Padrões para Integração de Visões Modeladas com UML

Este exemplar corresponde à redação final da Dissertação devidamente corrigida e defendida por Fabiana Gomes Marinho e aprovada pela Banca Examinadora.

Fortaleza, 4 de fevereiro de 2002.

Profa. Dra. Vânia Maria Ponte Vidal
(Orientadora)

Dissertação apresentada ao Mestrado em Ciência da Computação, UFC, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Mestrado em Ciência da Computação
Universidade Federal do Ceará

Padrões para Integração de Visões Modeladas com UML

Fabiana Gomes Marinho[1]

Fevereiro de 2002

Banca Examinadora:

- Profa. Dra. Vânia Maria Ponte Vidal (Orientadora)
- Prof. Dr. Décio Manuel da Fonseca
- Profa. Dra. Rossana Maria de Castro Andrade
- Prof. Dr. Angelo Roncalli Alencar Brayner

[1]Bolsista da CAPES

Resumo

O projeto de um banco de dados é composto de duas fases principais - projeto conceitual e projeto lógico. No projeto conceitual, o projetista especifica o banco de dados em termos do modelo semântico adotado. O resultado dessa fase constitui o esquema conceitual da aplicação. No projeto lógico, o esquema conceitual é traduzido em um dos modelos tradicionais de implementação. O resultado dessa fase é o esquema lógico da aplicação.

Durante o processo do projeto do esquema conceitual, as visões dos usuários são abstraídas e represemtadas. Em seguida, as várias visões são integradas em um esquema conceitual global que satisfaz os requisitos de toda a organização (esquema integrado). Em nosso enfoque o processo de integração de visões é dividido em três passos principais: *decomposição*, *combinação* e *otimização*. No entanto, qualquer metodologia para integração de visões deve ser baseada em um modelo de dados. Nós adotamos o modelo de objetos da UML. Na UML, a modelagem é descrita através da notação definida por seus vários tipos de diagramas. Neste trabalho focalizamos nossas atividades no diagrama de classes da UML.

A maior contribuição deste trabalho está em um catálogo de padrões desenvolvido com o objetivo de auxiliar os projetistas no processo de modelagem e integração de visões com a UML. Contudo, entendemos que, para usar os construtores de modelagem conceitual de forma correta, é preciso que o seu significado seja definido de forma rigorosa, o que não acontece com os construtores do modelo de objetos da UML. Outra contribuição importante do nosso trabalho está, portanto, na formalização do significado dos construtores do modelo de objetos da UML para que estes possam ser usados adequadamente na especificação dos padrões propostos.

Agradecimentos

"A vantagem de percorrer caminhos longos e árduos é poder ver as maravilhas que Deus realiza em nossas vidas."

Neste momento de realização e alegria, agradeço aos meus pais (Luiz e Zélia), irmãos (Luis Carlos, Alexandra e Luciana) e ao meu grande amigo Hortêncio, que estiveram sempre presentes nesta luta, me apoiando e animando, principalmente nos momentos em que parecia impossível a vitória.

Às minhas amigas, Danielle, Juliana, Renata e Bernadette, agradeço pelos momentos de incentivo mútuo compartilhados diante de todas as dificuldades enfrentadas durante esse período.

Aos meus amigos, Andréa e Marcelo, pelo exemplo e motivação sempre demonstrados.

Em especial, à minha orientadora Vânia, pela enorme paciência e tempo dispensados na orientação deste trabalho, todo o meu respeito e consideração.

Ao Departamento de Computação da Universidade Federal do Ceará e à Capes, agradeço pela oportunidade de tão grande crescimento profissional e pessoal que me proporcionaram.

Agradeço, principalmente, à Deus por toda força dada para realizar este trabalho. Por sua infinita bondade e generosidade comigo, pelo amparo nas tribulações e pelos amigos que se fizeram canal do Seu Amor.

E a todos que contribuíram direta ou indiretamente para a realização deste trabalho, meu eterno agradecimento. Esta vitória não é só minha, mas de todos que estiveram presentes nesta caminhada.

Conteúdo

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

O projeto de um banco de dados deve garantir que os dados necessários para o seu grupo de usuários estejam armazenados e que sejam facilmente acessados por um conjunto de aplicações específicas. Como mostrado na figura 1.1, é possível identificar 4 fases principais no desenvolvimento do projeto de um banco de dados: 1) análise dos requisitos, 2) projeto conceitual, 3) projeto lógico e 4) projeto físico.

