5
A Linguagem de Modelagem aSideML

Este Capítulo descreve uma abordagem para tratar do problema da *necessidade de linguagens de modelagem mais expressivas*, apresentado no Capítulo 1. Em particular, ele apresenta aSideML, uma *linguagem de modelagem construída para especificar e comunicar designs orientados a aspectos.*

A linguagem aSideML oferece semântica, notação e 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. Esses modelos ajudam a lidar com a complexidade de sistemas orientados a aspectos, ao fornecer visões essenciais da estrutura e do comportamento que enfatizam o papel dos elementos *crosscutting* e seus relacionamentos com outros elementos. Esses modelos também servem como planos preliminares (*blueprints*) que devem ser desenvolvidos na direção dos modelos de implementação de ferramentas e linguagens de programação orientadas a aspectos.

O Capítulo 6 descreve a semântica de aSideML, enquanto este capítulo enfatiza a notação (a sintaxe gráfica para expressar modelos de aspectos). O Capítulo 8 apresenta diretrizes para o design orientado a aspectos com aSideML.

Organização do Capítulo

A Seção 5.1 apresenta uma visão geral dos principais elementos da linguagem aSideML. As Seções 5.2, 5.3 e 5.4 descrevem os modelos estruturais, comportamentais e composicionais que podem ser construídos usando aSideML. Para cada um desses modelos, os tipos de diagrama e os elementos de apresentação correspondentes são descritos. Seguimos o formato usado no Guia de Notação da Especificação de UML [134]. Para cada tipo de diagrama, os elementos de modelagem encontrados

naquele diagrama e suas representações são listados. Para cada elemento de modelagem, as informações detalhadas a seguir são fornecidas:

- **Semântica**: um breve resumo da semântica de cada elemento de modelagem. Outros detalhes são fornecidos no Capítulo 6.
- **Notação**: descrição da representação notacional e suas variações.
- **Opções de apresentação**: descrição de opções na apresentação das informações dos elementos de modelagem e sugestões de formas alternativas de apresentar as informações dentro de uma ferramenta.
- **Exemplo**: ilustração de uso de um diagrama ou elemento de notação.
- **Mapeamento**: mapeamento de elementos de notação para os elementos de metamodelo correspondentes (Capítulo 6) é fornecido no Apêndice A.

A apresentação de `aSideML` é informal e baseia-se em um único exemplo: o padrão de projeto Observer [46], modelado com objetos e aspectos. Esse exemplo foi usado para apresentar trabalhos relacionados na Seção 4.6.

5.1 Visão Geral da Linguagem `aSideML`

A `aSideML` é uma linguagem de modelagem desenvolvida para especificar e comunicar designs orientados a aspectos. Ela oferece notação, semântica e regras com o principal propósito de tratar da modelagem conceitual de um sistema em termos de aspectos e *crosscutting* considerando o Modelo de Objetos de UML.

O nome `aSide` vem do termo em inglês *aside* (*to or toward the side*), para denotar elementos que estão `ao lado de` outros. O nome pode também ser interpretado como denotando "um lado" (*a side*), no sentido de uma *faceta* ou um *aspecto*.

5.1.1 Objetivos

Os objetivos apresentados a seguir estão relacionados ao projeto de `aSideML`.

– fornecer uma linguagem de modelagem visual expressiva a fim de desenvolver e comunicar modelos de aspectos.

  O Capítulo 4 apresentou os problemas relacionados a esse objetivo e que são tratados por `aSideML`.

– fornecer semântica e notação a fim de oferecer suporte aos novos requisitos de modelagem introduzidos pelos aspectos.

  O Capítulo 4 apresentou uma discussão sobre os novos requisitos de modelagem colocados pela MOA.

– fornecer um framework conceitual que incorpora o consenso da comunidade orientada a aspectos sobre os principais conceitos do DSOA e uma base rigorosa para compreender a linguagem de modelagem.

  A especificação de `aSideML` segue a abordagem de UML, ou seja, é usado um metamodelo a fim de especificar a semântica de seu modelo de aspectos (Capítulo 6); o metamodelo `aSide` oferece suporte à teoria de aspectos (Capítulo 3) e, portanto, fornece um framework conceitual e unificador para o DSOA.

– oferecer suporte às especificações independentes de linguagem que podem ser mapeadas para linguagens de programação orientadas a aspectos particulares.

– seguir os padrões dindústria .

  A `aSideML` é uma linguagem baseada em UML [17], a linguagem de modelagem visual preferida para a construção de sistemas orientados a objetos.

5.1.2 Artefatos

A `aSideML` oferece elementos de apresentação para a maioria dos elementos de modelagem descritos no Capítulo 6. Esses elementos de apresentação são organizados em diagramas gráficos que, juntos, fornecem várias visões do sistema em desenvolvimento. O modelo de aspectos subjacente integra essas visões de forma que possa ser construído um sistema autoconsistente. A Tabela 5.1 mostra os modelos de `aSideML` e os diagramas que os exibem, bem como os novos elementos de modelagem e as perspectivas relevantes para cada diagrama.

Modelo	Diagramas	Perspectivas	Elementos
estrutural	diagrama de aspectos		aspecto, interface transversal, característica transversal
	diagrama de classe estendido	centrado em aspecto	aspecto, *crosscutting* interface transversal, *order*
		centrado na base	
comportamental	diagrama de sequência estendido	ponto de combinação	ponto de combinação dinâmico
	diagrama de colaboração aspectual		instância de aspecto, colaboração aspectual
	diagrama de sequência		instância de aspecto, interação aspectual
processo de combinação	diagrama de classes combinadas		classe combinada
	diagrama de colaboração combinada		colaboração combinada
	diagrama de sequência combinada		interação combinada

Tabela 5.1: Dimensões da modelagem orientada a aspectos com `aSideML`.

Elementos de Modelagem

Em `aSideML`, os elementos de modelagem definidos pelo usuário podem ser *estruturais* ou *comportamentais*.

Os principais elementos estruturais de modelagem de `aSideML` são *aspectos*, *os elementos base (de UML)* que aspectos afetam e seus *relacionamentos*. Os aspectos são definidos como elementos parametrizados que abstraem em relação à identidade das classes que irão afetar, declarando parâmetros de templates a fim de manter os nomes reais de classes e dos

métodos. O relacionamento *crosscutting* representa um relacionamento entre um aspecto parametrizado e um elemento base; ele também realiza uma associação que define as operações e os elementos base que substituem os parâmetros de templates do aspecto.

Os principais elementos *comportamentais* de modelagem são instâncias de *aspecto, interações aspectuais* e *colaborações aspectuais.*

A `aSideML` também fornece *elementos de modelagem composicionais*, definidos para descrever elementos do processo de combinação. Os elementos de modelagem composicionais são elementos especiais que descrevem o resultado dos elementos *crosscutting* do processo de combinação e elementos base. Os modelos de processo de combinação são gerados automaticamente a partir de modelos de objetos e modelos de aspectos, em conformidade com a estratégia de combinação, e dependem do suporte de ferramentas. Os principais elementos de modelagem composicionais são *classes combinadas* (*woven classes*), *colaborações combinadas* (*woven collaborations*) e *interações combinadas* (*woven interactions*).

Diagramas

A `aSideML` fornece novos diagramas e enriquece alguns diagramas de UML a fim de apresentar os elementos *crosscutting* e seus relacionamentos para os elementos base:

- **Diagramas de aspecto**

  Um diagrama de aspecto fornece uma descrição completa de um aspecto. A descrição incorpora as interfaces transversais , características locais e relacionamentos de herança. Cada característica transversal comportamental pode ser visualizada em um diagrama de colaboração aspectual.

- **Diagramas de colaboração aspectuais**

  Um diagrama de colaboração aspectual oferece uma apresentação gráfica de uma colaboração aspectual, um tipo especial de colaboração que modela a realização de uma operação transversal definida dentro de um aspecto. Esse diagrama oferece suporte à visão de interação (*interaction view*) que envolve instâncias de aspectos e elementos base.

- **Diagramas de seqüência aspectuais**

  O diagrama de seqüência aspectual oferece uma apresentação gráfica de um conjunto de mensagens organizadas em seqüências temporais,

sendo que algumas dessas mensagens denotam invocações a operações de aspectos, sob a perspectiva de aspectos. Esse diagrama oferece suporte à visão de interação que envolve instâncias de aspectos e elementos base.

– **Diagramas de processo de combinação**

Um diagrama de processo de combinação oferece uma apresentação gráfica para um grupo de elementos combinados – elementos base adornados de forma a enfatizar as melhorias proporcionadas pelos elementos *crosscutting* . Os elementos combinados podem ser especializados para cada modelo de implementação disponível (AspectJ, Hyper/j etc.).

– **Diagramas de classes estendidos**

Um diagrama de classes estendido (*enhanced class diagram*) oferece uma apresentação gráfica da visão de projeto estático de um sistema em que as classes e os aspectos residem como cidadãos de primeira classe. Cada aspecto pode ser visualizado em detalhe e separadamente em um diagrama de aspecto correspondente.

Perspectivas

A `aSideML` também propõe perspectivas a fim de apresentar os elementos *crosscutting* e seus relacionamentos em alguns diagramas:

– **Perspectiva centrada em aspectos**

Uma perspectiva centrada em aspectos oferece uma apresentação gráfica da visão de projeto estático de um sistema em que o foco principal é a descrição de um aspecto e seus relacionamentos com os elementos base.

– **Perspectiva centrada em base** Uma perspectiva centrada em base oferece uma apresentação gráfica da visão de projeto estático de um sistema em que o foco principal é a descrição de um elemento base e os aspectos que o melhoram. Os diagramas centrados em base apresentam informações explícitas sobre a dependência entre elementos *crosscutting* que afetam um elemento base.

– **Perspectiva de pontos de combinação**

Uma perspectiva de ponto de combinação oferece uma apresentação gráfica para coleção de elementos base em que os pontos de combinação são apresentados explicitamente.

5.2 Modelagem Estrutural

Em aSideML, a modelagem estrutural oferece a visão estática de um sistema na presença dos aspectos. Os principais constituintes dos modelos estruturais são as classes, aspectos e seus relacionamentos.

A visão estática descreve as características transversais de aspectos como elementos de modelagem discretos, organizados em *interfaces transversais*. O comportamento dinâmico dessas características é descrito por modelos comportamentais (Seção 5.3).

5.2.1 Aspectos

Um *aspecto* é uma descrição de um conjunto de características que melhoram a estrutura e o comportamento de classes por meio de *crosscutting* de forma sistêmica. A aSideML oferece uma notação gráfica para a declaração e o uso de aspectos. Os aspectos são declarados em diagramas de aspectos e podem ser usados em outros diagramas, conforme explicado mais adiante.

Semântica

Um aspecto representa um *concern* transversal dentro do sistema que está sendo modelado. Os aspectos podem ter comportamento local e estrutura de dados local. Eles definem a estrutura e o comportamento, organizados em interfaces transversais, que serão combinados com a estrutura e o comportamento de classes por meio de *crosscutting*.

Um aspecto deve ter no mínimo uma interface transversal. Os aspectos podem melhorar as classes de forma heterogênea, ou seja, duas ou mais classes diferentes podem ser afetadas por diferentes subconjuntos de características transversais localizadas dentro do mesmo aspecto. As interfaces transversais modularizam a descrição de subconjuntos de características transversais e promovem o *crosscutting* horizontal (Seção 3.1.3.2).

Os aspectos também podem ter impactos variados sobre diferentes tipos de classes. Portanto, um aspecto é definido como um *elemento parametrizado* que provê abstração em relação à identidade das classes que irá afetar, declarando parâmetros formais a fim de guardar os nomes reais de classes e métodos. Podemos pensar em aspectos que não são elementos

parametrizados; nesse caso, o aspecto estaria acoplado a uma determinada classe.

A semântica detalhada de aspectos é apresentada na Seção B.1.3.5.

Notação

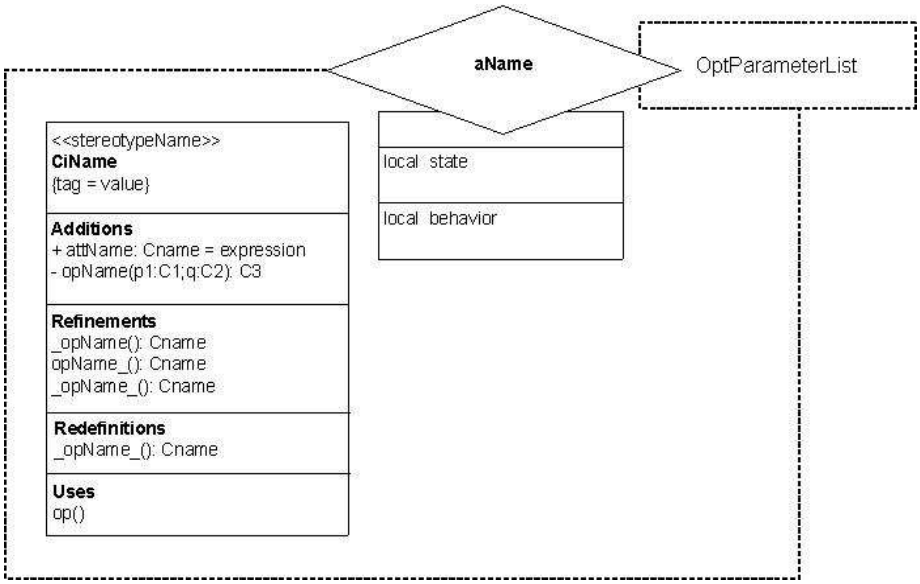

Figura 5.1: Declaração de aspecto completa.

Um aspecto é representado como um retângulo tracejado, com um losango contendo o nome do aspecto, que toca um retângulo usado para descrever suas características locais (atributos e métodos) (Figura 5.1). Um aspecto pode conter vários elementos internos, como interfaces transversais, classes, interfaces e relacionamentos. A notação para descrever interfaces transversais é apresentada em 5.2.2.

