1
Introdução

A edição de janeiro/fevereiro de 2001 da *Technology Review*, a revista de inovação do MIT, dedicou-se à análise das dez principais tecnologias (*The Technology Review Ten*), "dez áreas de tecnologia emergentes que, em breve, terão um grande impacto na economia e na forma como vivemos e trabalhamos" [133]. Uma das dez áreas escolhidas foi a *Programação Orientada a Aspectos*, uma tecnologia promissora na era da programação pós-objeto.

Nesta tese, mudamos o foco dos mecanismos que oferecem suporte à tecnologia de programação orientada a aspectos para os *conceitos* e as *propriedades* que a caracterizam como um paradigma emergente para o desenvolvimento de software. Em particular, concentramo-nos em como esses conceitos e propriedades podem ser explorados nas primeiras etapas do design de software para construir sistemas de mais fácil compreensão, evolução e reutilização.

1.1
Contexto

A *Separação de concerns*[1] é um princípio fundamental que cuida das limitações da cognição humana ao lidar com essa complexidade. Esse princípio defende que, para contornar a complexidade, deve-se resolver uma questão importante (ou *concern*) por vez [36]. Na engenharia de software, o princípio de separação de *concerns* está normalmente relacionado à *decomposição* e à *modularização* de sistemas [114]: os sistemas de software complexos devem ser decompostos em unidades modulares menores e claramente separadas, cada uma lidando com um único *concern*. Os benefícios esperados são uma melhor compreensibilidade e um maior potencial para a evolução e reutilização em sistemas de software complexos.

[1] Usaremos o termo *concern* ao invés de *preocupação*.

Os mecanismos de composição e abstração que têm suporte de sucessivas gerações de linguagens de programação evoluíram para promover a expressão de soluções para problemas do mundo real de forma mais natural, assim como para possibilitar alcançar a separação de *concerns* no nível do código-fonte. A *Programação Orientada a Objetos* (POO) tem sido há anos a tecnologia de programação dominante, e seus benefícios são amplamente reconhecidos. No entanto, a orientação a objetos possui algumas limitações no tratamento de *concerns* que cuidam dos requisitos que envolvem restrições globais e propriedades sistêmicas, como a sincronização, a persistência, tratamento de erros, mecanismos de auditoria, entre outros. Essas limitações apontaram a necessidade de uma tecnologia que traga uma ruptura, *"algo que esteja realmente além de objetos"* [18]. Esse *algo* que deve estar além ou, no escopo desta tese, *ao lado* do modelo de objetos, tem sido o foco de muitas pesquisas nos últimos anos, caracterizando uma nova geração de tecnologias de separação de *concerns* , chamada *Separação Avançada de Concerns* (SAdC).

Programação Orientada a Aspectos (POA) [73] é uma tecnologia em evolução que oferece suporte a um novo tipo de separação de *concerns* no nível do código-fonte. A POA é evolucionária porque considera melhorias reconhecidas da separação de *concerns* proporcionadas por tecnologias anteriores (principalmente, mas não apenas, a programação orientada a objetos), enquanto oferece suporte a novos mecanismos para lidar com alguns *concerns* especiais que não são tratados adequadamente por essas tecnologias.

A idéia central por trás da POA é que os mecanismos de abstração e composição das tecnologias existentes não são suficientes para separar alguns *concerns* especiais encontrados em sistemas mais complexos. Esses *concerns* são chamadas de *concerns* transversais (*crosscutting concerns*) porque afetam a modularidade de outros elementos (chamados de *componentes*), "atravessando" seus limites. Sincronização, persistência e outros *concerns* listados anteriormente são exemplos de *concerns* transversais relativos à modularidade orientada a objetos. Sem os meios apropriados para a separação e a modularização, *concerns* transversais tendem a ficar espalhados e entrelaçados com outros *concerns*. As conseqüências naturais são uma menor compreensibilidade, uma menor evolutibilidade e menos reusabilidade dos artefatos do código. A POA usa *aspectos* como um novo mecanismo de modularização para a separação de *concerns* transversais e fornece um novo mecanismo para *combinar* aspectos em componentes em *pontos de combinação* bem definidos.