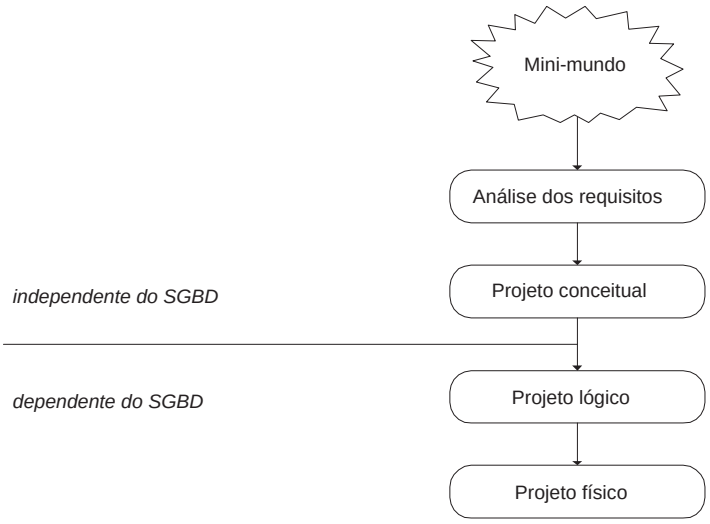

Figura 1.1: Projeto de banco de dados

1. *Análise dos requisitos* - durante essa fase, os projetistas analisam os requisitos dos dados dos vários grupos de usuários. Como resultado, é obtida uma especificação preliminar do domínio da aplicação.

2. *Projeto conceitual* - o principal objetivo dessa fase é produzir um esquema conceitual, utilizando para tal um modelo de dados de alto nível (modelo semântico). O esquema conceitual descreve de forma concisa os requisitos dos dados dos usuários e fornece ao projetista uma visão global do problema, independente de aspectos específicos de implementação.

3. *Projeto lógico* - durante o projeto lógico, o esquema conceitual é mapeado do modelo de dados de alto nível para o modelo de dados do sistema de gerenciamento de banco de dados escolhido, resultando no esquema lógico do banco de dados.

4. *Projeto físico* - durante essa fase, a organização dos arquivos e as estruturas internas de armazenamento são especificadas. Parâmetros físicos para o desempenho do banco de dados são determinados.

A modelagem conceitual é considerada a fase crucial do processo de projeto de um banco de dados. Durante essa fase o projetista deve compreender, estruturar e representar a semântica do mundo real através de um modelo de dados de alto nível, de modo que esses aspectos possam ser incorporados a um sistema de informação e reflitam as necessidades dos usuários e aplicações de forma natural e direta. Erros introduzidos no esquema conceitual podem se estender para as próximas fases do projeto, tornando-se cada vez mais caros e difíceis de serem removidos. Construir um bom esquema requer bastante cuidado e um entendimento detalhado do domínio da aplicação a ser modelada para que esta possa ser representada de forma clara e precisa.

De acordo com [EN00], existem duas abordagens distintas para o projeto do esquema conceitual. A primeira abordagem é centralizada. Nessa abordagem, os requisitos das diferentes aplicações e grupos de usuários são combinados em um conjunto de requisitos e um único esquema correspondendo a esse conjunto de requisitos é projetado. A segunda abordagem é baseada na integração de visões. Nessa abordagem, cada grupo de usuários analisa seus requisitos e especifica suas visões dos dados. Em seguida, as várias visões (esquemas) dos usuários são integradas em um esquema conceitual global (esquema integrado) que satisfaz os requisitos de toda a organização. Esse processo é denominado integração de visões. Segundo [SNL86], dois motivos principais justificam a necessidade de integração de visões durante o projeto conceitual de um banco de dados:

1. a estrutura de um banco de dados para grandes aplicações é bastante complexa de ser modelada por um único projetista em uma única visão;

2. em geral, os grupos de usuários trabalham de forma independente nas organizações e possuem seus próprios requisitos dos dados, que podem conflitar com os interesses de outros grupos.