Um aspecto é apresentado com um retângulo tracejado pequeno, sobreposto no canto superior direito do retângulo do aspecto. Esse retângulo é chamado de caixa de parâmetros de templates [17] e contém listas de parâmetros formais, sendo uma lista para cada interface transversal de aspecto. O primeiro parâmetro de cada lista é o nome da interface transversal correspondente.

Opções de apresentação

Há dois modos distintos para apresentar um aspecto: *visão total* ou *visão condensada*. A visão total de um aspecto fornece uma descrição de-

talhada de seus elementos, cada interface transversal é apresentada usando um retângulo com compartimentos (Figura 5.1).

A visão condensada de um aspecto omite todas as informações sobre seus elementos internos, exceto os nomes das interfaces transversais; cada interface transversal é exibida como um pequeno círculo com seu nome colocado ao lado (Figura 5.2). O círculo está ligado por uma linha sólida ao losango que representa o aspecto. A visão condensada é o padrão para a apresentação de aspectos em diagramas de classes estendidos (5.2.5.2).

Se o aspecto é parametrizado e for usada uma visão condensada, a(s) lista(s) de parâmetros formais deve(m) ser explicitamente exposta(s).

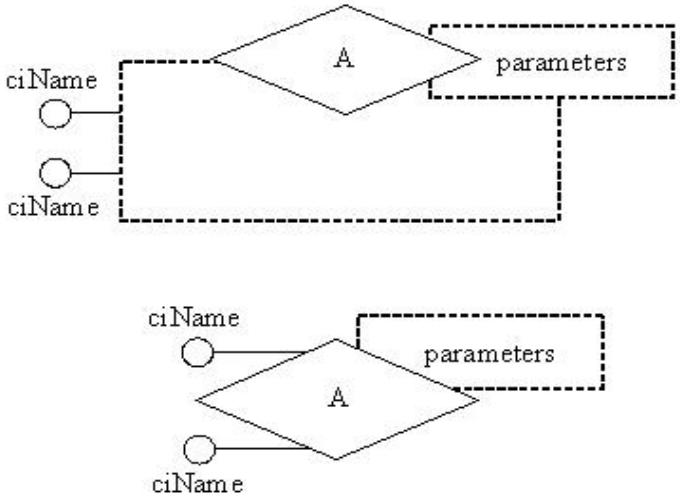

Figura 5.2: Declaração de aspecto condensada.

Diretrizes de estilo

Escolha um nome para o aspecto que denote uma propriedade, uma qualidade, e não uma coisa. Se o aspecto oferece suporte a *crosscutting* horizontal, pode ser usado um nome adequado para denotar uma colaboração.

A apresentação de um aspecto com muitas interfaces transversais pode poluir o diagrama e afetar a compreensibilidade. Mostre aspectos completos quando necessário em diagramas de aspecto e omita detalhes em outros contextos ou referências.

As ferramentas que oferecem suporte à MOA podem associar uma cor ou padrão diferentes a cada aspecto a fim de melhorar a compreensibilidade.

Exemplo

A Figura 5.3 apresenta `Observation`, um aspecto que descreve o padrão de projeto Observer [46]. Esse aspecto especifica o aspecto de AspectJ chamado `SubjectObserverProtocol`, apresentado no Capítulo 4. A Figura 5.4 apresenta outra solução para `Observation` que usa interfaces de UML (a fim de evitar o uso de parâmetros de templates no compartimento Additions).

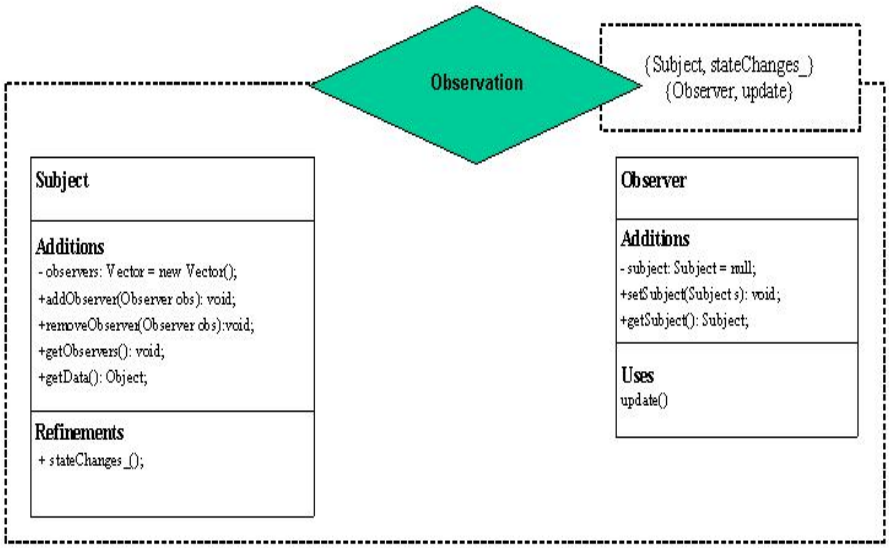

Figura 5.3: O padrão de projeto Observer modelado como um aspecto.

A composição entre o aspecto `Observation` e duas classes base que desempenham os papéis do padrão (a classe `Button` e a classe `ColorLabel`) é explicada na Seção 5.2.4, onde é introduzido o relacionamento *crosscutting*.

5.2.2 Interfaces transversais

As interfaces transversais são conjuntos de características transversais com nome associado, que caracterizam o comportamento *crosscutting* de aspectos. Elas são declaradas dentro de aspectos, em diagramas de aspectos.

Semântica

Uma interface transversal oferece uma descrição parcial do comportamento *crosscutting* do aspecto. O comportamento *crosscutting* de aspectos oferece melhorias às classes e suas instâncias.

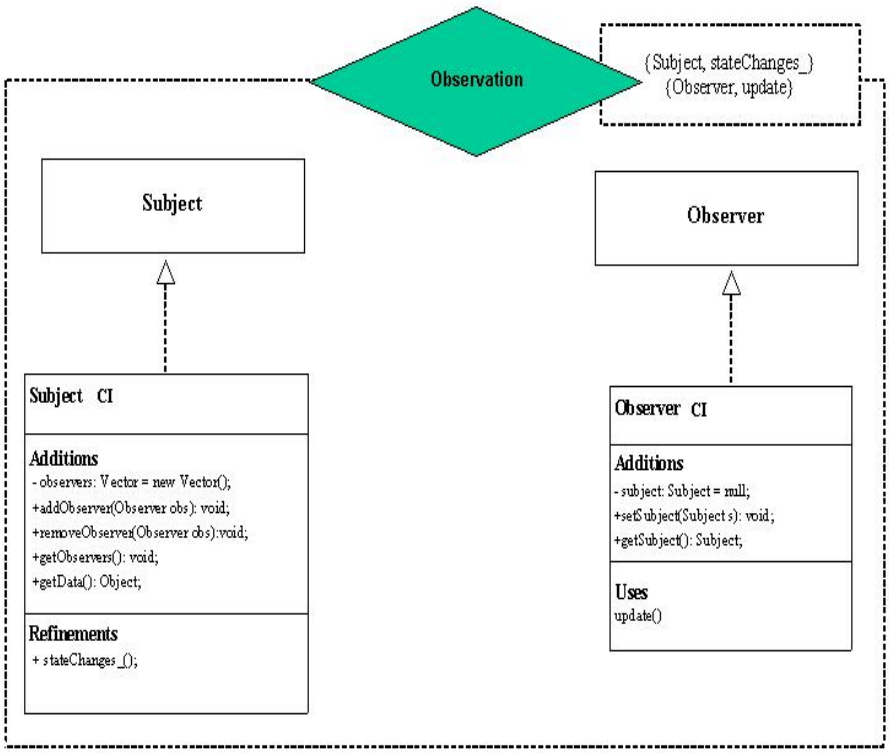

Figura 5.4: Observation com Interfaces.

Uma interface transversal declara:

- um conjunto de características comportamentais e estruturais que o aspecto adiciona a uma classe base;
- um conjunto de características transversais comportamentais que refinam ou redefinem algumas características comportamentais existentes na classe base. O pronome `base` é usado para denotar um *objeto base*;
- um conjunto de assinaturas para características requisitadas (*required features*) usadas pelo aspecto.

As interfaces transversais podem participar de relacionamentos com outros elementos que pertencem ao espaço de nomes do aspecto, incluindo outras interfaces transversais . Mais detalhes sobre a composição envolvendo aspectos e classes podem ser encontrados na Seção 5.2.4.1.

Notação.

Uma interface transversal é representada por um retângulo com linhas sólidas com compartimentos separados por linhas horizontais. O nome que aparece no topo do compartimento representa o nome da interface transversal. O segundo compartimento contém as características transversais que

oferecem suporte a *crosscutting* estático (*additions*), e o terceiro e quarto compartimentos contêm as características que oferecem suporte a *crosscutting* dinâmico (*refinements* e *redefinitions*). Pelo menos um deles não deve estar vazio. Um compartimento opcional pode ser fornecido para definir *placeholders* para operações requisitadas (*uses*).

Os nomes de algumas características transversais comportamentais são decorados com o símbolo _. A notação para descrever características transversais é apresentada na Seção 5.2.3. A Figura 5.5 mostra uma interface transversal com adições, refinamentos, redefinições e usos. A Figura 5.6 mostra uma interface transversal sem redefinições e usos.

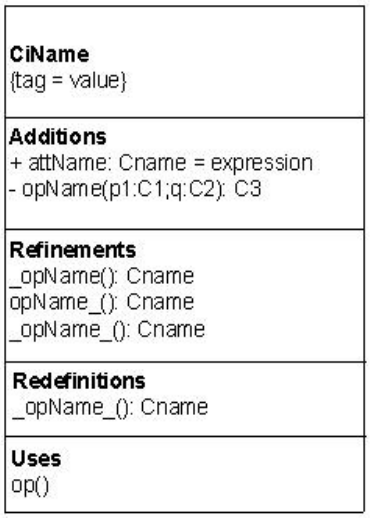

Figura 5.5: Interface transversal com adições, refinamentos, redefinições e usos.

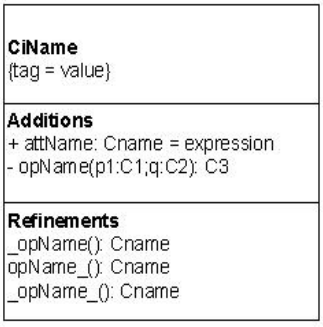

Figura 5.6: Interface transversal com compartimentos omitidos.

Opções de apresentação.

Os nomes de compartimento e os compartimentos podem ser omitidos; as opções de apresentação das classes de UML são adotadas ([134], pp. 3-36).

Na visão condensada de um aspecto, cada interface transversal é exibida como um pequeno círculo com o nome colocado ao lado do símbolo (Figura 5.2).

Diretrizes de estilo.

O nome da interface transversal deve ser considerado um nome de papel. Por exemplo, no aspecto `Observation`, as interfaces transversais são chamadas de `Observer` e `Subject`. Em um aspecto `Tracing`, o nome da interface transversal poderia ser `TracedElement`.

Exemplo.

A Figura 5.3 apresenta um aspecto com duas interfaces transversais. `Observer` é uma interface transversal que modulariza características transversais que afetam objetos arbitrários de forma que esses elementos se tornam observadores. `Observer` declara três adições – o atributo `subject` e duas operações públicas, `setSubject(Subject s)` e `getSubject()` – e um requisito – `update()`. A interface transversal `Subject` modulariza características transversais que afetam objetos arbitrários de forma que esses elementos se tornam sujeitos. `Subject` declara cinco adições e um refinamento – `stateChange_()` – que denota o comportamento a ser executado depois do comportamento base.

5.2.3 Características transversais

Uma *característica transversal* descreve uma propriedade nomeada (atributo ou operação) definida em um aspecto que pode afetar um ou mais elementos base em locais específicos por meio de *crosscutting*.

Semântica

Uma *característica transversal estrutural* é uma especificação de um atributo que será estaticamente introduzido na interface de uma ou mais classes base.

Uma *característica transversal* comportamental é uma especificação de uma fatia de comportamento que será adicionada a uma ou mais classes base, ou para refinar ou redefinir uma operação de uma ou mais classes base.

Uma *característica requisitada comportamental* é uma operação base que será usada dentro do espaço de nomes do aspecto. Às características comportamentais requisitadas, não se aplicam refinamentos nem redefinições.

Algumas características transversais comportamentais podem ser especificadas como elementos parametrizados, isto é, elas podem conter um ou mais parâmetros livres que representam elementos na assinatura da operação. A operação base pode ser explicitamente invocada dentro do corpo da operação do aspecto usando o pronome `base`.

Notação

Características transversais estruturais estão listadas no compartimento `Additions`. A notação segue o padrão do elemento `Attribute` de UML ([134], pp. 3-42).

Características transversais comportamentais são listadas em diferentes compartimentos, dependendo do tipo de melhoria (adição, refinamento, redefinição) ao qual oferecem suporte. Características transversais comportamentais que oferecem suporte à adição são listadas no compartimento `Additions` e não são parametrizadas. A notação segue o padrão do elemento `Operation` de UML ([134], pp. 3-44).

Características transversais comportamentais que oferecem suporte ao refinamento estão listadas no compartimento `Refinements`. Nesse compartimento, os nomes da operação são representados com o símbolo _, em três combinações permitidas: `_op`, `op_` e `_op_`. Esses adornos indicam que a operação de *crosscutting* oferece o comportamento que deve ser combinado antes, depois ou depois/antes do comportamento da operação base. No último caso, a operação base, denotada por `op`, sem os adornos, é invocada explicitamente dentro do corpo de `_op_` usando o pronome *base* `base.op()` (esse uso é análogo àquele do `around advice` com `proceed` encontrado na linguagem AspectJ).

Características transversais comportamentais que oferecem suporte à redefinição estão listadas no compartimento `Redefinitions`. Aqui, o nome da operação também é `_op_`, mas a operação base `op` não deve ser invocada dentro de seu corpo. O comportamento de `op` é redefinido (ou sobreposto) pelo comportamento de `_op_` (esse uso é análogo àquele do `around advice` sem `proceed` encontrado na linguagem AspectJ).

