6
O Metamodelo aSide

Este Capítulo descreve uma abordagem para tratar do problema da *necessidade de um modelo de aspecto de nível mais alto*, apresentado no Capítulo 1.

Propomos o modelo aSide, um modelo de aspectos *de alto nível*, *independente de linguagem* e que enfatiza a descrição adequada de um sistema completo em termos dos novos conceitos, propriedades e arquitetura introduzidos pelo projeto orientado a aspectos. O modelo aSide dá suporte à separação entre componentes e aspectos, oferecendo modelos semânticos que capturam de forma explícita as informações sobre o comportamento dinâmico e a estrutura estática de aspectos e seus relacionamentos com componentes e outros aspectos no nível do projeto. Ele permite que o projetista analise questões de alto nível do sistema sendo desenvolvido e serve como um plano para sua implementação usando ferramentas e linguagens de programação orientadas a aspectos.

O modelo aSide especifica a semântica dos modelos comportamentais e estruturais suportados por aSideML e, portanto, provê semântica para todas as notações de modelagem apresentadas no Capítulo 5.

O modelo de aspectos proposto é definido como uma instância da teoria de aspectos, o framework conceitual apresentado no Capítulo 3, no qual o modelo de componentes é o modelo de objeto suportado pela Unified Modeling Language (UML) [17]. Este Capítulo apresenta uma visão geral do modelo aSide e sua sintaxe abstrata, enquanto o Apêndice B descreve sua especificação usando uma abordagem de metamodelagem.

6.1
Visão Geral do Modelo aSide

O (meta)modelo aSide é um modelo lógico que define a semântica de modelos comportamentais e estruturais suportados por aSideML, uma linguagem de modelagem para especificar e comunicar projetos orientados a aspectos. Ele fornece um framework conceitual que incorpora as construções

e as regras necessárias para criar modelos de projeto orientados a aspectos para modelos de projeto orientados a objetos expressos em UML.

6.1.1 Objetivos

Os objetivos do metamodelo `aSide` são:

– **Fornecer um framework conceitual** que incorpora os principais conceitos do projeto orientado a aspectos e os relaciona aos principais conceitos do projeto orientado a objetos.

O metamodelo `aSide` oferece suporte à teoria de aspectos apresentada no Capítulo 3 e relaciona seus principais conceitos aos principais elementos orientados a objetos descritos pelo metamodelo de UML.

– **Fornecer uma base sólida** para a compreensão da linguagem de modelagem.

O metamodelo `aSide` segue a abordagem de UML e oferece uma definição rigorosa para `aSideML` usando diagramas de classes de UML, restrições bem definidas expressas em OCL e linguagem natural a fim de apresentar uma semântica detalhada.

6.1.2 Princípios de Projeto

O metamodelo `aSide` foi concebido com os seguintes princípios de projeto em mente:

– **Reutilização de conceitos de UML.**

O metamodelo `aSide` é construído sobre o metamodelo de UML. `Classifiers`, `Features`, `Collaborations` e outras construções definidas pelo metamodelo de UML são usadas e também são reutilizadas na definição de novas metaclasses.

– **Extensão conservativa de conceitos de UML.**

O metamodelo `aSide` usa e estende o metamodelo de UML sem contradizer sua semântica padrão. Por exemplo, os aspectos não são classes, portanto, a abstração de aspectos é especificada por uma nova metaclasse chamada `Aspect`. Por outro lado, classes são elementos base e, portanto, herdam da nova metaclasse `BaseElement`. Essas extensões não entram em conflito com a semântica padrão.

– **Independência de linguagem.**

O metamodelo `aSide` oferece suporte à teoria de aspectos; portanto, as construções e os relacionamentos definidos pelo metamodelo `aSide` representam as similaridades que podem ser mapeadas de diferentes modelos de implementação suportados pelas ferramentas e linguagens orientadas a aspectos.

– **Modularidade.**

O metamodelo `aSide` é organizado em um conjunto de pacotes coesos. Isso ajuda na compreensão do metamodelo.

6.1.3 Arquitetura

O metamodelo `aSide` é uma instanciação do framework conceitual apresentado no Capítulo 3, Figura 3.1. Ele pode ser visto como uma das camadas de uma arquitetura de modelagem de quatro camadas.

6.1.3.1 O metamodelo `aSide` e a teoria de aspectos.

O metamodelo `aSide` consiste em alguns submetamodelos que especificam os quatro modelos conceituais inter-relacionados apresentados no Capítulo 3, nos quais o metamodelo de UML exerce o papel do modelo de componentes. A Figura 6.1 apresenta o metamodelo `aSide`. As principais decisões relativas à organização do metamodelo `aSide` para que fique em conformidade com a teoria de aspectos são [26]:

– O modelo de componentes é o metamodelo de UML.

– O modelo de pontos de combinação especifica locais bem definidos relativos aos elementos de modelagem descritos pelo metamodelo UML. O metamodelo de pontos de combinação especifica pontos de combinação estáticos, em termos dos elementos usados nos modelos estruturais de UML, e pontos de combinação dinâmicos em termos dos elementos usados nos modelos comportamentais de UML.