A POA e a maioria dos trabalhos existentes sobre a separação avançada de *concerns* tratam do fenômeno de *crosscutting* no nível de implementação. Recentemente, muitos grupos de pesquisa começaram a discutir a função dos aspectos em outras atividades do processo de desenvolvimento de software.

O *Desenvolvimento de Software Orientado a Aspectos* (DSOA) é uma área emergente cujo objetivo é promover a separação avançada de *concerns* ao longo de todo o ciclo de vida do desenvolvimento de software. O *Design Orientado a Aspectos* (DOA) é uma parte crítica do DSOA que se concentra em notações, técnicas e ferramentas para a identificação, a estruturação, a representação e o gerenciamento de *concerns* transversais em projeto e arquitetura de software, oferecendo um caminho desde os requisitos até a implementação com POA.

1.2 Declaração do Problema

A evolução de um novo estilo de programação normalmente evolui da programação na direção das atividades de análise e de projeto a fim de proporcionar um caminho completo ao longo do ciclo de vida do desenvolvimento do software. A programação orientada a aspectos está alcançando a maturidade depois de quase uma década de pesquisas [71, 88, 98, 73, 75] com um número crescente de aplicações, ferramentas e usuários [8]. Contudo, muitas questões devem ser resolvidas antes de alcançar o desenvolvimento de software orientado a aspectos. Em particular, observamos os problemas descritos a seguir, enquanto avaliávamos e analisávamos algumas abordagens de SAdC a fim de desenvolver uma aplicação de POA.

A necessidade de um arcabouço conceitual unificador para a POA.

Aspectos, *métodos adaptativos*, *filtros*, *sujeitos*, *hyperslices*: o uso de mecanismos lingüísticos separados para cuidar de algum "aspecto" do sistema difícil de capturar no modelo de objetos puro é comum a várias abordagens, e não apenas à POA. Alguns trabalhos relacionados incluem Programação Adaptativa (PA) [82, 84], *Filtros de Composição* (FC) [3, 15], *Programação orientada a Sujeitos* (POS) [59] e *Separação Multidimensional de Concerns* (SMDdC) [132, 113]. Além da terminologia, essas abordagens se distinguem de outras formas. Recentemente, a POA foi considerada uma possível convergência desses caminhos independentes da pesquisa em

separação de *concerns* avançada [40], mas ainda é necessária a definição de um framework conceitual unificador para a POA que também possa ser usado em outras abordagens.

Uma terminologia consistente e uma semântica clara são fundamentais para a compreensão, comunicação e validação de idéias. Em sua ausência, a essência e benefícios dos mecanismos de composição e abstração que lidam com *concerns* transversais podem ser atrapalhados pela diversidade das abordagens de pesquisa que ainda estão em desenvolvimento. A adoção de um framework conceitual unificador para a POA é uma etapa importante para a caracterização do projeto das linguagens orientadas a aspectos e para o fornecimento de suporte ao desenvolvimento de software orientado a aspectos.

A necessidade de um modelo de aspectos de nível mais alto.

A programação orientada a aspectos enfatiza o *uso* efetivo de ferramentas e mecanismos de linguagens particulares a fim de concretizar as propriedades e os conceitos relacionados a aspectos no nível da implementação, normalmente no escopo da programação orientada a objetos.

Entretanto, é bastante difícil especificar, comunicar e compreender detalhes importantes e sutis envolvidos na composição de aspectos e componentes apenas no nível do código-fonte. O desenvolvedor precisa entender e lidar com a estrutura e o comportamento de aspectos individuais, os relacionamentos entre aspectos e componentes, as possíveis interações entre aspectos, as possíveis interferências entre os mecanismos de composição novos e convencionais, a estrutura e o comportamento do sistema diante de componentes e aspectos combinados, e assim por diante.

Ainda não existe um modelo de aspectos *independente de linguagem* e *de nível mais alto* que omita os detalhes específicos à implementação e que enfatize os detalhes comportamentais e estruturais importantes no nível do design. Sem a devida atenção às propriedades e aos conceitos relacionados a aspectos no nível do design, pode ser difícil gerenciar a complexidade e aumentar a compreensibilidade, a evolução e a reusabilidade no desenvolvimento de software orientado a aspectos. A definição de um modelo de aspectos de nível mais alto é outra etapa importante para o suporte ao desenvolvimento de software orientado a aspectos.