As características requisitadas comportamentais estão listadas no compartimento `Uses`.

A Figura 5.7 mostra uma interface transversal sem as informações de visibilidade.

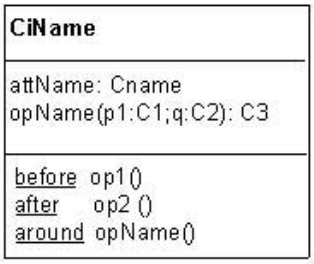

Figura 5.7: Interface transversal : detalhes no nível da análise.

Parâmetros.

Não discutimos aqui a especificação de parâmetros para características transversais comportamentais. Para efeito de simplificação, assumimos que qualquer número de parâmetros do tipo `Object` pode ser fornecido; os parâmetros abstraídos não são descartados, somente omitidos.

Opções de apresentação

Adornos textuais – before, after, around e use – podem substituir o símbolo _. Por exemplo, podemos escrever before `op` em vez de `_op`. A Figura 5.8 mostra uma interface transversal na qual o símbolo _ é substituído por adornos textuais que descrevem o tipo de *crosscutting*.

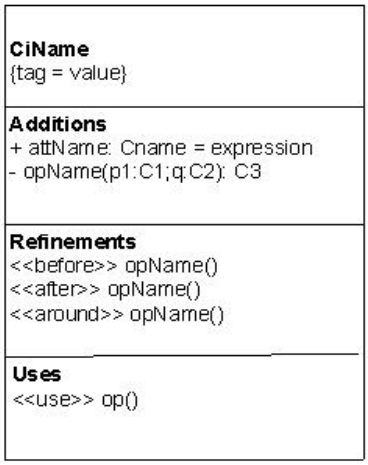

Figura 5.8: Interface transversal com adornos textuais.

Diretrizes de estilo

O nome de uma operação de *crosscutting* que oferece refinamento ou redefinição deve ser genérico a fim de denotar o propósito de uma operação realizada por um possível conjunto de elementos base afetado pelo aspecto. Por exemplo, o padrão Observation, através da interface transversal Subject, espera melhorar um elemento base com comportamento que permite notificar seus observadores depois de ocorrer alguma alteração ou mudança de estado. Portanto, a operação de *crosscutting* é chamada de `stateChange_()`.

Exemplo

A interface transversal Subject apresentada na Figura 5.3 declara cinco adições e um refinamento – `stateChange_()`. A operação de *crosscutting* `stateChange_` especifica o comportamento que melhora o comportamento de uma característica base denotada pelo nome `stateChange`.

5.2.4 Relacionamentos

Na modelagem orientada a objetos, há três tipos básicos de relacionamentos: generalização, associação e dependência. Na modelagem orientada a aspectos, além desses três tipos de relacionamentos, há um novo tipo: *crosscutting*, um relacionamento entre elementos *crosscutting* (por exem-

plo, aspectos) e elementos base (por exemplo, classes). *Crosscutting* é um relacionamento sobrecarregado:

– ele denota uma dependência de um aspecto a um elemento base. O aspecto precisa e usa esse elemento para realizar seu papel *crosscutting*. Se um aspecto afeta vários elementos base, vários relacionamentos de *crosscutting* são definidos, um para cada par aspecto-base;

– ele denota o mecanismo de combinação, pelo qual a estrutura e o comportamento do elemento base são melhorados pela estrutura e pelo comportamento especificados no elemento *crosscutting*.

Há dois tipos (predefinidos) de dependências entre aspectos, o relacionamento *precedence* e o relacionamento *requirement*.

A `aSideML` oferece uma representação gráfica para *crosscutting* e também para relacionamentos entre aspectos.

5.2.4.1 Crosscutting

Semântica

Crosscutting é um relacionamento entre um elemento *crosscutting* (por exemplo, um aspecto) e um elemento base. Ele especifica que o aspecto deve atravessar os limites do elemento base em pontos de combinação bem definidos e modificar incrementalmente a base nesses pontos.

Os modelos estruturais oferecem suporte a *crosscutting estrutural estático* (3.1.3.2), ou seja, *crosscutting* que usa pontos de combinação estáticos a fim de afetar a estrutura dos elementos base. A Tabela 5.2 apresenta os pontos de combinação estáticos considerados por `aSideML` em relação aos elementos estruturais do Modelo de Objetos de UML. O *crosscutting comportamental dinâmico* é descrito pelos modelos comportamentais (consulte 5.3).

Elemento base	Ponto de combinação estático	Contexto Léxico
classe	lista de features, ancestrais	nome, atributos, operações, ancestrais, ...
interface	lista de features, ancestrais	nome, operações, ancestrais, ...

Tabela 5.2: Elementos estruturais e pontos de combinação estáticos.

Em `aSideML`, o relacionamento de *crosscutting* classifica um relacionamento entre um aspecto parametrizado e um elemento base; ele também realiza uma associação que define as operações e os elementos base que substituem os parâmetros de templates do aspecto.

Os parâmetros de templates são agrupados por interface transversal. Para cada interface transversal especificada no aspecto, um conjunto de casamento de templates (*template matches*) que definem substituições para essa interface transversal são colocados entre brackets <>. O conjunto de substituições para os parâmetros de templates do aspecto usado no relacionamento *crosscutting* é descrito na Tabela 5.3. Esse conjunto pode ser estendido para oferecer suporte a *wildcards* e a uma sublinguagem de quantificação mais expressiva.

Template Match	Descrição
$< formalName \rightarrow actualNames >$	Cada template match é exibido como $formalName \rightarrow actualNames$, onde $formalName$ é um parâmetro de template e $actualNames$ pode ser uma lista separada por vírgulas de operações de classe existentes, entre brackets <>.
$< formalName \rightarrow all >$	Quando $formalName$ denota o nome da interface transversal, *all* denota que todas as classes visíveis são substituições para o parâmetro de template. Quando $formalName$ denota uma operação de template, *all* denota que todas as operações dentro de cada classe que está sendo substituída são substituições para a operação de template.

Tabela 5.3: Substituição para parâmetros de template do aspecto.

A restrição {`xor`} pode ser aplicada a um conjunto de relacionamentos *crosscutting* que compartilham uma conexão a um elemento base, especificando que, nesse conjunto, exatamente um aspecto afetará tal elemento base.

Notação

Um relacionamento de *crosscutting* é representado como uma seta tracejada com a parte final no elemento *crosscutting* e a ponta no elemento base, e a palavra-chave `<<crosscut>>`. Pode incluir uma lista de associações. A Figura 5.9 apresenta a representação gráfica do relacionamento *crosscutting* .

Cada associação relaciona um parâmetro de template definido na interface transversal do aspecto (nome da interface ou nome da operação)

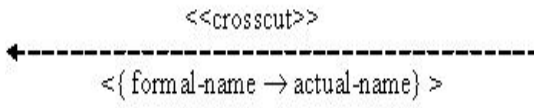

Figura 5.9: Crosscutting.

a um nome (ou seqüência de nomes) definido na interface de uma classe (nome de classe ou um nome de método). Essa informação *crosscutting* é exibida como uma lista de correspondências de parâmetros de templates separadas por vírgulas, $< \{templateMatch\}* >$.

A restrição `xor` é mostrada como uma string de texto entre chaves ({`xor`}).

Opções de apresentação

Diferentes relacionamentos *crosscutting* partindo do mesmo aspecto podem ser associados no estilo um-para-vários (*multiple-target style*). Cada caminho oferece uma lista de associações. A Figura 5.10 apresenta o relacionamento de *crosscutting* no estilo um-para-vários.

Uma associação como $< baseOp \rightarrow all >$ especifica que o aspecto afeta todas as operações definidas nas classes base.

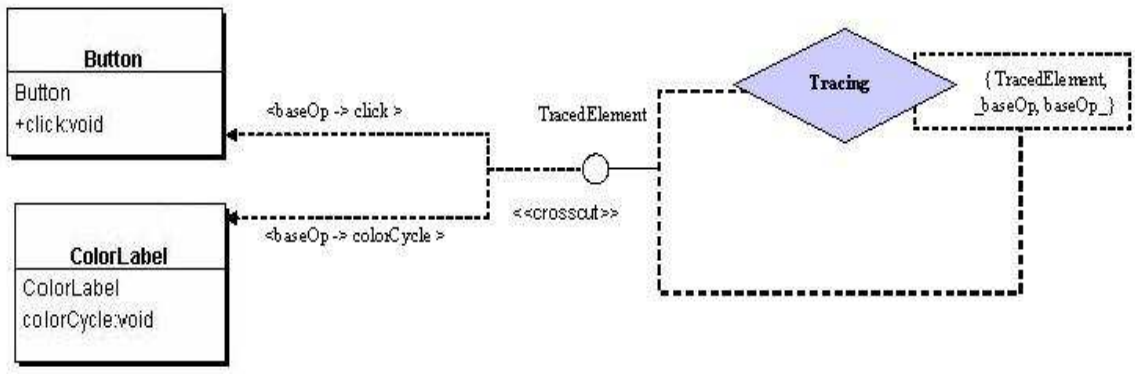

Figura 5.10: Crosscutting no estilo um-para-vários.

Exemplo

A Figura 5.14 apresenta dois relacionamentos de *crosscutting* que associam o aspecto `Observation` a `Button` (ligando `stateChange` a `click`) e `ColorLabel` (ligando `update` a `colorCycle`).

`Observation` afeta `Button` por meio de *crosscutting*; a estrutura de instâncias de `Button` inclui novos atributos e operações listados no compartimento Additions de `Observation`; seu comportamento é melhorado por `Observation` no ponto de combinação definido (`click`). `Observation` também afeta a estrutura de instâncias de `ColorLabel` e usa o método `colorCycle`.

5.2.4.2 Precedência

Semântica

O relacionamento `precedence` fornece uma conexão explícita entre dois aspectos que afetam o mesmo elemento base e oferece o mesmo tipo de melhoria em um ponto de combinação compartilhado. A precedência entre dois aspectos pode ser diferente quando são compostos com diferentes elementos base.

Esse relacionamento define que um aspecto (o cliente) precede outro aspecto (o fornecedor). Isso significa que o comportamento do aspecto cliente tem precedência em relação ao comportamento do aspecto fornecedor no tipo de *crosscutting* que esperam realizar.

Notação

O relacionamento `precedence` é apresentado como uma Dependência (de UML [134], pp.3-90): uma seta tracejada entre dois elementos do modelo. A seta é rotulada com o estereótipo `precede`. O aspecto na parte final da seta (o cliente) precede o aspecto na cabeça da seta (o fornecedor).

Exemplo

A Figura 5.11 apresenta `Tracing`, um aspecto que rastreia chamadas a `click` e `colorLabel`. Um relacionamento `precedence` é representado do aspecto `Observation` ao aspecto `Tracing`. Isso significa que o comportamento de notificação ocorre antes do comportamento de rastreamento.

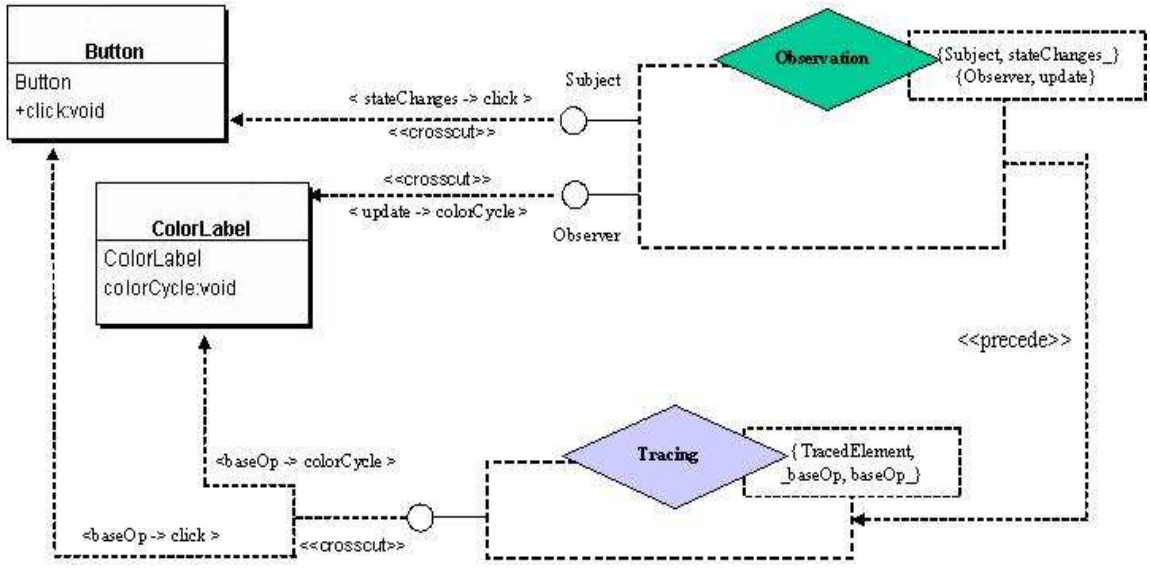

Figura 5.11: O relacionamento `precedence`.

5.2.4.3 Requisito

Semantics

O relacionamento `requirement` oferece uma conexão explícita entre um aspecto (o cliente) que requer a presença de outro aspecto (o fornecedor) para sua implementação ou funcionamento correto. Por exemplo, um aspecto pode requerer a presença de outro porque ele afeta uma operação adicionada a um elemento base por aquele aspecto requisitado. Em geral, um relacionamento `Requirement` também tem a semântica do relacionamento `Precedence` implícita.

Notação

O relacionamento `require` é apresentado como uma Dependência (de UML [134], pp.3-90): uma seta tracejada entre dois elementos do modelo. A seta é rotulada com o estereótipo `require`. O aspecto na parte final da seta (o cliente) requer a presença do aspecto na cabeça da seta (o fornecedor).

Exemplo

A Figura 5.12 apresenta dois relacionamentos `Requirement` do aspecto `Autonomy` e do aspecto `Adaptation` ao aspecto `Interaction`. Esse exemplo

é melhor explicado no Capítulo 7, no qual é apresentado o sistema multiagentes Portalware.