– O metamodelo principal (*core*) inclui metaclasses que especificam os principais conceitos – *aspecto* e *crosscutting* – e metaclasses que descrevem conceitos auxiliares, como interface transversal , característica transversal etc. O metamodelo principal também especifica *relacionamentos entre aspectos* (`precedence` e `requirement`);

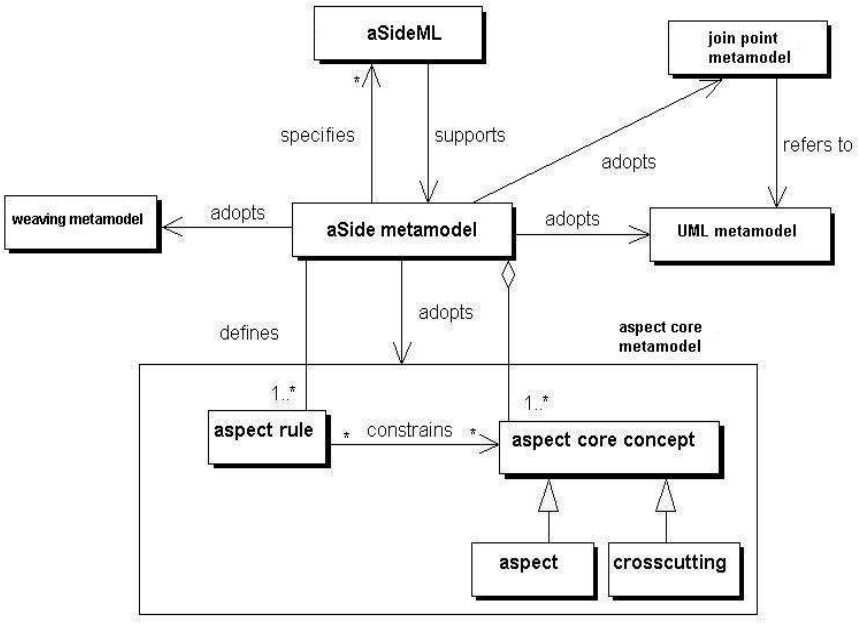

Figura 6.1: O metamodelo `aSide`.

– O metamodelo de processo de combinação inclui metaclasses abstratas. As metaclasses concretas devem ser fornecidas a fim de especificar modelos de implementação.

6.1.3.2 O metamodelo `aSide` e a arquitetura de metamodelo de quatro camadas.

O metamodelo de `aSideML` está em conformidade com a arquitetura de metamodelagem de quatro camadas [134] apresentada no Capítulo 4. Conforme apresentado na Tabela 6.1, as metacamadas são numeradas de M0 a M3, que corresponde ao MOF [92]. Um metamodelo (nível M2) é uma instância de um metametamodelo (nível M3). A principal responsabilidade da camada de metamodelo é definir uma linguagem para especificar modelos. Alguns exemplos de metaobjetos na camada de metamodelo são: `Class`, `Attribute`, `Operation` e `Aspect`.

6.1.3.3 O metamodelo `aSide` e UML.

O metamodelo `aSide` adota como seu modelo de componentes um subconjunto do metamodelo de UML. Esse subconjunto inclui o pacote

Metanível	Camada	Exemplos
M3	metametamodelo	Modelo MOF
M2	metamodelo	Metamodelo de UML Metamodelo `aSide`
M1	modelo	modelos em UML modelos em `aSide`
M0	objeto	sistemas modelados

Tabela 6.1: Arquitetura de metamodelagem de quatro camadas.

`Foundation`, o pacote `Common Behavior` e o pacote `Collaborations`; chamamos isso de *Modelo de Objetos de UML*.

O metamodelo de UML serve a dois propósitos:

– é usado como a estrutura principal (*backbone*) na qual se baseia a construção do metamodelo `aSide`; e

– é usado como o metamodelo para linguagens orientadas a objetos.

O metamodelo `aSide` é um metamodelo que está em conformidade com UML [1] e oferece uma extensão do Modelo de Objetos baseado em UML.

A especificação de UML também é usada das seguintes formas:

– Os diagramas de classes de UML são usados nas representações gráficas do metamodelo `aSide`; e

– a Object Constraint Language (OCL) pode ser usada a fim de representar restrições no metamodelo `aSide`.

6.1.4 Resumo da Abordagem

A especificação de modelos de aspectos de alto nível proposta e desenvolvida aqui é concretizada por um metamodelo que usa um subconjunto dos elementos de metamodelagem de UML como modelo de componentes e introduz novos elementos de metamodelagem a fim de descrever os principais conceitos do modelo principal: *aspectos* e *crosscutting*.

O metamodelo prescreve a dicotomia aspecto-base ao impor uma distinção explícita entre *elementos base* (*base elements*) e *elementos crosscutting* (*crosscutting elements*). As interações entre eles são restritas a pontos

[1] "As unidades básicas de conformidade com UML são os pacotes que definem o metamodelo de UML. A menos que seja qualificado de outra forma, estar em conformidade com um pacote requer a conformidade com sua sintaxe abstrata, regras bem definidas, semântica e notação" [135].

distintos de interação chamados de *pontos de combinação* (*join points*) (Figura 6.2).

Os elementos *crosscutting* são relacionados a elementos base por meio de um relacionamento direcionado, o *relacionamento de crosscutting* (*crosscutting relationship*).

