An Overview of AspectJ". In J. L. Knudsen, editor, 15th European Conference on Object-Oriented Programming, volume 2072 of LNCS, pages 327–353, Berlin, Heidelberg, and New York, Springer Verlag, 2001.
- [KIM, 2002] KIM, H. "AspectC#: An AOSD implementation for C#". M.Sc. Thesis, November 2002, Trinity College, Dublin.
- [KITCHENHAM et al., 1994] KITCHENHAM, B.; PICKARD, L. and PFLEEGER, S. "Case Studies for Method and Tool Evaluation" . IEEE, 1994.

- [KOTONYA & SOMMERVILLE, 1998] KOTONYA, G., SOMMERVILLE, I. "Requirements Engineering: Processes and Techniques". Wiley, John & Sons Inc, 1998.
- [LIEBERHERR & HOLLAND, 1989] LIEBERHERR, K. and HOLLAND, I. "Assuring Good Style for Object-Oriented Programs". IEEE Software, September 1989, pages 38-48.
- [LIEBERHERR, 1996] LIEBERHERR, K. "Adaptive Object-Oriented Software: The Demeter Method with Propagation Patterns". PWS Publishing Company, Boston, 1996.
- [LOUCOPOULOS & KARAKOSTAS, 1995] LOUCOPOULOS, P. and KARAKOSTAS, V. "System Requirements Engineering", McGraw Hill International Series in Software Engineering, 1995.
- [MASSONI et al., 2001] MASSONI, T.; ALVES, V.; SOARES, S. and BORBA, P. "PDC: Persistent Data Collections Pattern". In First Latin American Conference on Pattern Languages of Programming — SugarLoafPLoP, Rio de Janeiro, Brazil, October 2001.
- [MASUHARA & KICZALES, 2003] MASUHARA, H. and KICZALES, G. "Modeling Crosscutting in Aspect Oriented Mechanisms". In Proceedings of European Conference on Object-Oriented Programming, Darmstadt, Germany, July 2003, pp. 2–28.
- [MOREIRA et al., 2002] MOREIRA, A.; ARAÚJO, J. and BRITO, I. "Crosscutting Quality Attributes for Requirements Engineering", 14th International Conference on Software Engineering and Knowledge Engineering (SEKE 2002), ACM Press, Italy, July, 2002.
- [MURPHY et al., 2001] MURPHY, G; WALKER, R; BANIASSAD, E.; ROBILLARD, M.; LAI, A. and KERSTEN, M. "Does Aspect-Oriented Programming Work?". Communications of the ACM, 44(10):75–77, October 2001.
- [MYLOPOULOS et al., 1992] MYLOPOULOS, J.; CHUNG, L. and NIXON, B. "Representing and Using Non-Functional Requirements: A Process-Oriented Approach". IEEE Transactions on Software Engineering, Vol. 18, No. 6, pp. 483-497, June 1992.
- [MYLOPOULOS et al., 2001] MYLOPOULOS, J.; CHUNG, L.; LIAO, S.; WANG, H. and YU, E. "Exploring Alternatives during Requirements Analysis". IEEE Software, pp. 2-6, Jan/Feb 2001.
- [OMG, 2003] OMG - Object Management Group. "Unified Modeling Language Specification Version 1.5", Mar. 2003.

- [OSSHER & TARR, 2000] OSSHER, H. and TARR, P. "Multi- Dimensional Separation of Concerns and the Hyperspace Approach". In Proceedings of the Symposium on Software Architectures and Component Technology: The State of the Art in Software Development. Kluwer, 2000.
- [OSSHER & TARR, 2001] OSSHER, H. and TARR, P. "Using Multidimensional Separation of Concerns to (Re) Shape Evolving Software". Communications of the ACM, October 2001, pp.43-50.
- [PACE & CAMPO, 2001] PACE, J. and CAMPO, M. "Analyzing the Role of Aspects in Software Design". Communications of the ACM, 44(10):67–73, October 2001.
- [PARNAS, 1972] PARNAS, D. "On the criteria to be used in decomposing systems into modules". In Communications of the ACM, Vol. 15(12):1053-1058, 1972.
- [PAWLAK et al., 2002] PAWLAK, R.; DUCHIEN, L.; FLORIN, G.; LEGOND-AUBRY, F.; MARTELLI, L.; SEINTURIER, L. "A UML Notation for Aspect-Oriented Software Design". First AOSD Workshop on Aspect-Oriented Modeling with UML, Enschede, The Netherlands, April 2002.
- [PRESSMAN, 2001] PRESSMAN, R. Software Engineering: a Practitioner's Approach. 5th ed., McGraw-Hill, 2001.
- [RASHID et. al, 2002] RASHID, A.; SAWYER, P.; MOREIRA, A. and ARAÚJO, J. "Early Aspects: a Model for Aspect-Oriented Requirements Engineering". IEEE Joint Conference on Requirements Engineering, Essen, Germany, September, 2002.
- [RASHID et. al, 2003] RASHID, A. MOREIRA, A. and ARAUJO, J. "Modularisation and Composition of Aspectual Requirements". 2nd International Conference on Aspect-Oriented Software Development, ACM, pp. 11-20, 2003.
- [RATIONAL, 2002] RATIONAL Software Corporation. "The Rational Unified Process Documentation", Version 2002.05.01.305.000, 2002, disponível na intranet do Centro de Informática da UFPE. *Propriedade e direitos reservados da Rational Software Corporation.*
- [RATIONAL, 2004] RATIONAL Software Corporation Web Site. Disponível em: <http://www.rational.com>. Último acesso: fevereiro/2004.
- [SIMON, 1981] SIMON, H. "The Sciences of the Artificial"; 2nd Edition, MIT Press, Cambridge, Massachusetts, USA, 1981.
- [SOMMERVILLE, 1996] SOMMERVILLE, I. "Software Engineering". Addison-Wesley, 5th edition, 1996.