Em nosso trabalho, adotamos a segunda abordagem para o projeto do esquema conceitual do banco de dados. No nosso enfoque, o processo de integração de visões é dividido em três atividades principais: *decomposição*, *combinação* e *otimização*. Durante a decomposição, as classes são analisadas com o objetivo de eliminar as restrições de dependência funcional indesejáveis de forma a reduzir as redundâncias do esquema e minimizar as anomalias de inserção, remoção e atualização, além de identificar a presença de classes embutidas no esquema conceitual. Durante a combinação, as várias visões dos usuários são analisadas e comparadas para determinar correspondência entre conceitos, descobrir relacionamentos entre visões e detectar possíveis conflitos. Os objetivos do passo de otimização de visões são:

(i) compatibilizar as visões de forma que a fusão das partes comuns seja possível;

(ii) aumentar a clareza e expressividade do esquema, capturando, tanto quanto possível, as restrições de integridade na própria estrutura do esquema;

(iii) reduzir o tamanho do esquema através da fusão de partes comuns.

Qualquer metodologia para integração de visões deve ser baseada em algum modelo de dados. Um modelo de dados é a base para especificarmos as visões e os relacionamentos semânticos entre os conceitos nas diferentes visões [Vid94]. O modelo de dados Entidade e Relacionamento (modelo ER) tem sido o método dominante para a modelagem conceitual dos dados. De acordo com [Lós98], esta popularidade se deve aos seguintes fatores: (i) é um modelo relativamente fácil de usar, (ii) grande variedade de ferramentas CASE suportam este modelo, (iii) acredita-se que entidades e relacionamentos são conceitos de modelagem natural no mundo real e (iv) é um modelo rico em semântica, o que facilita o entendimento das correspondências existentes entre os componentes dos esquemas.

Entretanto, o modo elegante pelo qual o paradigma da orientação a objeto permitiu representar os conceitos do mundo real como objetos com estrutura e comportamento associados, atraiu os pesquisadores. Como mostrado na figura 1.2, à medida que esses pesquisadores se aprofundavam nas diferentes abordagens de análise e projeto orientados a objeto, diversas linguagens de modelagem começaram a surgir, destacando-se entre elas, OMT [R$^+$92], Booch [Boo94] e OOSE [J$^+$92]. Apesar de guardarem alguma semelhança entre si, essas linguagens de modelagem apresentam também sensíveis diferenças quanto ao nível de descrição dos problemas e divergências técnicas de representação, dificultando o aprendizado de novos usuários e impossibilitando uma comunicação consistente e efetiva entre os grupos de desenvolvimento de software.

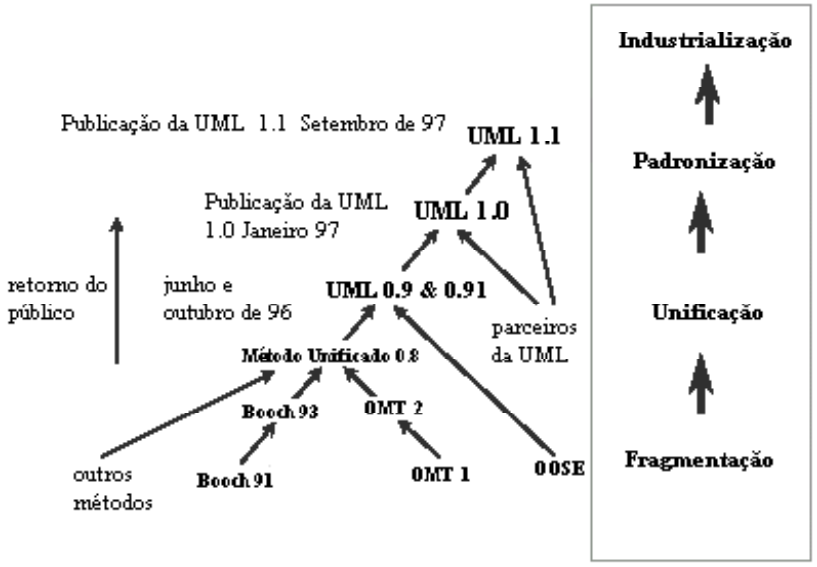

Figura 1.2: Evolução da UML - Figura retirada de [BJR97]

Essa situação motivou o desenvolvimento da UML - *Unified Modelling Language* - em outubro de 1994, quando Grady Booch e James Rumbaugh começaram o trabalho de unificação dos métodos Booch e OMT, tendo como principais objetivos:

- prover uma linguagem de modelagem visual expressiva, que permita a troca de modelos entre usuários sem perda de informação e sem exigir que seus modelos sejam mapeados para um formato abstrato;
- encorajar o desenvolvimento de ferramentas orientadas a objeto fornecendo uma notação e semântica padrão;
- criar uma linguagem de modelagem que seja independente das linguagens de programação e processos de desenvolvimento específicos;
- fornecer mecanismos para estender os elementos base. Os usuários serão capazes de (i) construir modelos usando os conceitos básicos; (ii) adicionar novos conceitos e notações; (iii) escolher entre as várias interpretações dos conceitos existentes; e (iv) especializar conceitos, notações e restrições de acordo com o domínio da aplicação;

Em outubro de 1995 a versão 0.8 da UML foi desenvolvida. No mesmo ano, Ivar Jacobson uniu-se ao grupo. Os esforços de Booch, Rumbaugh e Jacobson resultaram na criação das versões 0.9 e 0.91, respectivamente, em junho e outubro de 1996. A partir de 1996, várias organizações viram na UML um ponto estratégico para seus negócios e

uniram forças para produzir uma definição mais rigorosa da versão 1.0. Em setembro de 1997, a versão 1.1 foi lançada com o propósito de enriquecer a semântica da versão 1.0.

Por ser unificada e facilitar a interpretação correta e sem ambigüidade dos modelos gerados por diferentes grupos de desenvolvimento, a UML mostrou todos os sinais de se tornar a linguagem de modelagem definitiva para especificar, visualizar, documentar e construir aplicações orientadas a objeto. O OMG - *Object Management Group* - adotou a UML como a notação padrão para modelagem orientada a objeto [Mul99].

A UML possui elementos dos três principais padrões de projeto orientado a objeto: OMT [R+92], Booch [Boo94] e OOSE [J+92]. O resultado dessa combinação é um conjunto de diagramas que definem sua notação: diagramas de caso de uso, diagramas de classes, diagramas de transição de estado, diagramas de comportamento (diagramas de atividade e diagramas de interação) e diagramas de implementação. Os diagramas de caso de uso mostram as fronteiras e principais funções da aplicação. Os diagramas de comportamento mostram a cooperação entre os diversos objetos envolvidos. A arquitetura física da aplicação é revelada pelos diagramas de implementação e para modelagem dos dados, a UML usa o diagrama de classes.

O diagrama de classes é um diagrama estático que descreve a estrutura de uma aplicação através de classes, atributos e relacionamentos. De várias formas, o diagrama de classes lembra o diagrama ER. Na verdade, a maioria das técnicas que a UML oferece para modelagem conceitual dos dados tiveram origem na modelagem ER [Mul99]. Entretanto, o diagrama de classes da UML é bem mais abrangente, já que pode ser usado para modelar interfaces e até mesmo instâncias de classes individuais. O foco do nosso trabalho, no entanto, está nas classes e seus relacionamentos, enfatizando o processo de integração das visões dos usuários.

Observamos que, apesar do importante papel desempenhado pela modelagem conceitual no desenvolvimento do projeto de banco de dados, pouco trabalho tem sido feito no sentido de aprimorar a capacidade de se obter esquemas conceituais com a UML que representem adequadamente os requisitos dos usuários e aplicações. Definir uma metodologia que ajude a promover bons projetos conceituais e que, de certa forma, padronize a maneira como estes devem ser desenvolvidos é, portanto, necessário.

A maior contribuição deste trabalho está no conjunto de padrões definido para auxiliar os projetistas nas atividades desempenhadas no processo de integração de visões, de modo a assegurar que essa tarefa seja realizada de forma segura e eficiente. Vale a pena ressaltar que o esquema integrado resultante do processo de integração proposto em nossa metodologia está representado de tal forma que é possível um mapeamento direto para

um esquema objeto-relacional, o que é feito em [Nor01].

Contudo, nós entendemos que, para usar os construtores de modelagem conceitual de forma correta, é preciso que o seu significado seja definido de forma rigorosa, o que não acontece com os construtores do modelo de objetos da UML. Uma outra contribuição deste trabalho está, portanto, na formalização do significado de alguns elementos do modelo de objetos da UML para que estes possam ser usados adequadamente na especificação dos padrões propostos. Esse formalismo é necessário porque é através dele que conseguimos validar as soluções propostas em nossos padrões.