Os elementos *crosscutting* e os elementos base são compostos de acordo com uma *estratégia de combinação* (*weaving strategy*). A estrutura geral do sistema composto resultante (estático e dinâmico) é capturada por *elementos combinados* (*woven elements.*).

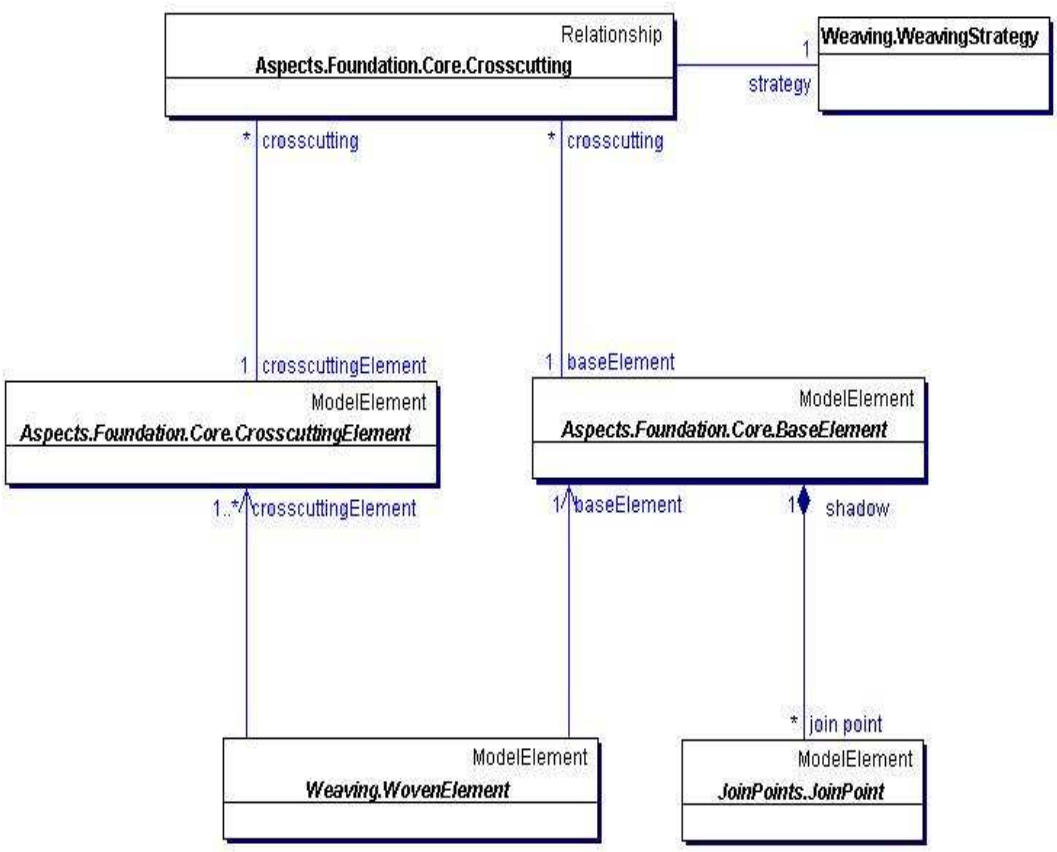

Figura 6.2: Resumo da abordagem.

6.1.5 Organização

O metamodelo `aSide` é organizado em torno de alguns pacotes de alto nível que lidam com seus submetamodelos (Figura 6.3). O pacote *Aspects* é organizado em dois subpacotes que lidam com a modelagem comportamental e estrutural dos principais conceitos de aspectos (Figura 6.4).

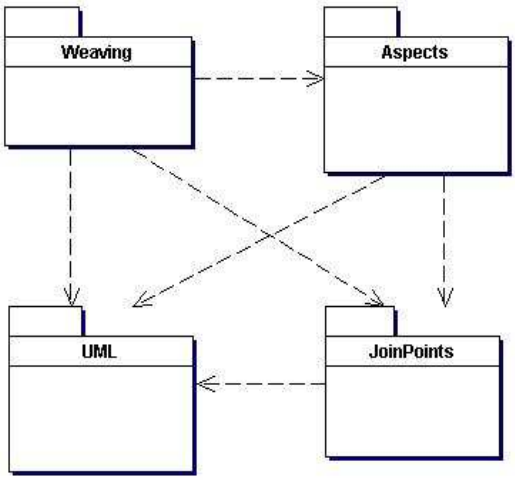

Figura 6.3: Pacotes de alto nível.

Cada pacote agrupa conjuntos de metaclasses que podem ser abstratas ou concretas. As metaclasses abstratas não são instanciáveis e são comumente usadas para concretizar construções-chave, compartilhar estrutura e organizar o metamodelo. Já as metaclasses concretas são instanciáveis e refletem as construções que podem ser usados pelas linguagens de modelagem.

6.2 Aspect Core

O pacote *Aspects.Foundations.Core*, ou simplesmente, o pacote *Aspect Core*, define as construções básicas, abstratas e concretas, do metamodelo necessárias para a criação de modelos de aspectos. As construções abstratas definidas em *Aspect Core* incluem `BaseElement`, `CrosscuttingElement` e `CrosscuttingFeature`. As construções concretas especificadas em *Aspect Core* incluem `Aspect`, `Crosscutting` e `CrosscuttingInterface`.