A necessidade de linguagens de modelagem mais expressivas.

As linguagens de modelagem atuais não oferecem suporte à expressão de modelos de aspectos genéricos. Essas linguagens ou (i) não consideram aspectos como cidadãos de primeira classe em um modelo, e logo, não oferecem suporte à modelagem orientada a aspectos, ou (ii) oferecem mecanismos para modificar ou estender elementos de modelagem de primeira classe de UML (classes, pacotes) a fim de permitir a descrição separada de *concerns* transversais e, portanto, fornecer suporte limitado à modelagem orientada a aspectos, ou (iii) estão muito presas a um determinado modelo de implementação.

Abordagens recentes que seguem (ii) e (iii), no entanto, não levam em consideração várias questões importantes:

– **O status de primeira classe de aspectos.**

Os aspectos são cidadãos de primeira classe em linguagens de POA e devem ser tratados como tal em uma linguagem de modelagem que pretenda ser *orientada a aspectos*.

– **A dicotomia conceitual entre aspectos e classes [2].**

Os aspectos se diferenciam das classes devido à sua natureza *cross-scutting*. Isto é, os aspectos tendem a afetar várias classes; eles não oferecem serviços, e sim melhoram os serviços fornecidos pelas classes; as aspectos requerem um tipo diferente de mecanismo de composição para serem combinados a classes e objetos. Portanto, a especialização da abstração de classe para modelar a abstração de aspecto pode ser conceitualmente incorreta.

– **A necessidade de interfaces de aspectos claras.**

Como um novo tipo de unidade modular, um aspecto deve ter uma interface clara. A interface de aspectos deve descrever características comportamentais e estruturais que devem ser combinadas a classes e objetos. Além disso, ela também deve descrever os pontos de combinação de interesse e como eles são afetados pelos elementos *cross-cutting*. Quando essas interfaces são claras, elas promovem a previsibilidade da composição, aumentando a compreensão e a reutilização.

– **A necessidade de decomposição das interfaces de aspectos.**

Aspectos podem afetar as classes de forma diferente, ou seja, duas ou mais classes podem ser afetadas por diferentes subconjuntos de

[2]Nos itens a seguir, usaremos "classe" em vez de "componente", uma vez que as abordagens recentes são todas orientadas a objetos.

elementos *crosscutting* localizados dentro do mesmo aspecto. A decomposição da interface do aspecto em duas ou mais subinterfaces promove a previsibilidade da composição e melhora a compreensão.

– **A necessidade de uma representação explícita das interações entre aspectos.**

A programação orientada a aspectos é mais complexa do que uma simples combinação de aspectos ortogonais com classes em pontos de combinação bem definidos, um a um. De fato, a POA torna-se uma ferramenta interessante e poderosa exatamente quando os aspectos não são ortogonais, mas interferem construtiva ou negativamente. Os aspectos podem ter intra e interdependências. Eles podem interagir quando combinados ao mesmo ponto de combinação e assim por diante. Esses relacionamentos devem ser explicitamente representados a fim de aumentar a compreensão e facilitar a evolução.

– **A necessidade de uma semântica de composição consistente para crosscutting.**

Crosscutting entre aspectos e componentes é o principal problema da POA. Os aspectos afetam classes e fornecem melhorias comportamentais e estruturais. Essas melhorias podem ser alcançadas usando subclasses e herança, desde que não sejam sensíveis ao contexto. Logo, *crosscutting* e herança podem ser considerados mecanismos de extensão que podem entrar em conflito e interferir entre si. É necessária uma semântica de composição consistente para *crosscutting* a fim de descrever a intenção da composição e permitir o gerenciamento da propagação de mudanças e interferências.

– **A necessidade de uma descrição explícita das extensões sensíveis ao contexto.**

Os aspectos podem estender ou melhorar objetos em situações especiais, por exemplo, quando esses objetos interagem com determinado(s) objeto(s). Esse tipo de extensão sensível ao contexto pode não ser fácil de alcançar usando apenas mecanismos de composição convencionais; é uma característica diferenciadora da POA que deve receber o suporte de uma linguagem de modelagem orientada a aspectos.