A dissertação está dividida em cinco capítulos como se segue. No capítulo 2, apresentamos o modelo de objetos que foi utilizado na formalização dos padrões propostos. No capítulo 3, é discutido o processo de integração de visões e o estado da arte, onde apresentamos algumas das principais metodologias para integração de visões, descrevemos os modelos semânticos mais usados para a modelagem conceitual dos dados e discutimos brevemente os trabalhos relacionados. No capítulo 4, descrevemos detalhadamente o catálogo de padrões desenvolvido. O capítulo 5 contém nossas conclusões e direções para trabalhos futuros.

Capítulo 2

Modelo UML Estendido

A modelagem conceitual dos dados é descrita na UML pelo diagrama de classes, que através de classes, atributos e relacionamentos representa o esquema conceitual de uma aplicação. Este capítulo discute alguns elementos do modelo de objetos da UML que fazem parte do diagrama de classes. Grande parte dos conceitos fundamentais do paradigma orientado a objeto, incluindo classes, identidade dos objetos e herança, são discutidos. Outros conceitos, como encapsulamento e sobrecarga não são considerados, pois estão relacionados ao comportamento das aplicações, enquanto aqui, focalizamos apenas aspectos estruturais e semânticos dos esquemas conceituais. Na seção 2.1 apresentamos as perspectivas de visualização do diagrama de classes. Na seção 2.2, detalhamos os elementos de modelagem que compõem o diagrama de classes da UML. Na seção 2.3, discutimos as restrições de integridade necessárias para a formalização das regras de modelagem propostas no capítulo 4.

2.1 Perspectivas

Segundo [Fow97b], o diagrama de classes pode ser visualizado a partir de três perspectivas diferentes: *conceitual*, *especificação* e *implementação*. A seguir discutimos cada uma dessas perspectivas.

- **Conceitual:** na perspectiva conceitual, o diagrama de classes representa os conceitos do domínio. Esses conceitos serão implementados, no entanto, não existe um mapeamento direto para os construtores da linguagem que será utilizada durante a implementação. Nessa perspectiva, o modelo conceitual deve ser independente da linguagem utilizada para implementação.

- **Especificação:** na perspectiva de especificação, o diagrama de classes representa a interface do software, não a implementação. Estamos preocupados com tipos em vez de classes. Uma interface pode possuir muitas implementações devido a diversos fatores, tais como, ambiente de desenvolvimento, características de desempenho, distribuidor da linguagem adotada para implementação etc.

- **Implementação:** na perspectiva de implementação, o diagrama de classes representa os conceitos no nível da implementação.

Compreender a perspectiva adotada no diagrama de classes é essencial tanto para a construção como para a interpretação do diagrama. Infelizmente, os limites entre as três perspectivas não estão bem definidos na literatura. Na medida que apresentamos o diagrama de classes, nós enfatizaremos aqueles elementos de modelagem que dependem da perspectiva utilizada.

2.2 Os Elementos de Modelagem da UML

O diagrama de classes trata-se de uma estrutura lógica estática composta de uma coleção de elementos de modelagem que representam o esquema conceitual de uma aplicação. Nesta seção descrevemos os principais elementos de modelagem que compõem o diagrama de classes da UML, tais como classes, relacionamentos, atributos e operações. A figura 2.1 mostra um exemplo de um diagrama de classes típico para uma aplicação de processamento de pedidos.

2.2.1 Objeto

Os *objetos* representam entidades do mundo real. Um objeto possui dois componentes: *estado* (porção dos dados) e *comportamento* (porção funcional). O estado de um objeto é definido pelos valores de suas propriedades. O comportamento de um objeto é definido por um conjunto de operações que podem ser executadas nos objetos e define o modo como os objetos agem e reagem a estímulos externos em termos de mudança de estado e passagem de mensagens.

Qualquer modelo de dados deve conter uma forma de identificar unicamente cada objeto. O paradigma orientado a objeto fornece essa unicidade através de um identificador denominado *Object-Identifier* (OID). O OID designa univocamente o objeto na representação e é preservado até mesmo quando o estado do objeto muda completamente.