A Figura 6.5 mostra os elementos de modelagem que constituem a estrutura principal (*backbone*) do metamodelo `aSide`.

BaseElement

Um elemento base é um elemento nomeado que pode participar de um relacionamento de *crosscutting* com um elemento *crosscutting* e, portanto, pode ser afetado por um elemento *crosscutting* em pontos de combinação bem definidos.

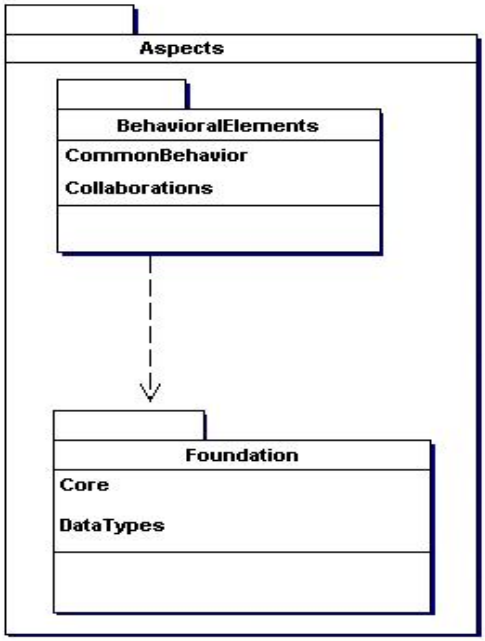

Figura 6.4: O pacote Aspects.

Um elemento base pode participar de vários relacionamentos de *crosscutting*. `BaseElement` é uma metaclasse abstrata que especifica elementos base.

CrosscuttingElement

Um elemento *crosscutting* é um elemento nomeado que pode participar de um relacionamento de *crosscutting* com um elemento base e, portanto, pode afetá-lo por meio de *crosscutting*.

Um elemento *crosscutting* pode participar de vários relacionamentos de *crosscutting*.

`CrosscuttingElement` é uma metaclasse abstrata que especifica elementos *crosscutting*. Um `CrosscuttingElement` com pelo menos uma associação `TemplateParameter` é um elemento *crosscutting* parametrizado.

Crosscutting

Crosscutting é um relacionamento de um elemento *crosscutting* com um elemento base que especifica que o elemento *crosscutting* afeta a estrutura ou comportamento do elemento base em locais bem definidos.

Quando o elemento *crosscutting* é parametrizado, o relacionamento de *crosscutting* possui um conjunto de casamentos de parâmetros de template.

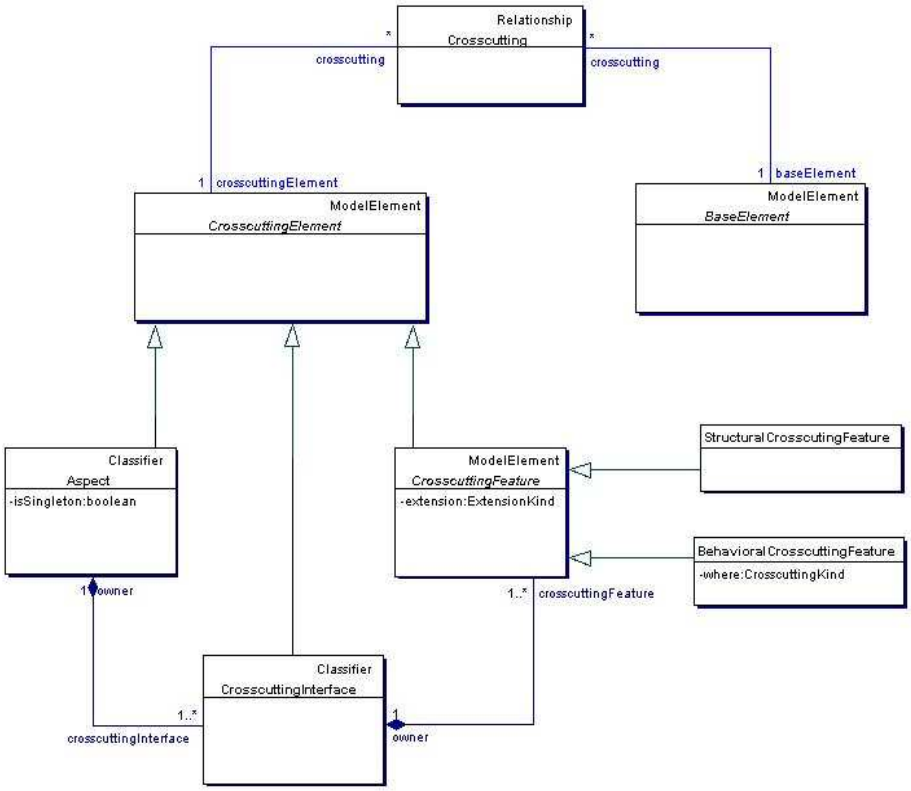

Figura 6.5: Aspect Core: *Backbone*.

Cada casamento de padrão (ou correspondência) associa um parâmetro formal declarado no elemento *crosscutting* com parâmetros reais que indicam os locais nos elementos base em que outras estruturas e comportamentos serão combinados (Figura 6.6).

`Crosscutting` é uma metaclasse concreta que especifica relacionamentos de *crosscutting*.