– **A utilidade de uma visão do sistema depois do processo de combinação.**

A separação entre aspectos e classes aumenta a compreensibilidade do sistema que está sendo projetado. Todavia, as visões dinâmicas e

estáticas do sistema depois do processo de combinação são altamente desejáveis a fim de permitir uma avaliação inicial dos resultados da composição.

A Tabela 1.1 apresenta um resumo dos principais problemas tratados nesta tese. Nos capítulos a seguir, demonstraremos por que as abordagens existentes não resolvem essas questões de forma satisfatória.

Evidência	Problema
POA, PA, POS, FC, SMDdC (diversas abordagens)	Necessidade de arcabouço conceitual unificador
Ênfase em detalhes de implementação	Necessidade de um modelo de aspectos no nível de design
Aspectos não-ortogonais	Necessidade de especificação clara sobre relacionamentos e interações
Interferência entre *crosscutting* e herança	Necessidade de semântica de composição consistente para *crosscutting*
Regularidade de *crosscutting*	Necessidade de especificação clara de interfaces de aspectos
Crosscutting heterogêneo	Necessidade de decompor interfaces de aspectos

Tabela 1.1: Resumo da definição do problema.

1.3 Solução Proposta

Esta tese trata do desenvolvimento de software orientado a aspectos em uma etapa preliminar do design (ou projeto) de software, no domínio da modelagem orientada a objetos. Conforme discutido anteriormente, é bastante difícil compreender, especificar e comunicar detalhes importantes e sutis que fazem parte da separação e da composição de aspectos e componentes apenas no nível do código-fonte. Este trabalho propõe uma abordagem para o design de software com base nos conceitos e propriedades da programação orientada a aspectos. Em particular, ele se concentra na definição de um modelo de design orientado a aspectos e uma linguagem de modelagem orientada a aspectos, com o objetivo de permitir a construção de sistemas orientados a aspectos mais fáceis de compreender, desenvolver e reutilizar. Os termos *design orientado a aspectos* e *projeto orientado a aspectos* poderão ser usados no escopo de nossa abordagem.

Que novos conhecimentos, métodos ou tecnologias esta pesquisa gerará?

Como uma primeira etapa, adotamos um *framework conceitual* para a POA, cujas raízes estão nos conceitos originalmente apresentados em [73], a fim de caracterizar um espaço de projeto de linguagens orientadas a aspectos que não estão limitadas ou não estão necessariamente relacionadas a linguagens orientadas a objetos.

Segundo, propomos um *modelo de aspectos de nível mais alto* para a especificação e a racionalização de uma solução em termos de aspectos e propriedades relacionadas no nível do projeto, no escopo do projeto orientado a objetos. O modelo de aspectos abstrai detalhes específicos a linguagens de programação e enfatiza os detalhes comportamentais e estruturais importantes, incluindo os relacionamentos entre aspectos e componentes, as interações entre aspectos e a estrutura e o comportamento do sistema depois da combinação de aspectos e componentes. Como as tecnologias orientadas a aspectos têm como objetivo evitar o entrelaçamento e o espalhamento em modelos de objetos, principalmente no nível da implementação, ao fornecer novas abstrações que localizam aspectos *ao lado do* modelo de objetos, chamamos o modelo de aspectos proposto de *Modelo aSide*.

Finalmente, propomos *uma linguagem de modelagem para especificar e comunicar designs orientados a aspectos* que ofereça suporte ao modelo `aSide`. Essa linguagem é naturalmente chamada de Linguagem de Modelagem `aSide` ou simplesmente `aSideML`. A linguagem `aSideML` oferece semântica, notação e diretrizes que permitem que o projetista construa modelos comportamentais e estruturais nos quais os aspectos sejam explicitamente tratados como cidadãos de primeira classe. Esses modelos servem como um plano para a implementação com base em ferramentas e linguagens de programação orientadas a aspectos.