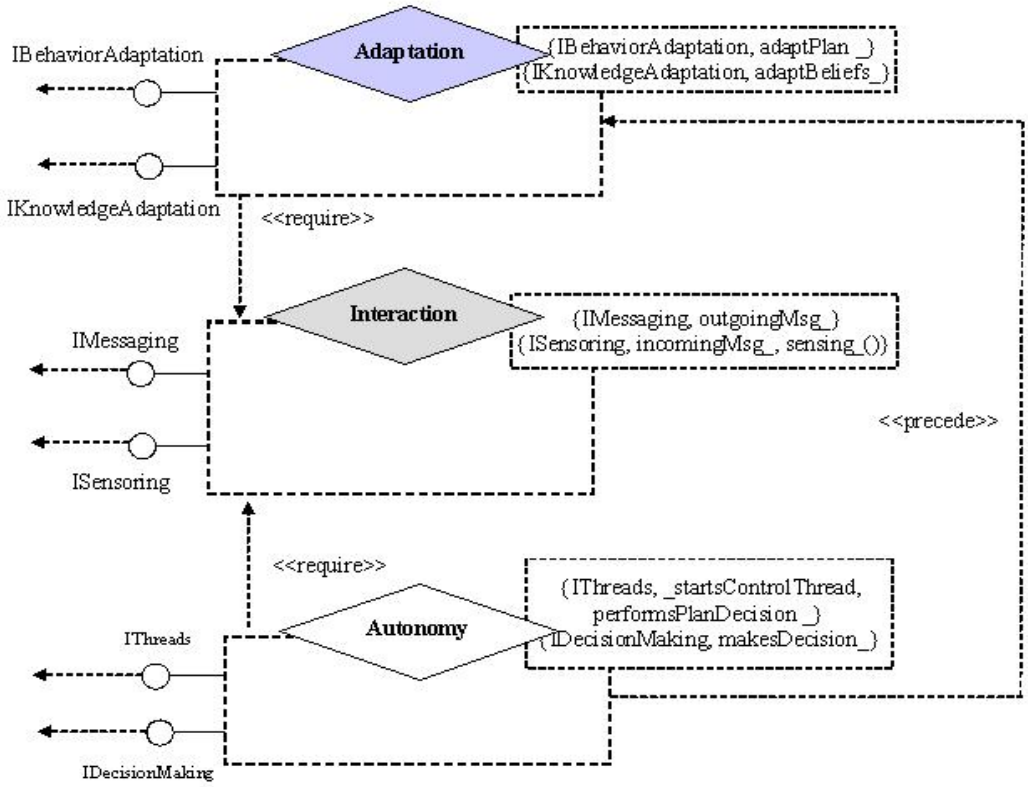

Figura 5.12: Relacionamento `requirement`.

5.2.5 Diagramas

5.2.5.1 Diagramas de Aspecto

Um diagrama de aspecto é uma visão gráfica do modelo de aspectos estrutural estático, envolvendo elementos do núcleo ou modelo principal. Os diagramas de aspecto individuais não representam divisões no modelo subjacente.

Um diagrama de aspecto fornece uma descrição completa de um aspecto. A descrição incorpora as interfaces transversais , características locais e relacionamentos com outros aspectos. Cada característica transversal comportamental pode ser transformada em um diagrama de colaboração aspectual.

Notação

Um diagrama de aspecto é uma coleção de elementos de modelagem estrutural, como aspectos e seus relacionamento, conectados como um grafo.

Exemplo

A Figura 5.3 mostra um diagrama de aspecto que apresenta o aspecto `Observation`. A Figura 5.13 apresenta um diagrama de aspecto que mostra o aspecto `Timing` do sistema Telecom (c.f. [76]).

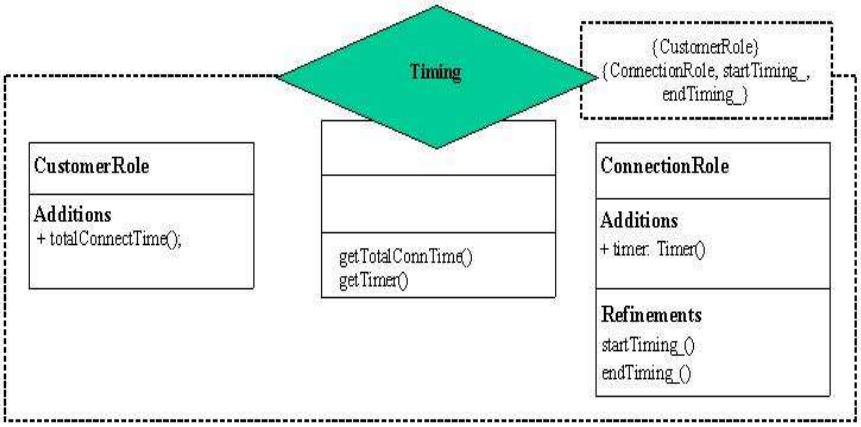

Figura 5.13: Diagrama de aspecto e o aspecto `Timing`.

5.2.5.2 Diagramas de classes estendidos

O diagrama de classes estendido é um grafo de aspectos e classes conectado por seus diferentes relacionamentos estáticos.

Um diagrama de classes estendido oferece uma apresentação gráfica da visão estática de projeto de um sistema em que as classes e os aspectos residem como cidadãos de primeira classe. Cada aspecto pode ser apresentado em uma visão detalhada separada através de um diagrama de aspecto.

Notação

Um diagrama de classes estendido é uma coleção de elementos de modelagem estrutural, como aspectos, classes, interfaces e seus relacionamentos, conectados como um grafo entre si. Os aspectos são apresentados usando a visão condensada.

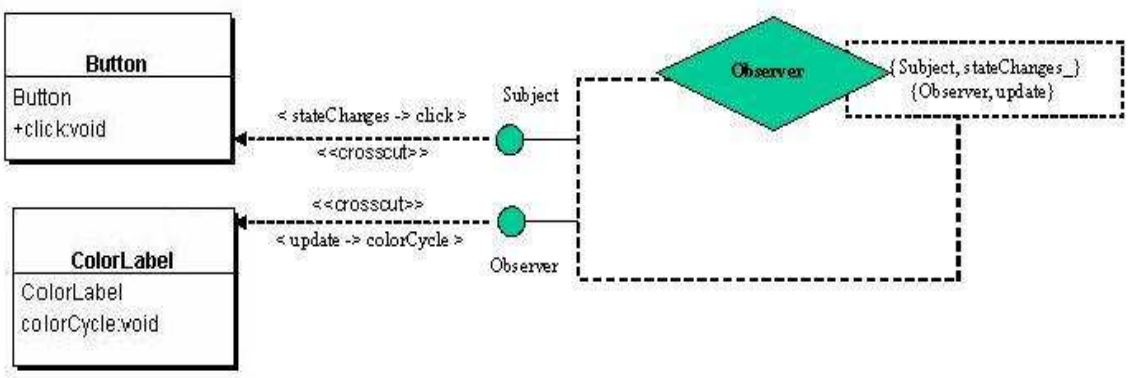

Figura 5.14: Diagrama de classes estendido.

Exemplo

A Figura 5.14 apresenta um diagrama de classes estendido.

5.2.5.3 Perspectiva centrada em aspectos

Uma perspectiva centrada em aspectos oferece uma apresentação gráfica da visão estática de projeto de um sistema em que o foco principal é a *descrição de um aspecto e seus relacionamentos com os elementos base.*

Exemplo

A Figura 5.10 apresenta uma perspectiva centrada em aspectos para o aspecto `Tracing`.

5.2.5.4 Perspectiva centrada em base

Uma perspectiva centrada em base oferece uma apresentação gráfica da visão estática de projeto de um sistema em que o foco principal é a *descrição de um elemento base e os aspectos que o afetam.* Os diagramas centrados em base apresentam informações explícitas sobre a dependência entre elementos de *crosscutting* que afetam um elemento base. Esses diagramas podem ser automaticamente gerados por ferramentas de projeto.

Exemplo

A Figura 5.15 apresenta uma perspectiva centrada em base para a classe `Button` a partir do diagrama de classes estendido apresentado na Figura 5.11.

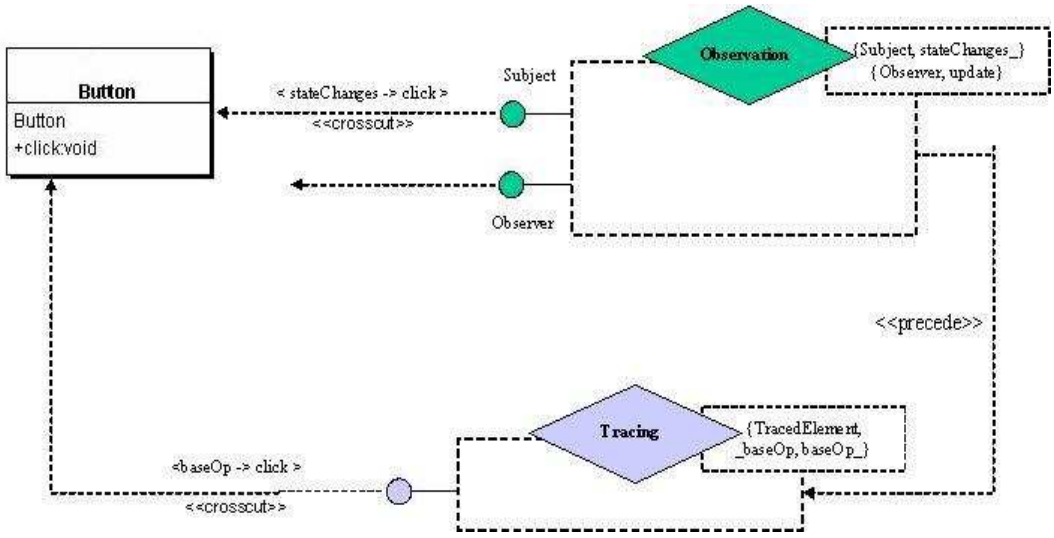

Figura 5.15: Perspectiva centrada em base.

5.2.5.5 Perspectiva de pontos de combinação

Uma perspectiva de ponto de combinação oferece uma apresentação gráfica para uma coleção de elementos base em que os pontos de combinação são apresentados explicitamente.

Em diagramas de classes estendidos, o relacionamento de *crosscutting* é usado a fim de associar elementos *crosscutting* a elementos base, indicando as operações base afetadas explicitamente em uma lista de associações. No entanto, quando um aspecto afeta um grande número de classes e operações, o diagrama pode se tornar um conjunto confuso de setas entrelaçadas, de uma origem (aspecto) até vários destinos (classes); a falta de compreensibilidade pode se tornar um problema.

Na perspectiva de ponto de combinação, os relacionamentos de *crosscutting* podem ser eliminados na origem (Figura 5.16) e no destino (Figura 5.17). No destino (classe base), a cor ou o nome do aspecto devem ser mostrados, e os pontos afetados (nomes de operação) devem ser enfatizados.

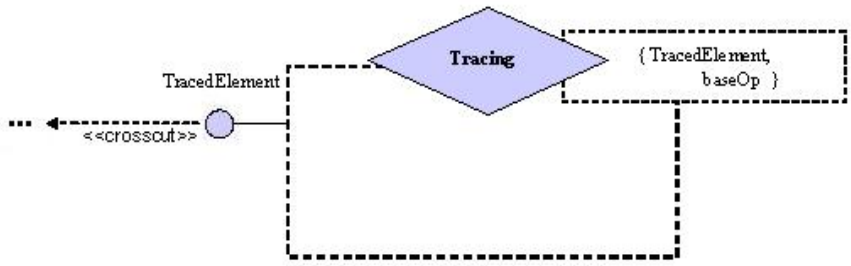

Figura 5.16: Perspectiva de ponto de combinação: *crosscutting* omitido na fonte.

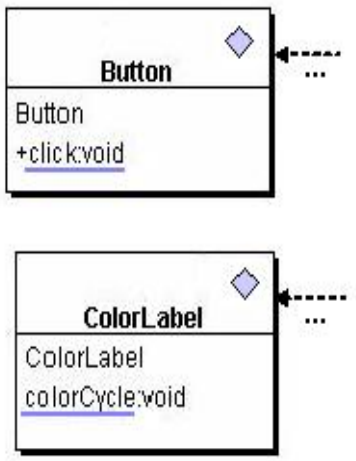

Figura 5.17: Perspectiva de ponto de combinação: *crosscutting* omitido no destino.

5.3 Modelagem Comportamental

Em aSideML, a modelagem comportamental oferece a visão de interação de um sistema na presença de aspectos. Os principais constituintes dos modelos comportamentais são objetos, instâncias de aspectos e suas colaborações e interações.

Os modelos comportamentais caracterizam o comportamento de aspectos em termos da forma como interagem com objetos. Há duas facetas na modelagem comportamental:

– a descrição do comportamento que corresponde ao *corpo de uma operação de crosscutting* e a forma como a instância de aspecto interage com outras instâncias (não as instâncias base afetadas) a fim de fornecer outros comportamentos. Nesse caso, basta um diagrama de seqüência comum, no qual a instância mais à esquerda é uma instância de aspecto (consulte a Figura 5.18);

– a descrição dos protocolos entre instâncias de aspecto e objetos base. Nesse caso, as colaborações aspectuais e as interações aspectuais são fornecidas. Como esses elementos dependem de uma *estratégia de combinação* a fim de descrever os efeitos de *crosscutting* em modelos comportamentais existentes, eles também são considerados modelos de processo de combinação (5.4).