TemplateMatch

Um casamento de template (*template match*) relaciona o(s) parâmetro(s) atual(is) a um parâmetro de template formal como parte de um relacionamento de *crosscutting*.

`TemplateMatch` é uma metaclasse que associa um ou mais parâmetros reais a um parâmetro de template formal dentro do contexto de um relacionamento de *crosscutting*.

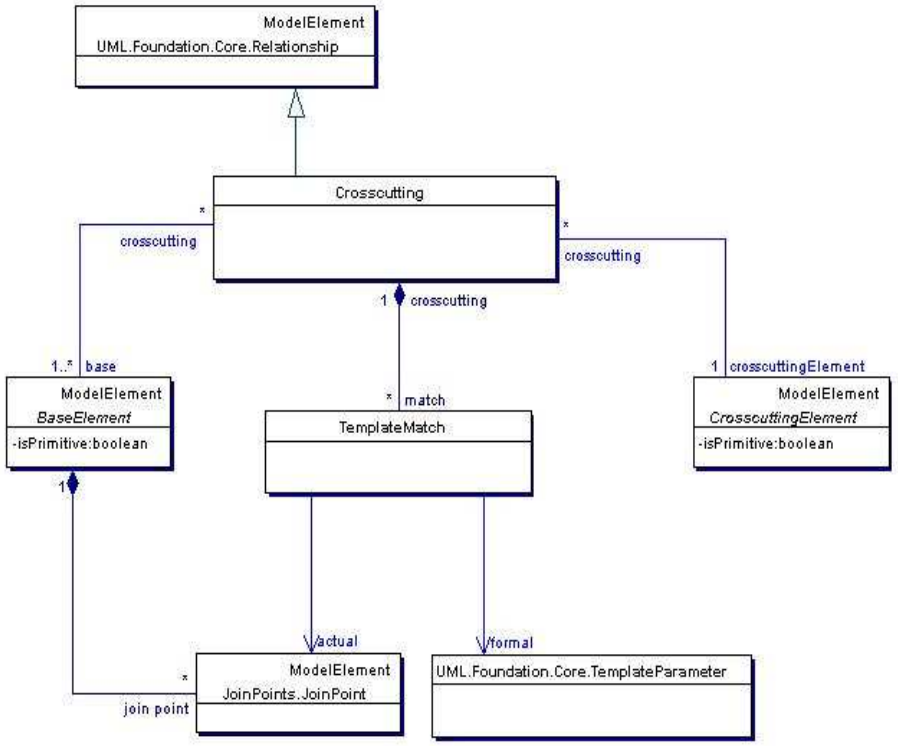

Figura 6.6: Relacionamento de *crosscutting* .

Aspect

Um aspecto é uma descrição de um conjunto de características, organizadas em interfaces transversais, que afetam a estrutura e o comportamento de classes por meio de *crosscutting* de modo sistêmico. Um aspecto deve ter no mínimo uma interface transversal.

Um *aspecto parametrizado* abstrai a identidade das classes que irá afetar, declarando parâmetros formais a fim de manter os nomes de classes e métodos.

`Aspect` é uma metaclasse concreta que especifica aspectos. É uma especialização de `CrosscuttingElement` e `Classifier`.

A Figura 6.7 mostra os elementos de modelagem que definem aspectos.

CrosscuttingInterface

Uma interface transversal (*crosscutting interface*) é um conjunto nomeado de características transversais e operações necessárias que caracterizam conjuntos de comportamentos e estruturas *crosscutting* coesos definidos dentro de aspectos.