- [SOUSA & CASTRO, 2003] SOUSA, G. and CASTRO, J. "Towards a Goal-Oriented Requirements Methodology Based on the Separation of Concerns Principle". In: Proceedings of the 6th International Workshop On Requirements Engineering. p. 223-239, Piracicaba, Brasil, 2003.
- [SOUSA et al., 2003] SOUSA, G.; SILVA, I. and CASTRO, J. "Adapting the NFR Framework to Aspect-Oriented Requirements Engineering". In: Proceedings of the XVII Brazilian Symposium on Software Engineering, p.177 – 192, Manaus, Oct. 2003.
- [SOUSA et al., 2004] SOUSA, G.; SOARES, S.; BORBA, P.; CASTRO, J. "Separation of Crosscutting Concerns from Requirements to Design: Adapting an Use Case Driven Approach". Early Aspects 2004: Aspect-Oriented Requirements Engineering and Architecture Design. Workshop at International Conference on Aspect-Oriented Software Development, AOSD 2004. Lancaster, UK, 22 a 26 de Março de 2004.
- [SPINCZYK et al., 2002] SPINCZYK, O.; GAL, A. and SCHRÖDER-PREIKSCHAT, W. "AspectC++: An Aspect-Oriented Extension to C++", Proceedings of the 40th International Conference on Technology of Object-Oriented Languages and Systems (TOOLS Pacific 2002), Sydney, Australia, February 18-21, 2002
- [STEIN et al., 2002] STEIN, D.; HANENBERG, S. and UNLAND, R. "A UML-based Aspect-Oriented Design Notation For AspectJ ". 2002
- [SUZUKI & YAMAMOTO, 1999] SUZUKI, J. and YAMAMOTO, Y. "Extending UML with Aspects: Aspect Support in the Design Phase", In Proceedings of the 3rd Aspect-Oriented Programming (AOP) Workshop at ECOOP'99, Springer LNCS 1743, 1999.
- [TARR et al., 1999] TARR, P.; OSSHER, H.; HARRISON, W. and SUTTON Jr, S. "N Degrees of Separation: Multidimensional Separation of Concerns". In Proceedings of the 21st International Conference on Software Engineering (ICSE'99), 107–119, May 1999.
- [TRAVASSOS et al., 2002] TRAVASSOS, G.; GUROV, D. and AMARAL, E. "Introdução à Engenharia de Software Experimental". Relatório Técnico, RT-ES-590/02, Rio de Janeiro, 2002.
- [VIEGA & VUAS, 2000] VIEGA, J. and VUAS, J. Can Aspect-Oriented Programming Lead to More Reliable Software? IEEE Software, 17(6); p. 19-21 Nov.-Dec. 2000.
- [WAMPLER, 2003] WAMPLER, D. "Use Cases as Aspects – An Approach to Software Composition". Disponível em: <http://www.aspectprogramming.com/>. Último Acesso em: 19/11/2003

- [WOHLIN et al., 2000] WOHLIN, C.; RUNESON, P.; HOST, M.; OHLSSON, M.; REGNELL, B. and WESSLEN, A. "Experimentation in Software Engineering: an Introduction", Kluwer Academic Publishers, USA, 2000.
- [WORKSHOP ON EARLY ASPECTS, 2003] WORKSHOP ON EARLY ASPECTS: Aspect-Oriented Requirements Engineering and Architecture Design, held in conjunction with the 2nd International Conference on Aspect-Oriented Software Development, March 17-21, 2003, Boston, USA.
- [ZAKARIA et al., 2002] ZAKARIA, A.; HOSNY, H. and ZEID, A. "A UML Extension for Modeling Aspect-Oriented Systems". AOSD-UML 2002

GLOSSÁRIO

A pesquisa em Desenvolvimento de Software Orientado a Aspectos é recente e, atualmente, não existe nenhum consenso sobre conceitos comumente usados nessa área. Por essa razão, nesta seção determinamos a ontologia considerada neste trabalho. Além da semântica para termos próprios da orientação a aspectos, também definimos o significado considerado para outros termos a fim de deixar claro a perspectiva que assumimos nesse trabalho.