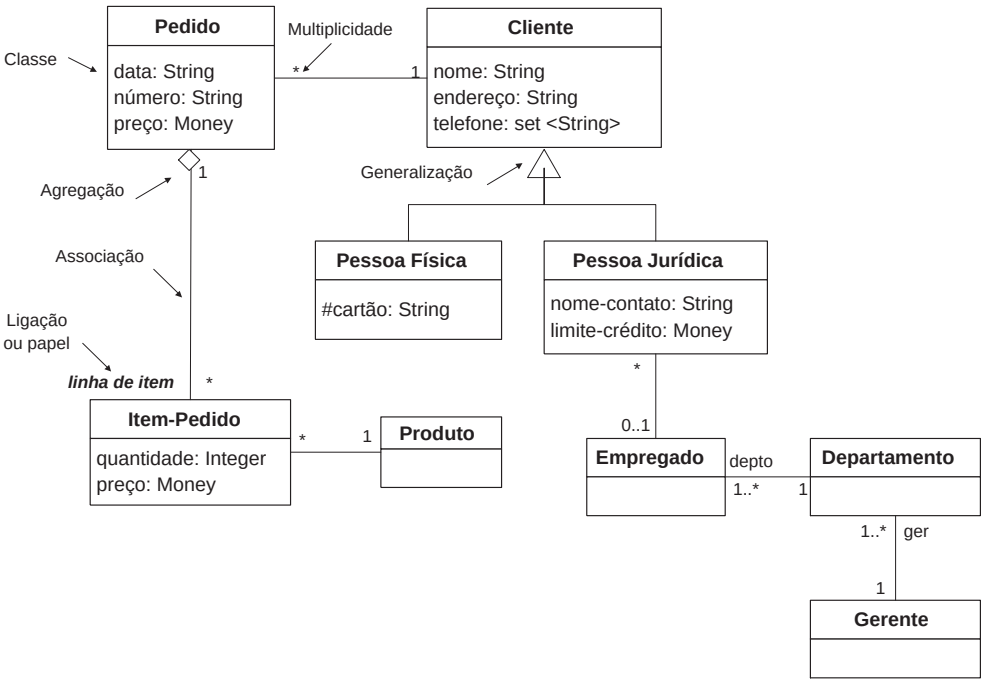

Figura 2.1: Exemplo - Diagrama de Classes

2.2.2 Classe

Na UML, uma *classe* é uma descrição de um conjunto de objetos que compartilham os mesmos atributos, operações, relacionamentos e semântica [Rat97]. Uma definição de classe possui o nome da classe, que deve ser único em todo o banco de dados, um tipo e uma extensão. A *extensão* de uma classe **C** (*Extensão*(**C**)) consiste em um conjunto de objetos que são membros ou *instâncias* de **C**. O *tipo* de uma classe **C** (*Tipo*(**C**)) é uma tupla <*propriedades*, *operações*> que indica o conjunto de propriedades e operações que são aplicados a todos os objetos de **C**. Uma propriedade de uma classe pode ser vista como uma definição dos dados mantida pelas instâncias da classe (atributo) ou como relacionamentos dos objetos da classe com objetos de outras classes (ligação). Uma operação é um serviço que uma instância de uma classe executa quando recebe uma mensagem de outro objeto. Esses conceitos serão definidos nas seções a seguir.

Na representação gráfica da UML, as classes são representadas como um retângulo sólido com três compartimentos. O primeiro compartimento é para o nome da classe, o segundo e o terceiro são usados para listar respectivamente os atributos e as operações definidas para a classe. Por exemplo, no diagrama de classes mostrado na figura 2.1, temos a classe **Cliente**, que possui como atributos **nome**, **endereço** e **telefone**. O

terceiro compartimento é opcional, sendo mais utilizado quando estamos interessados em visualizar o aspecto de implementação da aplicação.

2.2.3 Atributo

Um *atributo* é uma definição de dados mantida pelas instâncias de uma classe [Rat97]. Todo atributo tem associado um tipo que corresponde a uma classe pré-definida. O valor de um atributo é uma instância do tipo do atributo.