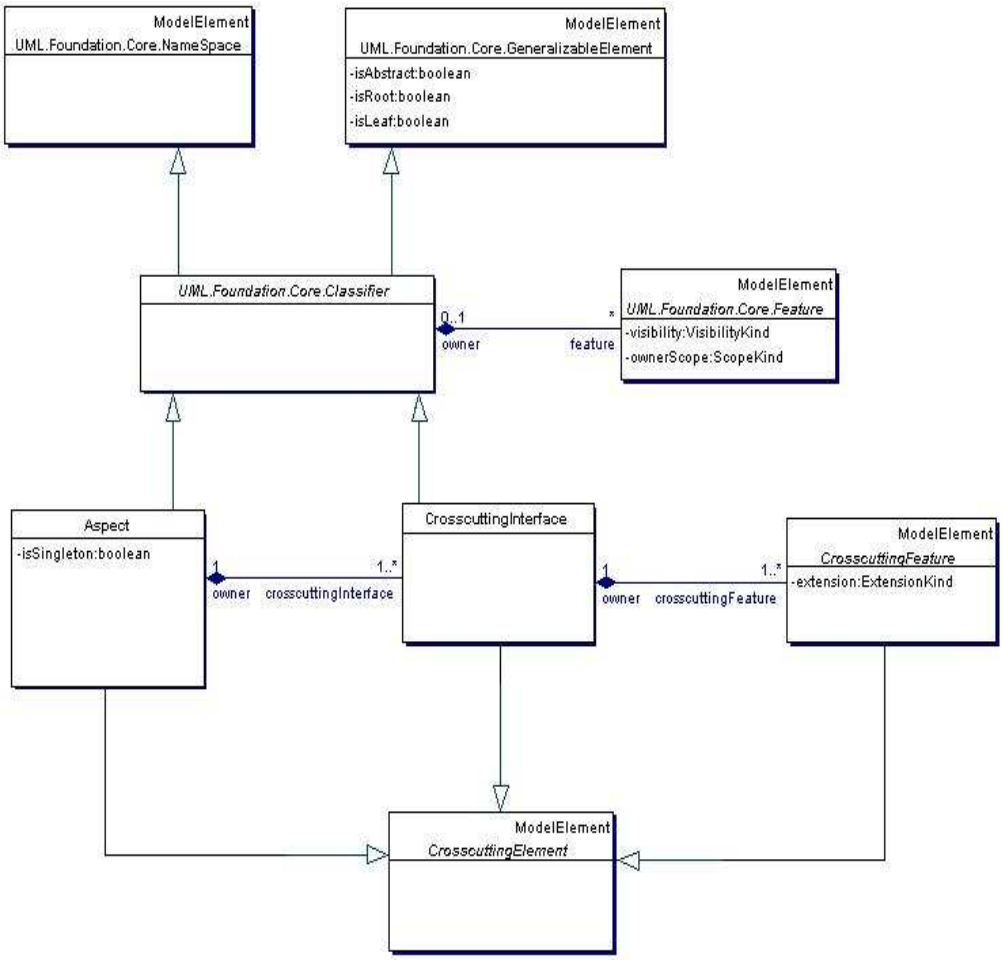

Figura 6.7: Aspect Core: Aspectos

Uma característica transversal declara uma característica estrutural ou comportamental que afeta elementos base. As características requisitadas especificam serviços que um aspecto precisa usar a fim de realizar sua função.

`CrosscuttingInterface` é uma metaclasse concreta.

CrosscuttingFeature

Uma característica transversal especifica a estrutura ou o comportamento que afetam ou melhoram a estrutura ou o comportamento de instâncias base no contexto de um relacionamento de *crosscutting*.

As melhorias caracterizam-se como adições, refinamentos ou redefinições. As adições são melhorias que oferecem nova estrutura ou comportamento aos elementos base. Os refinamentos são melhorias que estendem o comportamento base. As redefinições são melhorias que sobrepõem ou redefinem o comportamento base. As características transversais são declaradas em uma interface transversal.

`CrosscuttingFeature` é uma metaclasse abstrata.

StructuralCrosscuttingFeature

Uma característica transversal estrutural é uma característica transversal que especifica melhorias estruturais para instâncias base no contexto de um relacionamento de *crosscutting*. As melhorias estruturais caracterizam-se como *adições*.

`StructuralCrosscuttingFeature` é uma metaclasse abstrata.

BehavioralCrosscuttingFeature

Uma característica transversal comportamental é uma característica transversal que especifica comportamentos para melhorar o comportamento de instâncias base no contexto de um relacionamento de *crosscutting*. As melhorias caracterizam-se como adições, refinamentos ou redefinições.

As características transversais comportamentais são especificadas como elementos parametrizados, isto é, elas contêm um ou mais parâmetros livres que representam a assinatura e o nome da operação. Uma operação base pode ser explicitamente invocada dentro do corpo da operação do aspecto usando o pronome *base*. As características transversais comportamentais possuem colaborações aspectuais associadas que descrevem como o comportamento *crosscutting* e o comportamento base são compostos.

`BehavioralCrosscuttingFeature` é uma metaclasse concreta.

Elementos Padrão

A Tabela 6.2 contém uma lista de elementos padrão predefinidos para o metamodelo `aSide`.

Elemento Padrão	Elemento Base	Tipo	Descrição
Precedence	Dependency	Stereotype	Precedência entre aspectos
Require	Dependency	Stereotype	Requisito entre aspectos
Xor	Crosscutting	Constraint	Exclusão mútua entre aspectos

Tabela 6.2: Elementos padrão.

6.3
Elementos Comportamentais

O pacote *Aspects.BehavioralElements* define elementos de metamodelagem concretos e abstratos para a criação de modelos comportamentais.

Os elementos concretos especificados nesse pacote incluem `AspectInstance`, `Aspectual Collaborations` e `Aspectual Interactions`.

A sintaxe abstrata para o pacote *Aspects.BehavioralElements* é expressa em notação gráfica nas Figuras 6.8, 6.9 e 6.10.