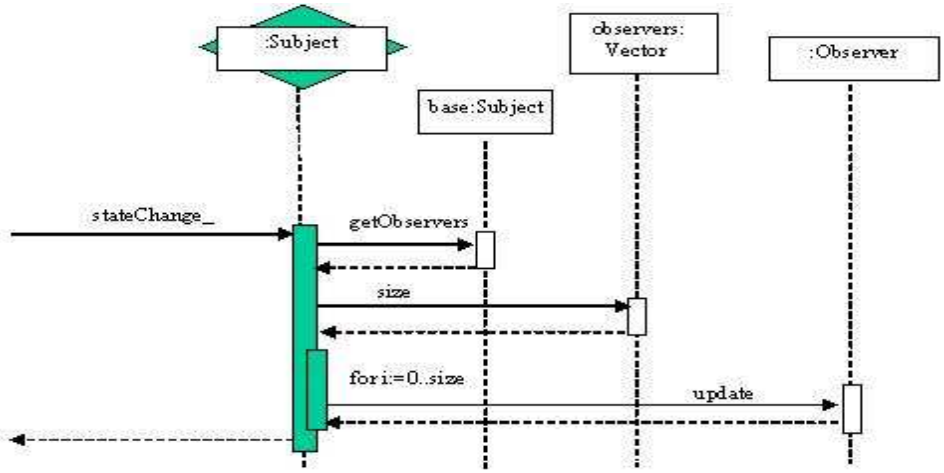

Figura 5.18: Diagrama de sequência para `stateChange`.

5.3.1
Instâncias de aspecto

Semântica

Uma *instância de aspecto* tem identidade e pode ter estado. As instâncias de aspecto não podem ser explicitamente criadas ou destruídas; só podem ser tratadas dentro de aspectos. Elas são elementos de modelagem; podem ser objetos comuns no nível de implementação.

Notação

A notação da instância de aspecto é derivada da notação de objeto, sendo representada por um losango acima do mesmo (Figura 5.19).

As instâncias de aspecto são instâncias anônimas, por definição. Portanto, o nome da instância de aspecto é sempre omitido, mas o nome do aspecto que dá origem está sempre presente e sublinhado, usando a sintaxe `:aspectName`.

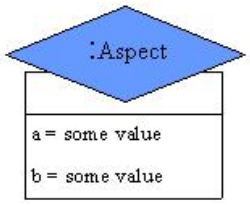

Figura 5.19: Instância de aspecto.

Uma notação semelhante também representa o papel exercido por um aspecto dentro de uma colaboração aspectual porque papéis têm características semelhantes a instâncias.

Opções de apresentação

O compartimento de valor de atributos como um todo pode ser omitido. Os atributos cujos valores não são de interesse podem ser omitidos.

Diretrizes de estilo

As ferramentas que oferecem suporte à MOA podem associar uma cor diferente a cada aspecto e suas instâncias. A apresentação de instâncias de aspecto serve para expor os valores de seus atributos locais.

5.3.2 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.

Esse tipo de colaboração modela a realização de uma operação de *crosscutting* definida em um aspecto; um conjunto de colaborações aspectuais modela a realização do comportamento *crosscutting* dinâmico do aspecto (consulte 5.2.4.1).

Semântica

A colaboração aspectual define um conjunto de papéis e um conjunto de interações que descreve a estrutura e a comunicação entre a instância de aspecto e objetos que exercem os papéis definidos.

Uma colaboração aspectual possui uma parte estática e dinâmica. A estática descreve os papéis que objetos e instâncias de aspecto exercem. Já a dinâmica é apresentada por uma interação aspectual (5.3.3), que mostra os fluxos de mensagens ao longo do tempo para realizar o comportamento *crosscutting* de acordo com os papéis.

Uma colaboração aspectual é uma colaboração parametrizada com pelo menos três papéis classificadores definidos em sua parte estática. Há dois papéis predefinidos para objetos: `Sender` e `Receiver`, denotando o chamador e o chamado da operação base sendo afetada pelo aspecto.

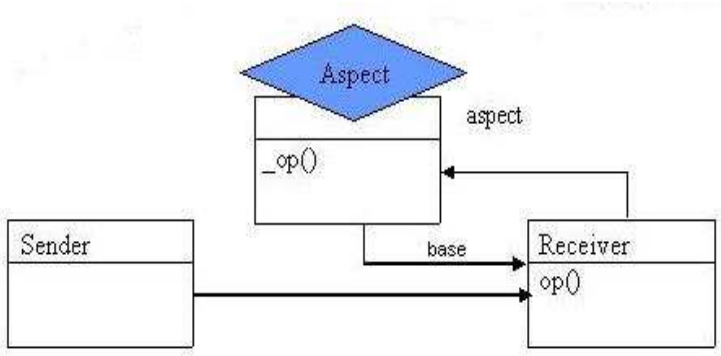

Figura 5.20: Colaboração aspectual: parte estática.

Há três *papéis de associação*: um do aspecto para um objeto, que denota a base; outro do emissor para o receptor; e, finalmente, um terceiro papel de associação do objeto base para a instância de aspecto. Esse terceiro tipo de papel de associação é *crosscutting*; ele afeta a estrutura da instância base ao introduzir uma nova associação, do objeto para a instância de aspecto. Finalmente, o comportamento *crosscutting* pode afetar objetos exercendo os papéis de Sender (melhoria do lado da chamada de método) ou de Receiver (melhoria do lado da execução do método).

Notação

A Figura 5.21 apresenta a representação gráfica de uma colaboração aspectual. A parte estática da colaboração é apresentada na parte superior, representando três papéis e suas associações. A associação base indica que

o comportamento *crosscutting* estende o comportamento dos objetos que exercem o papel `Receiver`. A notação usada para representar os papéis definidos por uma colaboração aspectual é semelhante à notação usada para apresentar os papéis de colaboração (de UML, pp.3-126).

Já a parte dinâmica oferece uma interação aspectual que descreve o comportamento estendido de algum objeto de um tipo denotado por `Receiver` que possui um método `op` chamado por algum objeto de um tipo denotado por `Caller`. A chamada a `_op` deve ser inserida no início da ativação da operação `op`. O pequeno losango denota o ponto de combinação.

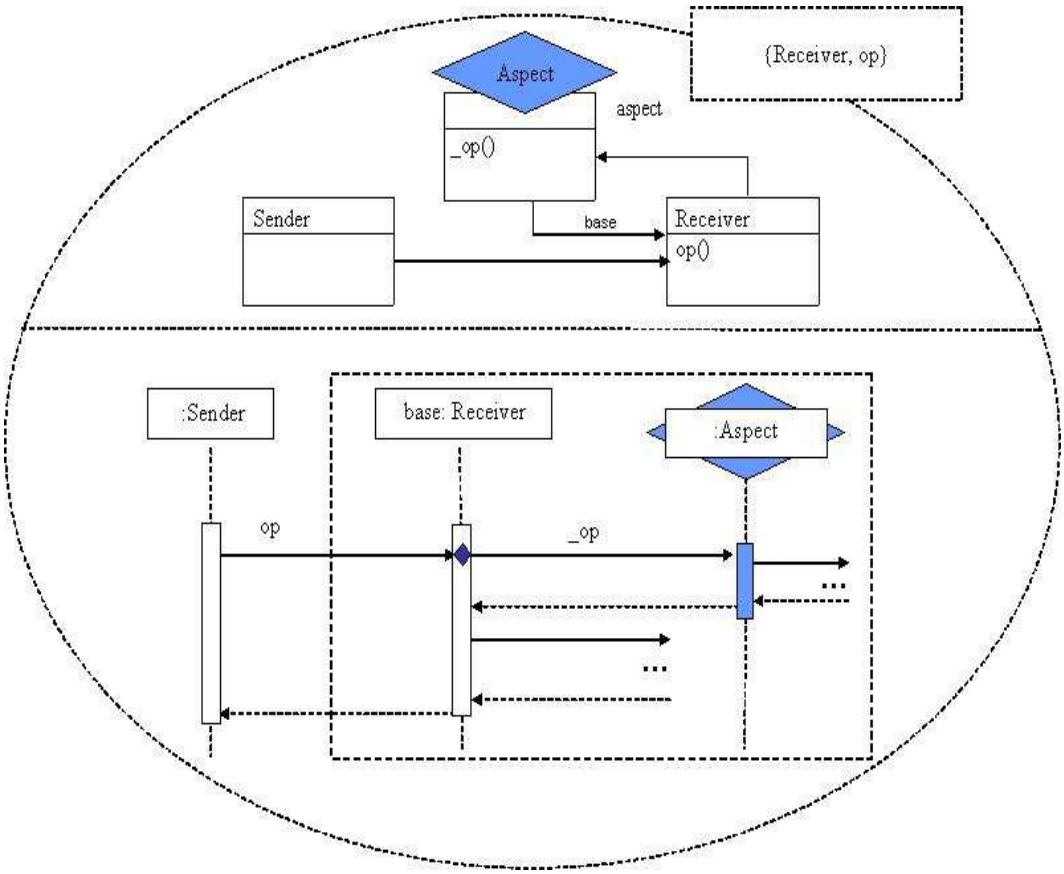

Figura 5.21: Colaboração aspectual: Refine-Before em `Receiver`.

Exemplo

A Figura 5.22 apresenta uma colaboração aspectual para a operação `stateChange_`. Ela descreve o comportamento estendido de algum objeto de um tipo denotado por `Sender` que chama a operação `stateChange`.

As melhorias incorporam a inserção de um *link* chamado `aspect` a partir do objeto base (denotado por Sender) até a instância de aspecto e a inserção de uma chamada na operação de *crosscutting* chamada `stateChange_`

na sombra estática associada ao ponto de combinação dinâmico. Nesse caso, a chamada a `stateChange_` é inserida depois da chamada a `stateChange`.

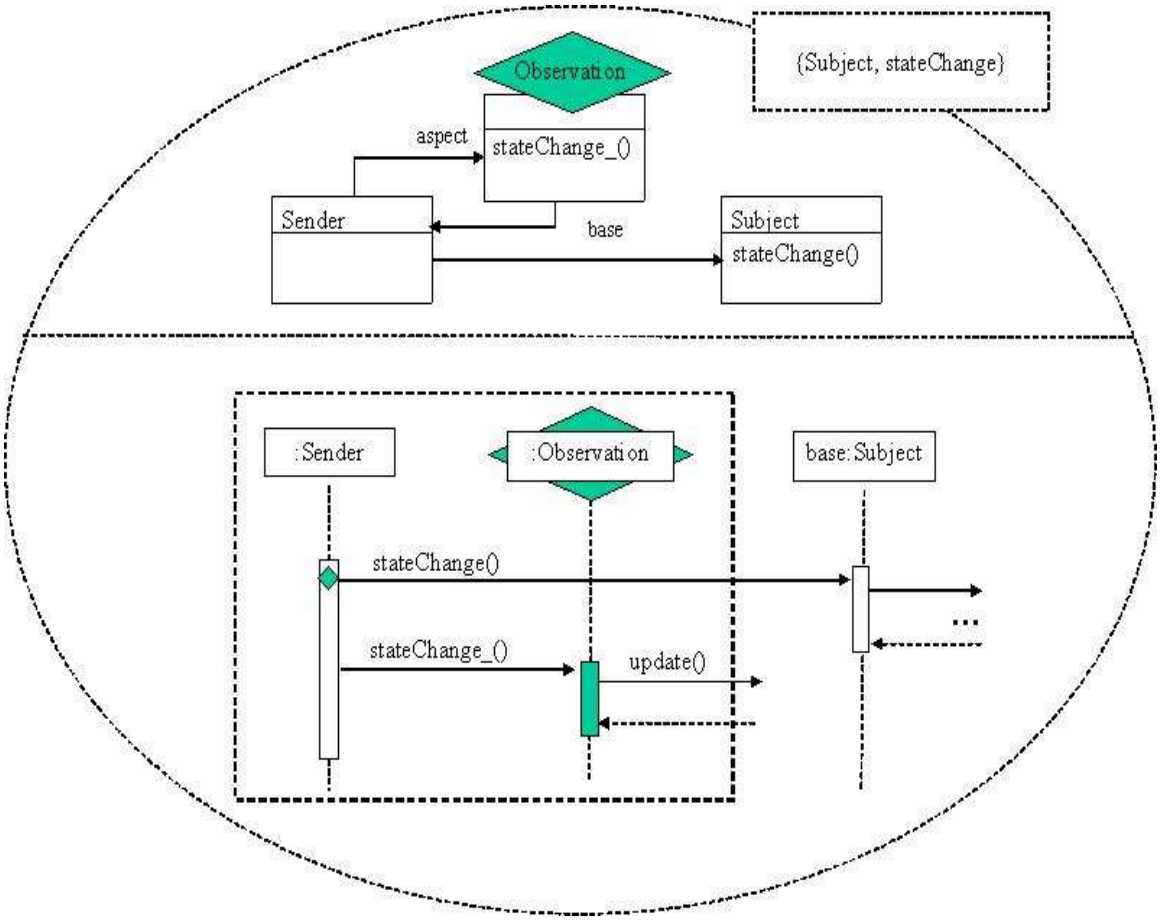

Figura 5.22: Colaboração aspectual para `stateChange_`.

Técnicas de modelagem

Na modelagem estrutural, a especificação de uma característica transversal comportamental define o tipo de *crosscutting* (`before`, `after` ou `around`) e o tipo de extensão (`refine` ou `redefine`) suportados. Essas informações são apresentadas por meio de interfaces transversais.

Na modelagem comportamental, a especificação de uma característica transversal comportamental usando colaboração aspectual define:

- o tipo de colaboração aspectual usado, com base no tipo de *crosscutting* (`before`, `after` ou `around`) e o tipo de extensão (`refine` ou `redefine`) associados à operação de *crosscutting* ;
- o tipo de ponto de combinação dinâmico, o local de melhoria na instância base (Sender ou Receiver), adornados com um pequeno losango na ativação correspondente;

– o contexto exposto (parâmetros, valor de retorno e objetos) no ponto de combinação.

  Apesar de não serem tratados aqui, os parâmetros e outras informações de contexto usados por aspectos podem ser explicitamente apresentados em diagramas comportamentais.

O projetista não deve desenhar os elementos internos de diagramas de colaboração aspectual. O suporte da ferramenta é necessário para fornecer templates de figuras, com papéis predefinidos e as informações já disponíveis a partir dos diagramas de aspecto estáticos.