TERMO	SEMÂNTICA
<i>acoplamento</i>	O grau de dependência entre dois módulos [AMBLER, 2001]: quanto maior o número de conexões entre dois módulos, maior será o acoplamento entre deles [GHEZZI et al., 1991]. Um forte acoplamento entre unidades diminui o potencial de reuso, compreensão e manutenção dessas unidades.
<i>artefato</i>	Uma <u>unidade</u> concreta que serve para representar, descrever ou modelar uma preocupação. De maneira geral, é um documento, modelo, arquivo, diagrama, ou outro item que é produzido, modificado ou usado durante o desenvolvimento de um sistema [AMBLER, 2001].
<i>aspecto</i>	Representa uma <u>preocupação transversal</u> dentro do sistema que está sendo modelado. Esse termo também é usado para nomear a entidade que provê representação modular de uma preocupação transversal.
<i>coesão</i>	o grau de relação que os integrantes de uma <u>unidade</u> possuem um com o outro [AMBLER, 2001]: quanto maior for a similaridade entre as preocupações dos integrantes de uma unidade (ex: uma classe), maior será a coesão dessa unidade [GHEZZI et al., 1991]. Portanto, para uma unidade ser coesiva o seu conteúdo deve ser fortemente relacionado.
<i>operacionalizações</i>	dados, operações e restrições no sistema que contribuem para a satisfação de uma <u>preocupação</u> não-funcional [CHUNG et al., 2000].
<i>operador de composição</i>	veja <u>regra de composição</u> .
<i>ponto de junção</i>	indica em que ponto(s) na especificação da(s) <u>unidade(s)</u> base o comportamento do <u>aspecto</u> deve ser incluído [BRITO & MOREIRA, 2003];
<i>preocupação</i>	Qualquer questão de interesse do software que precisa ser pensada como uma entidade relativamente bem-definida. Geralmente corresponde a um objetivo de alto-nível do sistema que está sendo desenvolvido [KOTONYA & SOMMERVILLE, 1998]. Na comunidade de desenvolvimento orientado a aspectos, esse termo é conhecido como <i>concern</i> .
<i>preocupação transversal</i>	Um tipo de preocupação que precisa ser considerada em várias unidades do sistema e que, utilizando mecanismos da orientação a objetos não consegue ser separada dessas unidades, ficando espalhada por elas e misturada com as preocupações específicas dessas unidades. Na comunidade de desenvolvimento orientado a aspectos, esse termo é conhecido com <i>crosscutting concern</i> .

TERMO	SEMÂNTICA
<i>regra de composição</i>	expressa a ordem seqüencial na qual o aspecto deve ser combinado com a(s) unidade(s) que ele afeta [MOREIRA et al., 2002], ou seja, especifica como o comportamento do aspecto deve ser aplicado no(s) ponto(s) de junção. Neste trabalho definimos as regras de composição utilizando um os operadores propostos por MOREIRA et al. (2002): <i>overlap (before ou after)</i> , <i>override e wrap</i> .
<i>requisito</i>	Descrição de um serviço ou restrição que o sistema deve fornecer para que um usuário alcance algum objetivo [IEEE-STD. '610', 1990] ou para representar alguma informação característica do domínio do problema.
<i>requisito funcional</i>	Capturam o comportamento do sistema em termos de serviços, tarefas ou funções que o sistema deve fornecer [LOUCOPOULOS & KARAKOSTAS, 1995].
<i>requisito não-funcional</i>	Requisitos que não estão preocupados especificamente com a funcionalidade de um sistema. Eles impõem restrições no produto que está sendo desenvolvido ou no processo de desenvolvimento, ou ainda especificam restrições externas que o produto deve satisfazer [KOTONYA & SOMMERVILLE, 1998].
<i>template</i>	modelo ou protótipo de <u>artefato</u> que é usado para construir outros artefatos.
<i>unidade</i>	Cada <u>artefato</u> é criado usando abstrações ou construtores sintáticos que podem ser visualizados como <i>unidades</i> . O que compreende cada <i>unidade</i> depende do formalismo e do contexto usados, ou seja, uma unidade pode ser: um caso de uso, um diagrama UML, uma classe, uma função, etc.
<i>unidade base</i>	<u>unidade</u> do sistema afetada por um aspecto.