AspectInstance

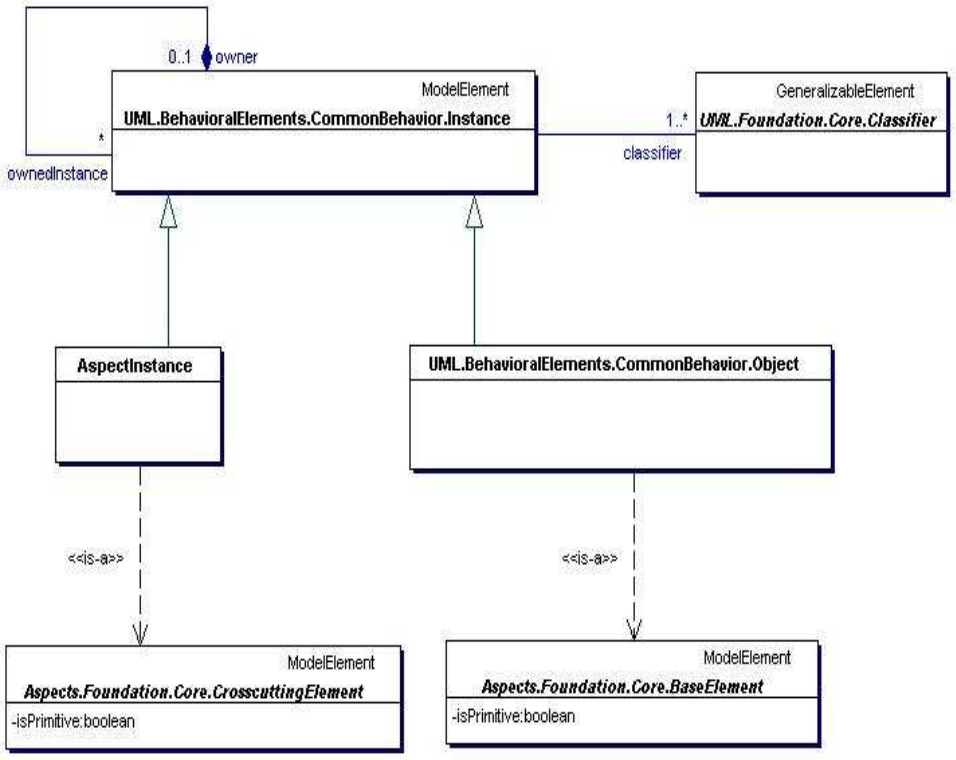

Figura 6.8: Instâncias de aspecto.

Uma instância de aspecto é uma instância originada de um aspecto.

No metamodelo, uma `AspectInstance` é um elemento `Instance` (de UML) originado de exatamente um elemento `Aspect`.

`AspectInstance` é uma metaclasse concreta.

AspectualCollaboration

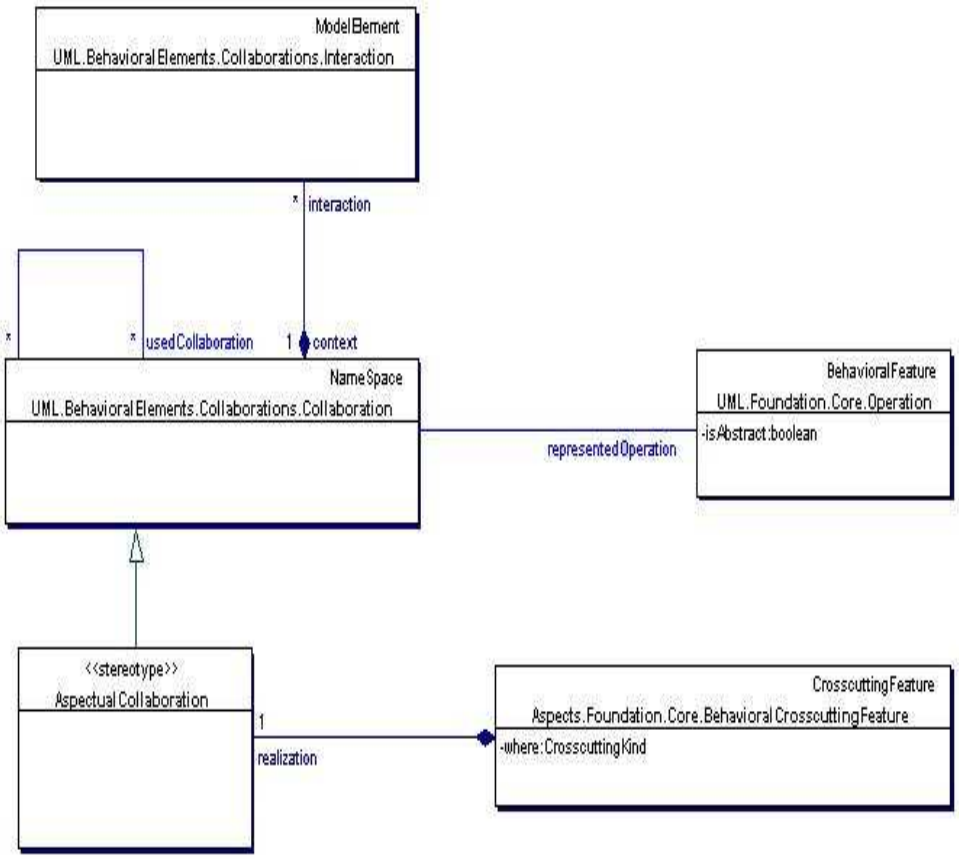

Figura 6.9: Colaborações aspectuais.

Uma *colaboração aspectual* é uma descrição de uma organização geral de objetos e instâncias de aspectos que interagem dentro de um contexto a fim de implementar o comportamento *crosscutting* de uma característica transversal comportamental. Uma colaboração aspectual modela a realização de uma característica transversal comportamental.

`AspectualCollaboration` é uma metaclasse. Ela estende `Collaboration` (de UML) e oferece algumas novas restrições.

AspectualInteraction

Uma *interação aspectual* é uma especificação comportamental que incorpora uma seqüência de comunicações trocadas entre um conjunto de objetos e uma instância de aspecto a fim de conseguir a implementação de uma característica transversal comportamental que refina ou redefine o comportamento base. As interações aspectuais são definidas no contexto das colaborações aspectuais.