5.3.3 Interações aspectuais

Uma *interação aspectual* é uma especificação comportamental que incorpora uma seqüência de comunicações trocadas entre um conjunto de objetos e uma instância de aspecto dentro de uma colaboração aspectual a fim de realizar a implementação de uma operação de *crosscutting* .

Semântica

Uma interação aspectual especifica a parte dinâmica de uma colaboração aspectual (5.3.2). É uma `Interaction` (de UML) com restrições adicionais. As interações aspectuais definem a seqüência de comunicações trocadas a fim de realizar o comportamento *crosscutting* de acordo com os papéis de colaboração.

Notação

A Figura 5.21 apresenta a representação gráfica de uma colaboração aspectual; a interação aspectual é mostrada na parte inferior da figura.

As interações aspectuais são mostradas como diagramas de seqüência (de UML [134], pp.3-102) com alguns adornos. Um retângulo tracejado cerca o papel `Aspect` e o papel `Receiver` a fim de denotar que o comportamento *crosscutting* afeta os objetos que exercem o papel `Receiver`. Um ponto de combinação dinâmico está de certa forma sombreado na linha da vida do objeto base. Um pequeno losango denota o ponto de combinação; nesse caso, um ponto no começo da ativação do objeto base.

Portanto, a interação aspectual especifica que o comportamento *crosscutting* refina o comportamento de objeto Receiver e executa antes de o método denotado por `op` ser executado.

Exemplo

A interação aspectual para a operação `stateChange_` é apresentada na parte inferior da Figura 5.22. Ela especifica que o comportamento *crosscutting* refina o comportamento de objeto Sender e executa após o método denotado por `stateChange` ser executado.

5.3.4 Padrões de interação aspectual

Um aspecto possui um conjunto de colaborações aspectuais associadas, uma para cada operação de *crosscutting* comportamental. Cada colaboração aspectual oferece o contexto para uma interação aspectual.

Uma interação aspectual é uma interação parametrizada. Os parâmetros de templates denotam nomes de operação e de classificadores. Em cada uso dessa interação, os nomes de operação e classificadores reais são substituídos pelos valores dos parâmetros associados.

Considere a interação apresentada na Figura 5.23. Ela mostra um objeto que chama `click` e uma instância de `Button`. O adorno descreve o ponto de combinação (chamada de método).

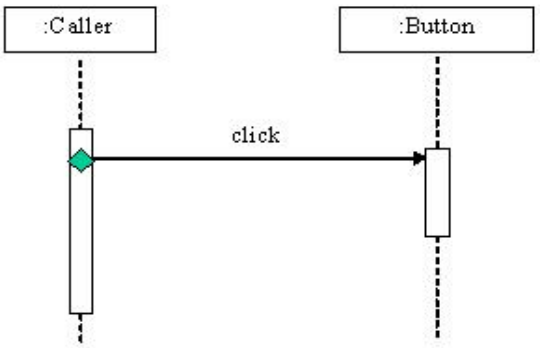

Figura 5.23: Interação adornada com o símbolo de ponto de combinação.

Quando os parâmetros do aspecto Observation apresentado na Figura 5.3 são associados a `Button, click, ColorLabel` e `colorCycle`, a interação aspectual apresentada na Figura 5.22 especifica que a operação `stateChange_`, definida no aspecto Observation, melhora o comportamento

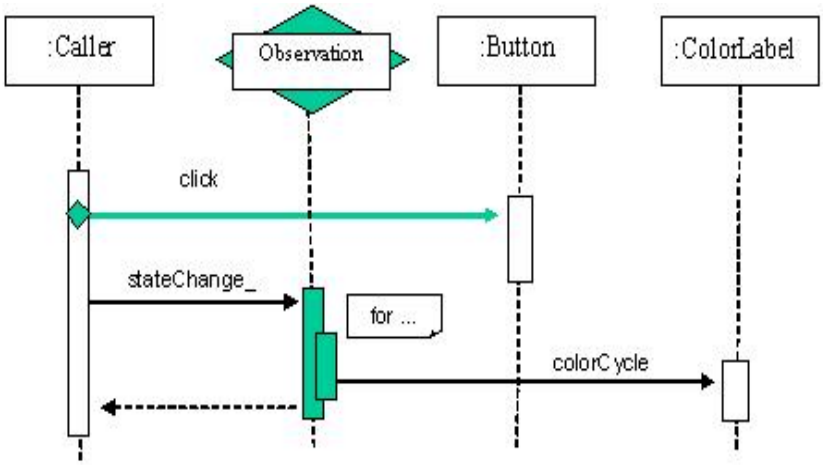

Figura 5.24: Interação aspectual.

de classes arbitrárias que chamam o método `click` (de `Button`). A Figura 5.24 apresenta a interação resultante.

Uma interação aspectual oferece suporte a outros níveis de customização. Ela pode ser caracterizada por:

1. o modelo de combinação (modificação local ou migração cliente);
2. o tipo de extensão (refine ou redefine);
3. o tipo de *crosscutting* (before, after ou around);
4. o tipo de ponto de combinação (chamada, execução, criação).

O modelo de processo de combinação (1) é uma propriedade no nível do modelo; ele descreve o tipo de estratégia de combinação adotada em todo o modelo (B.6) e deve ser garantido por uma ferramenta CASE. As outras três dimensões (2, 3 e 4) são propriedades no nível de operação e caracterizam uma característica transversal comportamental. O tipo de extensão e de *crosscutting* são derivados a partir da própria característica transversal (visão estática); a quarta propriedade deve ser explicitamente fornecida.

Um *padrão de interação aspectual* é uma interação aspectual caracterizada por essas propriedades. Há quatro tipos de combinações válidas do tipo de extensão e *crosscutting*: *refine-before*, *refine-after*, *refine-around* e *redefine*. Esses tipos de combinação correspondem às operações expressas no compartimento `Refinements` e no compartimento `Redefinitions` (consulte 5.2.3). Além disso, cada um desses padrões pode ser refinado para descrever os tipos de pontos de combinação.

Há quatro tipos de padrões de interação aspectual, uma para cada combinação válida do tipo de extensão e *crosscutting*: *refine-before*, *refine-after*, *refine-around* e *redefine*. Esses padrões podem ser refinados para descrever os tipos de pontos de combinação: *refine-before-call*, *refine-before-execution*, *refine-before-creation*, *refine-after-call*, etc.

As Figuras 5.25, 5.26, 5.27 e 5.28 apresentam quatro tipos de interações aspectuais para execução de método; todos oferecem suporte à estratégia de combinação de modificação local.

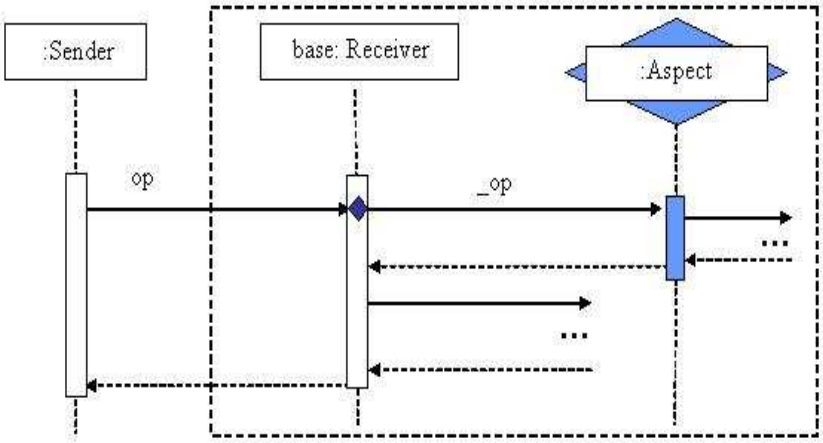

Figura 5.25: Padrão de interação Refine-Before-Execution.

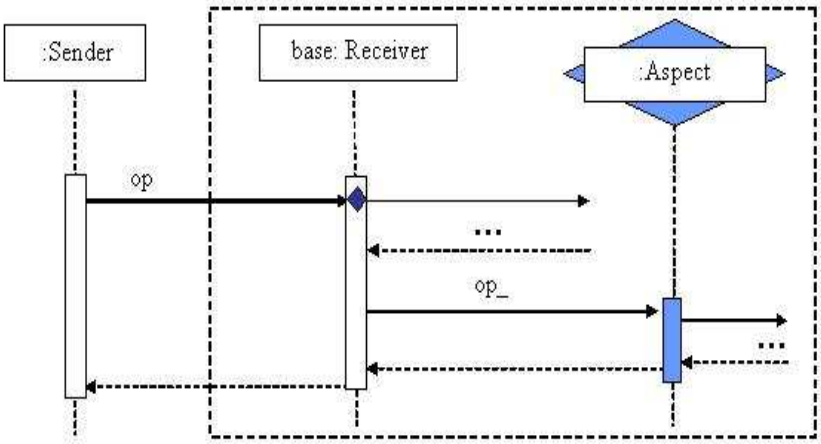

Figura 5.26: Padrão de interação Refine-After-Execution.

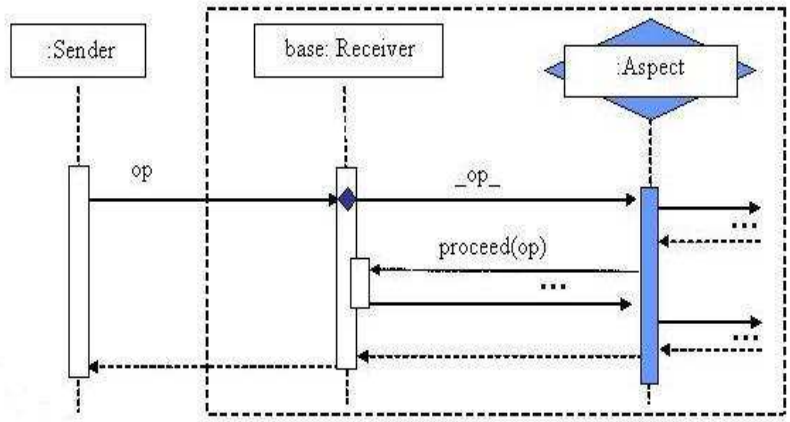

Figura 5.27: Padrão de interação Refine-Around-Execution.

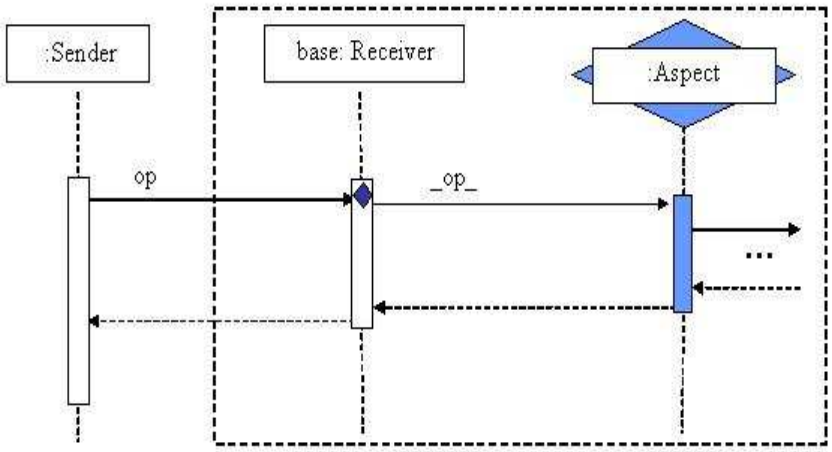

Figura 5.28: Padrão de interação Redefine-Execution.

5.3.5
Informações de contexto e pontos de combinação dinâmicos

Uma colaboração aspectual é acoplada a uma característica transversal comportamental a fim de descrever o *contexto* no qual seu comportamento ocorre, ou seja, quais os papéis que as instâncias exercem para realizar o comportamento especificado pela operação de *crosscutting* . Como uma colaboração, ela também pode incluir os *argumentos* da operação, assim como associações comuns acopladas ao elemento classificador que possui a operação.

Contexto de Crosscutting

A aSideML oferece a notação para a identificação de pontos de combinação dinâmicos nos modelos comportamentais.

Os pontos de combinação dinâmicos são chamada de métodos, execução de métodos e criação de objeto. Esses elementos podem ser identificados e apresentados em diagramas de seqüência aspectuais usando um pequeno losango.

O contexto de *crosscutting* também pode ser exposto nos diagramas de seqüência aspectuais. Outros parâmetros de templates podem ser usados para representar os parâmetros formais de chamadas de métodos, a fim de permitir o tratamento de seus valores no contexto da interação aspectual.

5.3.6 Diagramas

As colaborações aspectuais e as interações aspectuais são apresentadas em *diagramas de colaboração aspectuais* e *diagramas de interação aspectuais*. A Figura 5.22 apresenta uma colaboração aspectual para a operação stateChange_ no diagrama de colaboração aspectual. As Figuras 5.25, 5.26, 5.27 e 5.28 são exemplos de diagramas de interação aspectuais.

5.4 Modelagem Composicional

Na linguagem aSideML, os modelos de processo de combinação (ou composicionais) descrevem as visões estáticas e de interação de um sistema depois da combinação de modelos de objetos e modelos de aspectos.

Os principais constituintes visuais de modelos de processo de combinação são *classes combinadas*, *colaborações combinadas* e *interações combinadas*. Esses elementos são apresentados em *diagramas de processo de combinação*. Diagramas de processo de combinação são gerados automaticamente com a ajuda de uma *ferramenta de combinação*; eles também podem ser considerados diagramas que oferecem uma perspectiva composicional em relação a diagramas de aspectos e diagramas de classes definidos separadamente.

Os elementos combinados modelam os resultados da combinação entre os elementos base e os elementos *crosscutting* . Os resultados de combinação dependem da *estratégia de combinação* adotada, que depende do modelo de implementação suportado pela ferramenta ou linguagem de programação

orientada a aspectos. A `aSideML` oferece suporte a dois tipos de estratégias de combinação: *modificação local* e *migração do cliente* (Seção 3.1.4). Neste trabalho, apresentamos apenas exemplos com modificações locais.