O modelo de aspectos de `aSideML` é definido como uma extensão do metamodelo de UML com novas metaclasses e metaassociações para a descrição de aspectos e crosscutting. A extensão proposta é apresentada seguindo um padrão similar usado para a especificação do metamodelo de UML [134].

Algumas características importantes fornecidas por nossa solução que a distinguem de outras abordagens são:

– A definição de interfaces transversais (ou *interfaces de crosscutting*) explícitas a fim de organizar a descrição de pontos de combinação e o comportamento de *crosscutting* dos aspectos.

– A definição de *aspectos* como elementos de modelagem parametrizados, com uma ou mais interfaces transversais explícitas.

– A definição de *crosscutting* como um mecanismo de extensão similar à herança, chamado *crosscutting vertical*, no qual as características de crosscutting são adicionadas, refinam ou redefinem as características das classes.

– A definição de *crosscutting* como um mecanismo de extensão que envolve múltiplas partes, chamado de *crosscutting horizontal* a fim de denotar crosscutting heterogêneo. A expressão dessa dimensão é facilitada pela adoção de interfaces transversais explícitas.

– A expressão de *crosscutting* sensível ao contexto.

– A definição de relacionamentos para modelar explicitamente os relacionamentos de dependência entre aspectos.

Como esses desenvolvimentos resolvem os problemas identificados?

Conforme discutido anteriormente, os mecanismos da POA fornecem vários benefícios, mas também introduzem alguns problemas. Nesta tese, defendemos que alguns desses problemas podem ser melhor tratados em um nível mais alto de abstração e identificamos algumas questões de modelagem importantes. Ilustramos que há algumas limitações nas técnicas e notações de modelagem atuais no tratamento de aspectos, *crosscutting* e dos problemas de modelagem identificados. Essas limitações podem obscurecer a percepção dos benefícios suportados pela POA no nível da implementação e também em outros níveis do processo de desenvolvimento de software. A abordagem de design orientada a aspectos proposta trata de um subconjunto dos problemas apresentados e melhora a compreensibilidade, a evolução e a reutilização dos modelos orientados a aspectos.

Compreensibilidade.

Os aspectos foram propostos como um mecanismo para a melhoria da separação de *concerns* e, portanto, para a promoção da compreensibilidade em sistemas de software complexos usando unidades modulares claramente separadas, cada uma lidando com um único *concern* .

Esta tese aborda alguns problemas remanescentes relacionados à compreensibilidade ao (i) adotar um framework conceitual unificador, (ii) fornecer um modelo de design orientado a aspectos independente da linguagem e

de alto nível, (iii) propor uma linguagem de modelagem que oferece suporte a aspectos como cidadãos de primeira classe, com interfaces transversais explícitas e relacionamentos também explícitos entre aspectos.

Evolução.

A POA oferece um melhor suporte para a evolução não antecipada por meio de adoção de um novo mecanismo de composição que não seja intrusivo e que forneça um tipo de composição *plug-in*. O modelo de design orientado a aspectos proposto oferece suporte a *crosscutting* como um novo relacionamento de composição que relaciona elementos de *crosscutting* a elementos básicos.

Reutilização.

Nosso trabalho trata de alguns problemas relacionados à reutilização de aspectos como o acoplamento léxico entre as especificações de aspectos e os nomes das classes, métodos etc. A definição de aspectos no nível de projeto como elementos parametrizados evita a dependência léxica nas classes básicas; ademais, a parametrização fornece uma alternativa para a herança entre aspectos no nível de projeto. Aspectos parametrizados, com interfaces transversais explícitas que documentam a interface de aspectos, promovem a reutilização dos mesmos.

1.4 Contribuições

As principais contribuições desta tese são:

1. Uma *teoria de aspectos* (o *modelo de aspectos*), um framework conceitual que representa uma descrição informal e preliminar das propriedades e dos conceitos fundamentais do desenvolvimento de software orientado a aspectos. A teoria de aspectos pode ser usada para avaliar o que é orientado a aspectos e o que não é e para ajudar no projeto de linguagens orientadas a aspectos. Essa teoria é apresentada no Capítulo 3.

2. A `aSideML`, uma linguagem de modelagem para especificar e comunicar designs orientados a aspectos nos quais aspectos e *crosscutting*

são explicitamente tratados como cidadãos de primeira classe. A linguagem `aSideML` é apresentada no Capítulo 5.