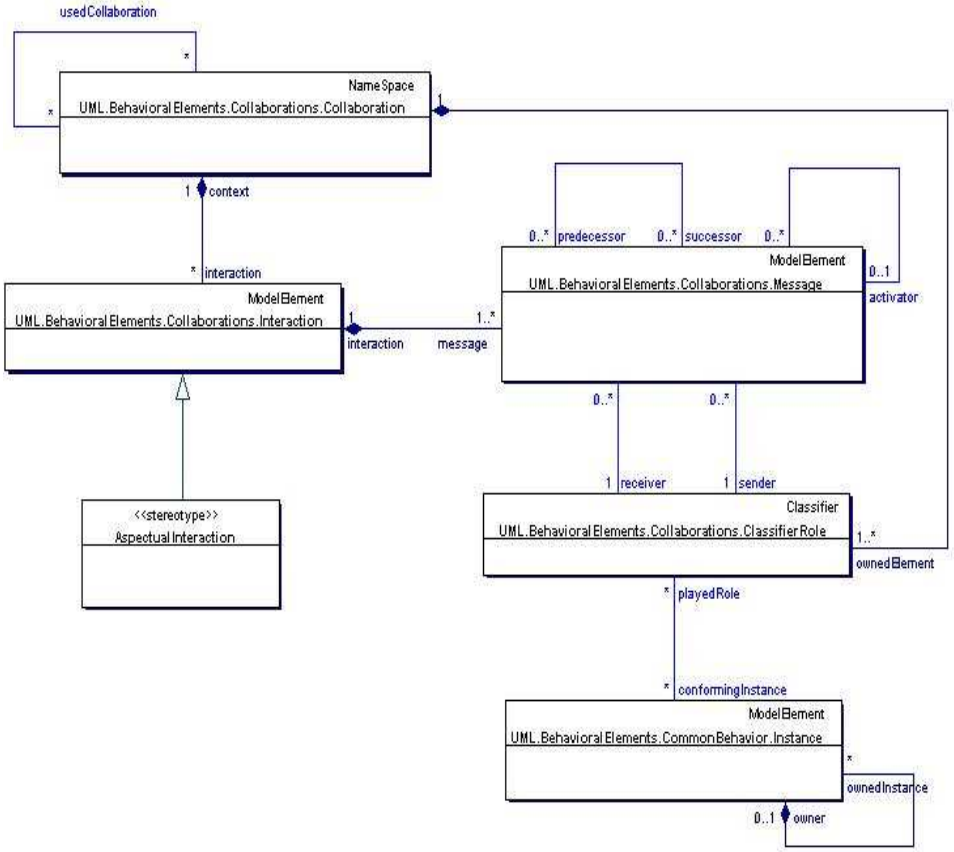

Figura 6.10: Interações aspectuais.

Interações aspectuais se baseiam em quatro padrões de interação diferentes. Um padrão de interação é uma interação parametrizada. Há quatro tipos de padrão de interação: *refine-before*, *refine-after*, *refine-around* e *redefine-around*.

`AspectualInteraction` é uma metaclasse. Ela estende `Interaction` (de UML) e oferece algumas novas restrições.

6.4 Considerações Finais

Neste Capítulo, apresentamos uma visão geral do metamodelo `aSide` e concentramo-nos em sua estrutura principal. O Apêndice B descreve sua especificação usando uma abordagem de metamodelagem, organizada como um guia de referência.

O metamodelo `aSide` está parcialmente definido em termos da especificação do metamodelo de UML, v1.4 [134].

Fornecemos uma avaliação do metamodelo `aSide` usando os seguintes critérios: escopo, extensibilidade e qualidade de documentação.

– Escopo do metamodelo: *estão definidos todos os conceitos de que precisamos?*

  Os objetivos do metamodelo `aSide` foram apresentados em 6.1. O primeiro objetivo, *fornecer um framework conceitual unificador que incorpore os principais conceitos do projeto orientado a aspectos e que os relaciona aos principais conceitos do projeto orientado a objetos* definiu o *escopo* do metamodelo. As Seções B.1 até B.6 descrevem em detalhe os conceitos que são suportados pelo metamodelo `aSide`. Estes conceitos são os mesmos listados na Tabela 3.2: *Conceitos e propriedades definidos no modelo de aspectos* e apresentados na Seção 3.2.

– Extensibilidade: *está claro onde deve entrar um novo elemento e como o metamodelo oferece suporte a essa adição? É possível fazer isso sem introduzir inconsistências?*

  Conforme mencionado na Seção 6.1.2, o metamodelo oferece uma extensão conservativa ao metamodelo de UML, v1.4. Além disso, a provisão de `BaseElement` e `CrosscuttingElement` como metaclasses abstratas oferece suporte à extensibilidade.

  Alterações realizadas nas versões subseqüentes de UML não serão consideradas nesta tese, mas devem ser incorporadas em trabalhos futuros da área.

– Qualidade de documentação: *é possível entender o que significa um conceito, mesmo sem contexto?*

  A documentação do metamodelo `aSide` segue um padrão adaptado da especificação do metamodelo de UML, v1.4 [134].