5.4.1 Classes combinadas

Semântica

Uma *classe combinada* é um elemento gerado a partir de uma classe (elemento base) e um ou mais elementos *crosscutting* (aspectos) relacionados pelos relacionamentos de *crosscutting*.

É uma classe com adornos adicionais. Os adornos indicam as características estendidas pelos aspectos. As classes combinadas são representadas em diagramas de processo de combinação com a ajuda de ferramentas (Seção 5.5).

Notação

Uma classe combinada é mostrada como uma pilha de retângulos tracejados, deslocados ligeiramente na direção horizontal e vertical. O primeiro retângulo descreve a classe base, estendida com atributos e operações introduzidos pelos aspectos que a afetam. Os outros retângulos exibem o nome desses aspectos e também losangos pequenos à direita de cada característica de classe afetada, seja atributo ou método (Figura 5.29).

Os losangos representam marcas de apresentação que fazem a distinção entre adições, refinamentos e redefinições fornecidos por cada aspecto (Figura 5.30).

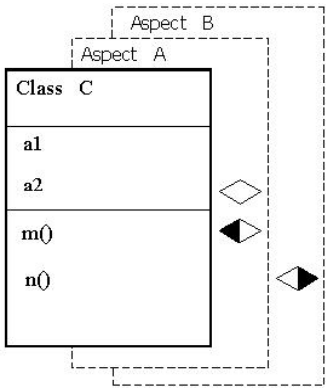

Figura 5.29: Notação.

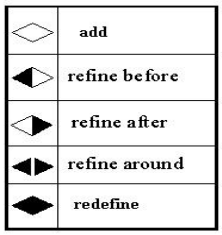

Figura 5.30: Notação.

Opções de apresentação

As classes podem ser apresentadas em um estilo não-hierárquico (*flat style*) que descreve todas as características, mesmo as herdadas, seguindo a abordagem sugerida em [43]. O compartimento do nome da classe é adornado com o símbolo Λ e as características de classe são adornadas com um retângulo que indica se são herdadas, refinadas ou redefinidas (Figura 5.31).

Usando o estilo não-hierárquico, os aspectos que afetam uma superclasse e que são herdados por uma classe base podem ser explicitamente associados a ela no diagrama de processo de combinação.

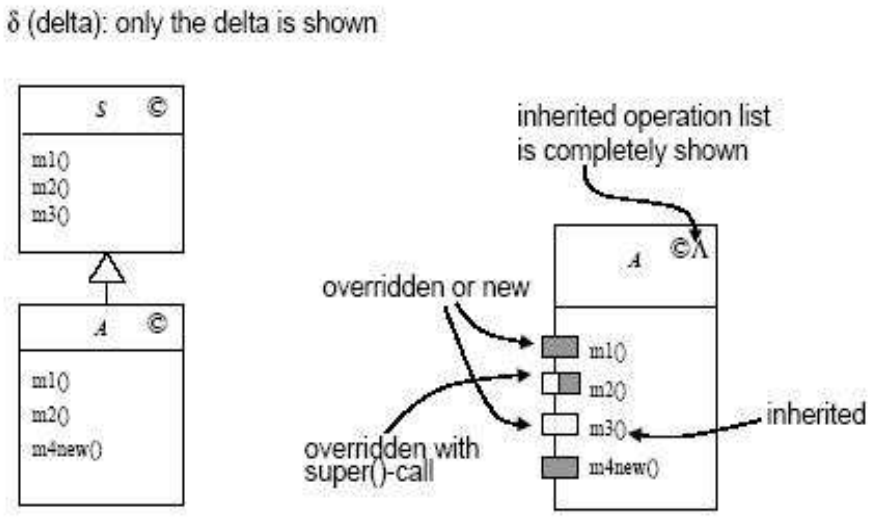

Figura 5.31: Estilo não-hierárquico [43].

Exemplo

A Figura 5.33 apresenta duas classes combinadas, `Button` e `ColorLabel`, afetadas pelo aspecto Observation. As classes que contêm

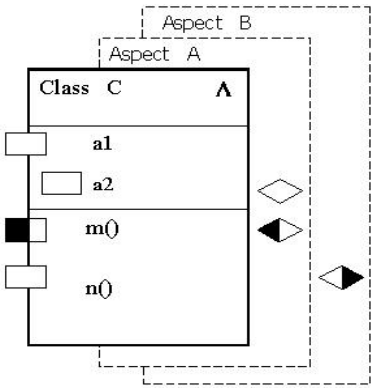

Figura 5.32: Classe combinada no estilo não-hierárquico.

uma chamada a `click()` (com operações estendididas pela operação `stateChange_`) não são mostradas.

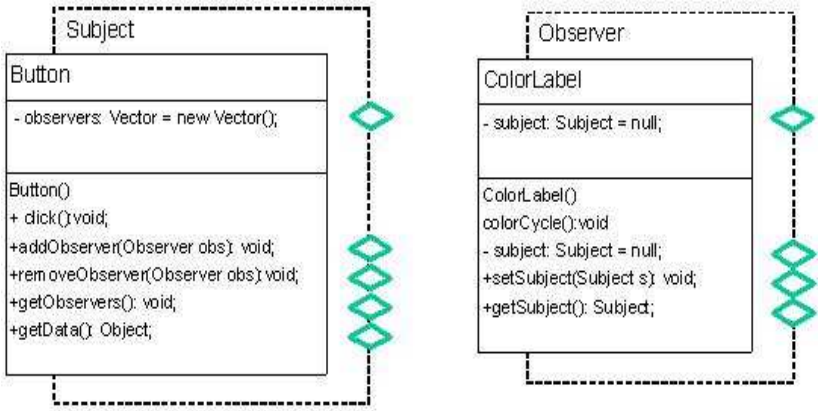

Figura 5.33: Classes combinadas: Button and ColorLabel.

5.4.2 Colaborações combinadas

Semântica

Uma *colaboração combinada* é um elemento de apresentação que descreve o resultado do processo de combinação de uma colaboração de UML (elemento base) e uma ou mais colaborações aspectuais.

A colaboração base em UML descreve o contexto para a chamada de uma operação base (Figura 5.34). Cada colaboração aspectual descreve o contexto de uma operação de *crosscutting* (Figura 5.22).

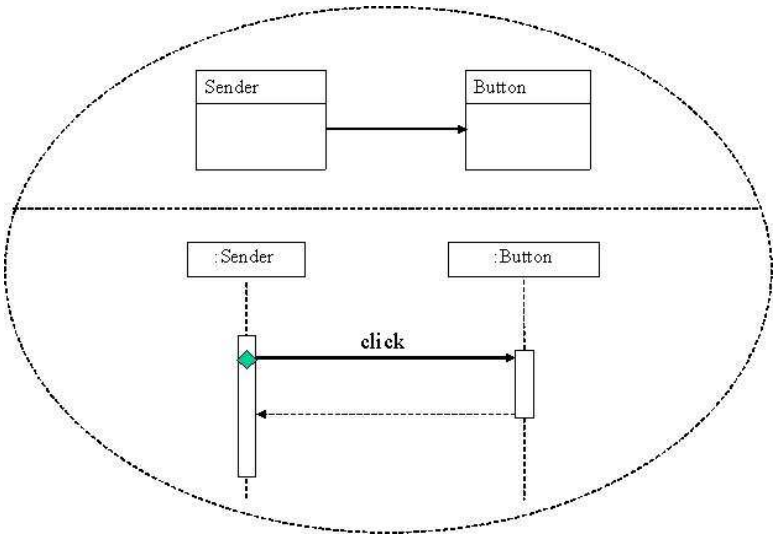

Figura 5.34: Colaboração base.

Uma colaboração combinada é uma colaboração adornos adicionais. Os adornos indicam as extensões. As colaborações combinadas são representadas em diagramas de processo de combinação com a ajuda de ferramentas (Seção 5.5).

Exemplo

A Figura 5.35 apresenta a colaboração combinada gerada depois da combinação da colaboração base da Figura 5.34 e a colaboração aspectual da Figura 5.22.

As instâncias de aspecto são representadas como objetos comuns, mas retêm a cor ou padrão atribuídos como um adorno, a fim de enfatizar o comportamento que vem dos aspectos.

A Figura 5.36 apresenta a colaboração combinada gerada depois da combinação da colaboração base de UML da Figura 5.34, a colaboração aspectual da Figura 5.22 e as colaborações aspectuais definidas para o aspecto `Tracing`.

5.4.3 Interações combinadas

Uma *interação combinada* é um elemento de apresentação que descreve o resultado do processo de combinação de uma interação e uma ou mais interações aspectuais.

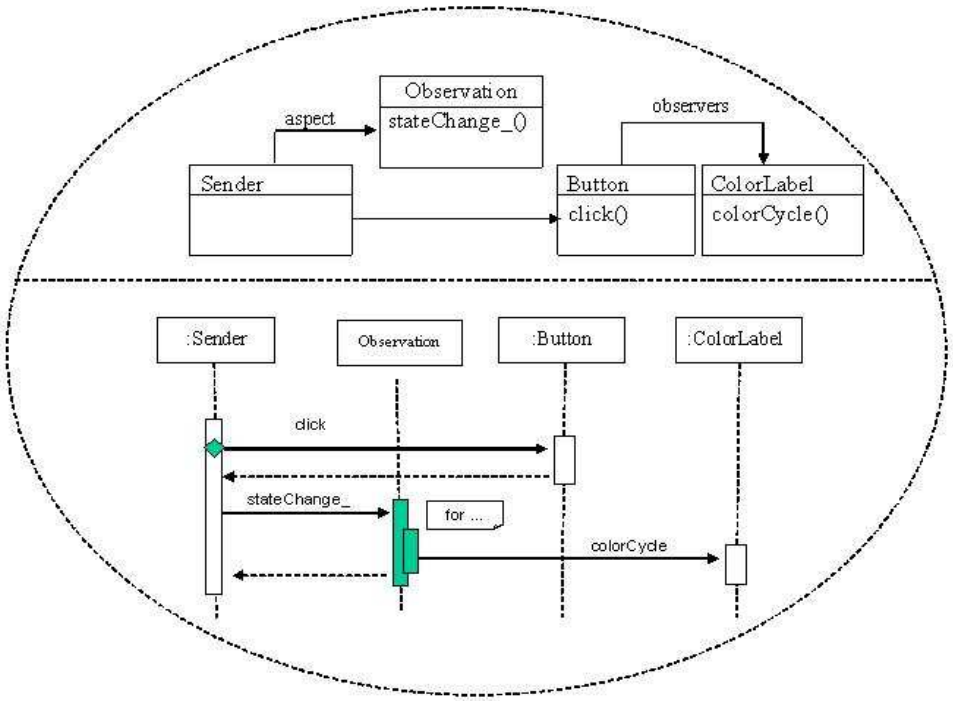

Figura 5.35: Colaboração combinada: `click` e `stateChange_`.

As interações combinadas são definidas no contexto das colaborações combinadas. As notações e os exemplos são apresentados na Seção 5.4.2.

5.4.4 Diagramas de processo de combinação

Um *diagrama de processo de combinação* oferece uma apresentação gráfica para uma coleção de elementos combinados – elementos base adornados de forma a enfatizar as extensões proporcionadas pelos elementos *crosscutting*.

Exemplo

A Figura 5.33 apresenta um diagrama de processo de combinação que descreve duas classes combinadas. A Figura 5.35 apresenta um diagrama de processo de combinação que descreve uma colaboração combinada.

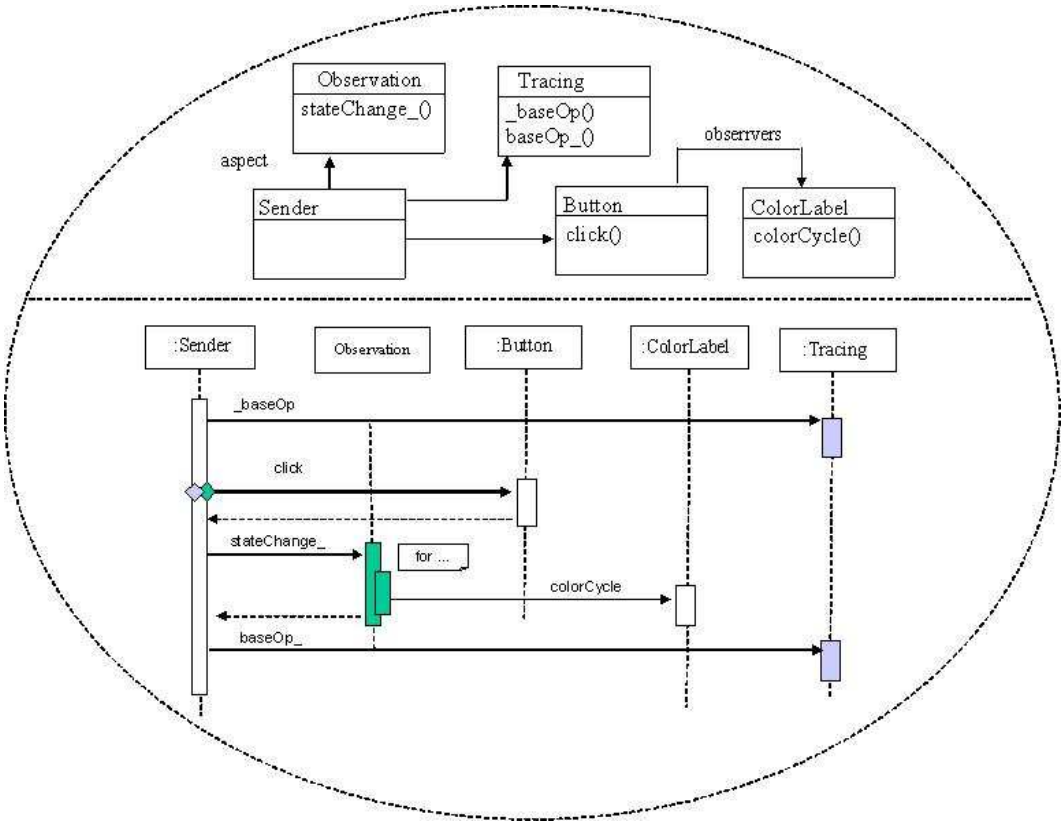

Figura 5.36: Colaboração combinada: `click`, `stateChange_` e `baseOp_`.