3. O metamodelo `aSide`, um modelo lógico que define a semântica dos modelos comportamentais e estruturais suportados pela `aSideML`. Ele é definido como um núcleo fundamental que oferece suporte a aspectos e crosscutting e que pode ser estendido para oferecer suporte a modelos específicos a linguagens. O modelo `aSide` é apresentado no Capítulo 6.

4. Um conjunto inicial de princípios e diretrizes que devem ser usados na modelagem orientada a aspectos. A linguagem `aSideML` é usada para ilustrar as diretrizes apresentadas no Capítulo 8.

5. O uso de ferramentas e linguagens padrão e atuais. Construímos sobre tecnologias aceitas pela indústria e definimos:

   - um modelo de aspectos no nível de projeto que usa o Modelo de Objetos de UML.
   - `aSideML`, uma linguagem de modelagem orientada a aspectos com base em uma extensão de UML.
   - um mapeamento preliminar de `aSideML` para AspectJ e Hyper/J.

Essas contribuições foram parcialmente descritas em alguns trabalhos e relatórios técnicos [27, 26, 22, 23, 24, 25] e são melhor ilustradas através de um estudo de caso apresentado no Capítulo 7.

1.5 Organização da Tese

Esta tese trata de problemas atuais relacionados ao design e à modelagem orientados a aspectos. A Figura 1.1 apresenta o roteiro desta tese.

A análise de algumas abordagens da programação orientada a aspectos (2, na caixa superior da Figura 1.1) resulta em um framework conceitual unificador para a POA, a teoria dos aspectos (3, na caixa intermediária da Figura 1.1). Para validar as idéias da teoria, foi proposto um metamodelo para a UML, o metamodelo `aSide` (6, na caixa inferior da Figura 1.1). A arquitetura do metamodelo `aSide` é diretamente derivada do modelo de aspectos. O metamodelo `aSide` fornece a semântica de `aSideML`, uma linguagem de modelagem para o design orientado a aspectos (5, na caixa inferior da Figura 1.1).