5.5 Usando `aSideML`

Nas seções anteriores, tratamos da descrição dos artefatos de apresentação suportados por `aSideML`, que permitem aos projetistas especificarem e visualizarem os modelos de aspectos. O padrão de projeto `Observer` foi modelado com objetos e aspectos.

Por exemplo, para *especificar* o aspecto `Observation` e compô-lo com outros elementos de modelagem, o projetista deve:

- descrever o aspecto `Observation` em um diagrama de aspecto;
- descrever as colaborações aspectuais para cada característica transversal em diagramas de colaboração aspectuais;
- descrever o comportamento de cada operação de *crosscutting* usando diagramas de seqüência;
- usar diagramas de classes estendidos para apresentar classes, aspectos e relacionamentos de *crosscutting* .

Para *visualizar* o aspecto `Observation`, um projetista pode:

- usar diagramas de classes estendidos para visualizar aspectos, classes e seus relacionamentos estruturais;
- passar da visão condensada para a total nos diagramas aspectuais para ter uma visão detalhada do aspecto;
- usar diagramas centrados em base para selecionar uma classe base e entender a forma como o aspecto a estende;
- usar diagramas centrados em aspectos a fim de visualizar as classes estendidas pelo aspecto, apesar de esse tipo de diagrama ser mais apropriado para a apresentação de aspectos sistêmicos.

A `aSideML` deve ter o suporte de ferramentas CASE interativas. As ferramentas e sua interoperabilidade dependem de uma semântica sólida e da definição de uma notação, como a fornecida pelos metamodelos de UML e de `aSideML`.

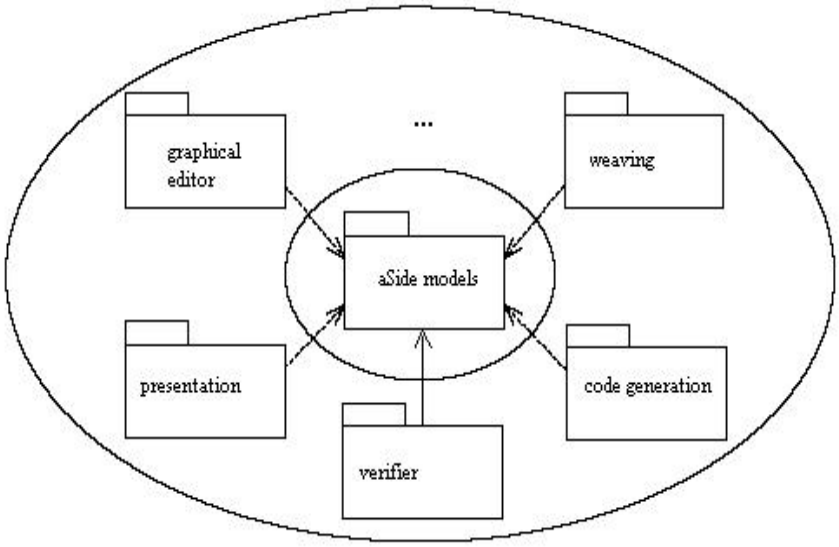

Figura 5.37: Ferramentas para `aSideML`.

Algumas ferramentas necessárias para o suporte às características descritas ao longo deste trabalho são:

- Ferramenta de edição gráfica: responsável pela representação de elementos nos diagramas.
- Ferramenta de visualização: responsável pela apresentação dos diferentes tipos de diagramas e pelo fornecimento de capacidades de navegação para navegar entre os elementos relacionados.

– Ferramenta de gerenciamento de dados: responsável pelo armazenamento da representação de projeto orientado a aspectos em alguma forma intermediária.

– Ferramenta de combinação:responsável pela geração dos modelos de processo de combinação. A ferramenta de combinação instrumenta a representação dos elementos base e insere características estruturais e chama os métodos de aspecto em pontos de combinação específicos, a fim de gerar os elementos combinados.

– Ferramenta de casamento de padrão: responsável pelo tratamento de quantificação em um grande conjunto de elementos de modelagem.

– Ferramenta de geração de código: responsável pela geração de código de POA para algum modelo de aspecto no nível da implementação.

– Ferramenta de análise: responsável pela verificação da corretude das representações de modelo de aspectos de nível de design e por reportar as inconsistências e as interferências. Pode ser usado em qualquer etapa do processo de modelagem.

– Ferramenta de medição: responsável pela aplicação de um conjunto de métricas [118].

5.6 Trabalhos Relacionados

No Capítulo 4,, apresentamos duas abordagens diferentes que oferecem as contribuições mais representativas para a modelagem orientada a aspectos no nível de design – o Modelo de Projeto Orientado a Aspectos (AODM) [128] e os Padrões de Composição [29]. As duas abordagens estendem UML com meios para o design de programas orientados a aspectos e oferecem a notação para a visualização e comunicação de modelos de aspectos. Nesta Seção, concentramo-nos na discussão e na notação. A Tabela 5.4 apresenta as características dos Padrões de Composição, AODM e `aSideML` de acordo com os requisitos apresentados na Seção 4.3.

As três abordagens tratam da modelagem de artefatos de nível de design na etapa de projeto e não consideram a modelagem de requisitos. *Crosscutting* é tratado nos diagramas de classes de UML e nos diagramas de colaboração.

A `aSideML` e os Padrões de Composição se valem da *parametrização* a fim de facilitar a definição de aspectos genéricos que podem ser reutilizados

	AODM	CP	aSideML
compatível com UML	sim	sim	sim
dependência de linguagem	AspectJ	SOP	customizável
aspectos são de primeira classe	"aspectos" são classes estereótipadas	"aspectos" são pacotes estereótipados	aspectos são `Aspect` (nova metaclasse)
interfaces de aspecto	não separadas; operações com estereótipos	pattern classes e operações template	interfaces transversais
apresentação de pontos de combinação	elementos de modelagem "highlighted"	argumentos template	adornos e argumentos template
interações explícitas entre aspectos	não	não	sim
extensões sensíveis ao contexto	sim	não	sim
semântica para crosscutting	AspectJ	delegação	customizável
quantificação	sim	sim	sim
visão do sistema combinado	sim	Subject output	sim
ferramenta	não	não	não

Tabela 5.4: Comparação.

em diferentes contextos. AODM oferece suporte à herança entre aspectos concretos e abstratos conforme definido em AspectJ.

Os mapeamentos de AODM para AspectJ e de Padrões de Composição para Hyper/J são diretos. AODM oferece suporte à descrição de várias características no nível da implementação, bem acopladas a AspectJ, o que facilita esse mapeamento. A `aSideML` oferece mapeamentos preliminares para AspectJ e Hyper/J e oferece suporte à expressão de *modificação local* ou *migração cliente* no nível de projeto, dependendo da estratégia de combinação adotada (Apêndice A).

Os Padrões de Composição especificam *concerns* transversais de uma maneira orientada a sujeitos que é inapropriada para o design de programas orientados a aspectos em AspectJ por várias razões diferentes. Padrões de Composição são limitados pelo modelo de pontos de combinação de Hyper/J (não há pontos de combinação para chamada de método, por exemplo); Hyper/J oferece suporte a um subconjunto de modelo de pontos de combinação

de AspectJ. Padrões de Composição não representam "advice" como uma característica diferenciada de aspectos. Eles têm de ser extraídos de diagramas de interação que descrevem que tipo de advice é (por exemplo, before, after ou around). Outras limitações dessa abordagem relativas ao modelo de AspectJ são apresentadas em [31, 127].

Finalmente, a `aSideML` está muito mais preocupada com a compreensibilidade para a visualização e a comunicação dos modelos de aspectos e sua arquitetura do que as duas outras abordagens:

- ela fornece um conjunto de diagramas e visões que apresentam aspectos como elementos de primeira classe e *crosscutting* como um relacionamento explícito entre aspectos e classes;
- o uso de diferentes símbolos gráficos (e mesmo cores) para aspectos, pontos de combinação, elementos combinados melhora a compreensibilidade, mas coloca mais requisitos para o ferramental de suporte;
- vários relacionamentos entre aspectos podem ser explicitamente apresentados;
- o uso de interfaces transversais explícitas é uma característica única de `aSideML` que melhora a modularidade de aspectos. A definição de compartimentos para adições, refinamentos e redefinições, com adornos como before, after, etc., oferece uma notação que enfatiza aspectos e suas características transversais como mecanismos úteis para a extensão incremental.

5.7 Considerações Finais

A seguir, discutimos alguns problemas da modelagem orientada a aspectos que demandam investigação futura a fim de melhorar a expressividade de `aSideML` e a compreensibilidade de seus modelos de aspectos.

Reutilização e evolução de aspectos

AspectJ oferece suporte a uma forma limitada de herança entre aspectos. Os aspectos também podem herdar características de classes. Eles podem estender outros aspectos, e os subaspectos podem herdar métodos e atributos locais e pointcuts. Um subaspecto pode sobrepor qualquer pointcut herdado de superaspectos; no entanto, os aspectos só podem estender

aspectos abstratos, e advice e introduções não podem ser refinados nem redefinidos. Assim, um subaspecto só pode modificar os pontos de combinação em que os efeitos de *crosscutting* de superaspectos no comportamento tiverem de ser executados. Ele não pode alterar o comportamento *crosscutting* propriamente dito. Hyper/J, Filtros de Composição e Programação Adaptativa não oferecem suporte à herança entre hyperslices, filtros e métodos adaptativos.

A `aSideML` oferece aspectos parametrizados para fornecer suporte à reutilização de aspectos em diversos contextos. Não oferecemos herança entre aspectos. Como trabalho futuro, os aspectos de `aSideML` podem ser especializados por herança, onde os *subaspectos* herdam as propriedades de seus *superaspectos*.

As propriedades herdáveis seriam características locais (atributos e métodos) e as interfaces transversais. Os subaspectos definiriam novas características locais ou sobreporiam características locais existentes; também poderiam definir novas características transversais, organizadas em novas interfaces transversais ou como extensões de interfaces transversais existentes. A Figura 5.38 apresenta dois aspectos relacionados através de generalização.

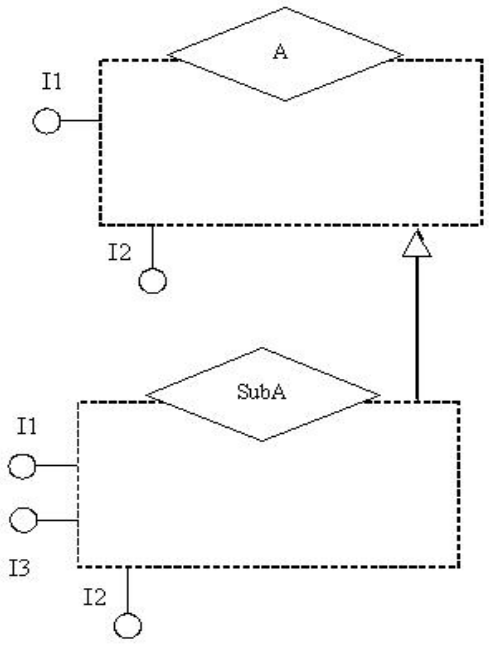

Figura 5.38: Herança e aspectos.

Quantificação de aspectos sistêmicos

Como discutido na Seção 4.1.3, os aspectos sistêmicos requerem mecanismos de quantificação mais poderosos, envolvendo o uso de expressões regulares a fim de facilitar a especificação dos pontos de combinação de interesse.

O conjunto de substituições dos parâmetros de template de aspectos usado no relacionamento de *crosscutting* , fornecido pela `aSideML`, pode ser estendido para oferecer suporte a uma sublinguagem de quantificação mais expressiva.

Visualização de aspectos sistêmicos

A apresentação visual de aspectos não-sistêmicos e seus relacionamentos a classes base em diagramas de classes estendidos é provavelmente mais compreensível e fácil de evoluir, mesmo quando estão envolvidos vários aspectos.

Contudo, a apresentação visual de relacionamentos entre aspectos sistêmicos e classes afetadas em diagramas de classes estendidos (usando o relacionamento de *crosscutting*) e também em diagramas de processo de combinação (usando adornos) permanece um desafio.

A solução oferecida é evitar o uso de relacionamentos de *crosscutting* explícitos para aspectos sistêmicos usando os detalhes omitidos e a perspectiva de pontos de combinação. A apresentação visual dos detalhes dos relacionamentos de *crosscutting* seria fornecida em diagramas centrados em base.

Especificação e visualização de crosscutting

Na Seção 3.1, definimos que uma sombra estática é uma região no componente que representa um ponto de combinação dinâmico relacionado a tal componente. Por exemplo, a sombra estática de um ponto de combinação de execução de método corresponde a um método que pertence a uma classe base.

O relacionamento de *crosscutting* é usado explicitamente em diagramas de classes para relacionar elementos *crosscutting* a elementos base, indicando os pontos em que os elementos base são afetados. A notação proposta é adequada para a descrição de *crosscutting* estrutural estático.

Além disso, ela é adequada para expressar *corsscutting dinâmico em execuções de métodos* uma vez que sua sombra estática faz parte da classe base e pode ser inferida a partir do nome do método fornecido na lista de associações do relacionamento de *crosscutting*.

O *crosscutting dinâmico em chamadas de métodos*, entretanto, traz um problema: a sombra estática de uma chamada de método é um método que contém a chamada, não necessariamente um método que pertence ao elemento base apontado pelo relacionamento *crosscutting*. Não podemos definir os elementos base que contêm chamadas de métodos a partir do nome do método chamado em modelos estruturais como diagramas de classes, somente a partir de modelos comportamentais que descrevem interações entre chamadores e chamados.

Por exemplo, as classes que contêm uma chamada a `click()` (com operações afetadas pela operação `stateChange_`) só foram representadas em uma colaboração aspectual, denotada por `Sender` (Figura 5.22). A identificação explícita de classes e métodos que contêm chamadas de métodos em modelos estruturais só é possível por meio de diagramas de processo de combinação e classes combinadas.