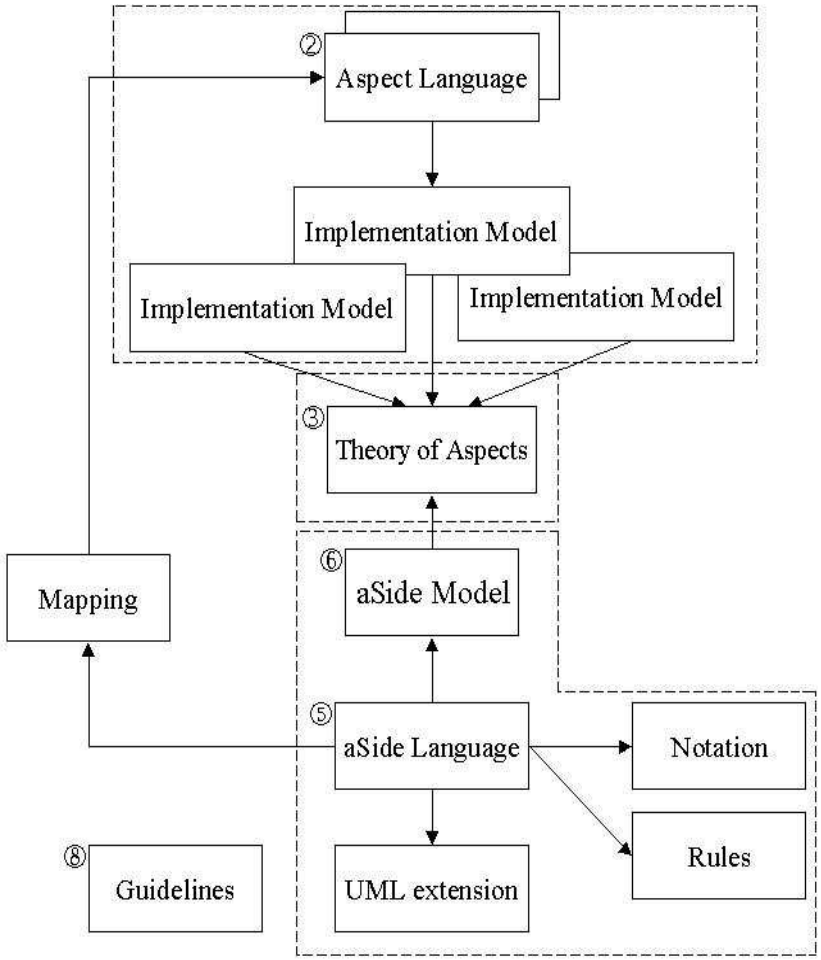

Figura 1.1: Roteiro da Tese.

Esta tese está organizada em nove capítulos e dois apêndices.

O Capítulo 2: Programação Orientada a Aspectos

apresenta uma introdução à programação orientada a aspectos e descreve as principais abordagens e os modelos de implementação relacionados.

O Capítulo 3: Uma Teoria de Aspectos

apresenta uma teoria informal de aspectos - um framework conceitual para analisar o problema com base em cinco conceitos principais: aspectos, componentes, pontos de combinação, crosscutting e processo de combinação. Essa teoria é usada para descrever e caracterizar linguagens orientadas a aspectos e, em especial, foi usada também para caracterizar e direcionar o projeto da linguagem de modelagem `aSideML` (Capítulo 5).

O Capítulo 4: Modelagem Orientada a Aspectos

ilustra alguns dos problemas descritos nesse Capítulo, motiva a necessidade de tratar a modelagem orientada a aspectos na etapa do projeto preliminar e introduz alguns requisitos para as linguagens de modelagem orientadas a aspectos. Algumas abordagens atualmente em uso para a modelagem orientada a aspectos são apresentadas, e suas limitações discutidas.

O Capítulo 5: A Linguagem de Modelagem `aSide`

apresenta a `aSideML`, uma linguagem de modelagem para especificar e comunicar designs orientados a aspectos. A `aSideML` oferece a semântica, a notação e as regras que permitem que o projetista construa modelos cujo foco sejam os principais conceitos, mecanismos e propriedades de sistemas orientados a aspectos, nos quais os aspectos e *crosscutting* sejam explicitamente tratados como cidadãos de primeira classe. A semântica da linguagem é fornecida pelo `modelo aSide`.

O Capítulo 6: O `Metamodelo aSide`

apresenta o `modelo aSide`, um modelo lógico que define a semântica dos modelos comportamentais, estruturais e dos processos de combinação suportados pela `aSideML`. O `modelo aSide` é definido como uma interpretação da teoria apresentada no Capítulo 3, na qual o modelo do componente é o Modelo de Objetos de UML. O Apêndice B apresenta trechos do metamodelo de UML (versão 1.4) usados como o modelo de componente, e o Apêndice C apresenta a especificação do `modelo aSide` usando uma abordagem de metamodelagem.

O Capítulo 7: Estudo de Caso: Portalware

apresenta um estudo de caso representativo no domínio de sistemas multiagentes, que envolve aspectos não-ortogonais e específicos de domínio (interação, autonomia, adaptação, colaboração etc.). Esse exemplo ilustra a adequabilidade de um design baseado na notação da `aSideML`.

O Capítulo 8: Princípios e Diretrizes para o Design Orientado a Aspectos

apresenta alguns princípios para o projeto de sistemas orientados a aspectos e diretrizes preliminares para a modelagem orientada a aspectos usando `aSideML`.

O Capítulo 9: Conclusões

discute as propostas e contribuições desta tese. Esse capítulo também discute os trabalhos futuros.

O Apêndice A

apresenta mapeamentos dos elementos da apresentação da `aSideML` para elementos do metamodelo `aSide`, AspectJ e Hyper/J, a fim de demonstrar a rastreabilidade entre as representações e a independência do modelo de projeto especificado pelo metamodelo `aSide` em relação a dois modelos de implementação específicos.

O Apêndice B

apresenta a especificação do modelo `aSide` usando uma abordagem de metamodelagem. O `metamodelo aSide` é um metamodelo que está em conformidade com UML e oferece uma extensão do Modelo de Objetos de UML.