Os atributos podem ser classificados em monovalorados ou multivalorados. O valor de um atributo monovalorado é um objeto do tipo atômico. Os tipos atômicos são tipos que não possuem estrutura e representam um único valor. Tipos numéricos e caracteres são exemplos de tipos atômicos. No diagrama de classes mostrado na figura 2.1, o atributo **nome** da classe **Cliente** definido por **nome**: **String** é um exemplo de atributo monovalorado que associa uma instância da classe pré-definida **String** a um objeto da classe **Cliente**, indicando que um cliente possui um nome. O valor de um atributo multivalorado é um objeto que contém uma coleção de objetos que são instâncias de um mesmo tipo. Uma coleção de elementos associada a um objeto de uma classe pode ser implementada usando um dos construtores: *Array*, *Set* e *List*. *Set* é uma estrutura de dados que captura a noção usual de conjuntos matemáticos. Os construtores *Array* e *List* são estruturas que capturam a noção usual de listas e vetores usados nas linguagens de programação. Por exemplo, na figura 2.1, o atributo **telefone** da classe **Cliente**, definida por **telefone** : *set* <**String**>, associa um conjunto de instâncias da classe pré-definida **String** a um objeto da classe **Cliente**, indicando que um cliente pode ter mais de um número de telefone.

2.2.4 Operação

Uma classe incorpora um conjunto de atribuições que definem o comportamento dos objetos da classe e que são executadas por suas operações. Uma *operação* é um serviço que uma instância de uma classe executa sempre que recebe uma mensagem de outro objeto. Na UML, há uma distinção importante entre *operação* e *método*: uma operação é algo invocado por um objeto, enquanto um método é um corpo de um procedimento. Assim, um método é a implementação de uma operação [Fur98].

2.2.5 Associação

Os objetos contribuem para o comportamento de uma aplicação cooperando entre si. Essa cooperação é alcançada através das associações. Uma associação é um relacionamento

semântico entre duas ou mais classes que envolve conexões entre suas instâncias [Rat97]. Por exemplo, no diagrama da figura 2.1, temos que as instâncias da classe **Pedido** estão relacionadas com as instâncias da classe **Cliente**.

Ligações de uma Associação

Caso o diagrama de classes esteja representando uma perspectiva de implementação da aplicação, a notação da UML permite adicionar uma seta à extremidade da associação para indicar sua navegabilidade. Se a navegabilidade existir somente em uma direção, chamamos a associação de unidirecional. Uma associação bidirecional contém navegabilidades nas duas direções [Fow97a]. Em um nível mais conceitual, a UML usa as associações sem setas para indicar que a navegabilidade é bidirecional. Para cada direção de navegação é associada uma *ligação* ou papel. Uma ligação relaciona os objetos de uma classe com objetos da outra classe envolvida na associação. Usaremos a notação $l : \mathbf{A} \rightarrow \mathbf{B}$ para indicar que a ligação l associa à classe **A** os objetos da classe **B** que estão associados a **A**. Neste caso, dizemos que l é uma ligação da classe **A**. Por exemplo, no contexto da figura 2.1, temos que a classe **Item-Pedido** está associada à classe **Pedido** através da ligação *linha de item*. Isso implica que as instâncias da classe **Item-Pedido** estão associadas à uma instância da classe **Pedido** através da ligação *linha de item* (*linha de item*: **Item-Pedido** $\rightarrow$ **Pedido**), ou seja, *linha de item* é uma ligação da classe **Item-Pedido**.

As extremidades das associações também contêm uma multiplicidade. A multiplicidade é uma indicação de quantos objetos podem participar de uma associação. Na figura 2.1, o * na extremidade da associação entre a classe **Pedido** com a classe **Cliente** indica que um cliente tem muitos pedidos associados a ele, enquanto 1 na outra extremidade indica que um pedido vem somente de um cliente. Se o valor máximo da multiplicidade especificada para uma ligação for 1, então dizemos que a ligação é *monovalorada*; caso seja maior que 1, a ligação é *multivalorada*.

Atributos como Ligação

É importante ressaltar que, no nível de implementação, um atributo também pode ser visto como uma ligação. A diferença está apenas na navegabilidade desse dois conceitos. Um atributo possui um único sentido de navegação: do tipo para o atributo. Já as ligações podem existir nas duas direções de navegação.

Classe de Associação

No mundo real existem algumas situações cuja semântica não pode ser capturada através de associações binárias. Para representar tais situações são utilizadas as *classes de asso-*

ciação. Uma classe de associação é um elemento de modelagem que representa relacionamentos n-ários e permite que atributos, operações e outras associações sejam adicionados às associações. O número de classes ligadas à classe de associação corresponde ao grau da classe de associação.

Uma classe de associação pode ser vista tanto como uma associação que tem propriedades de classe ou como uma classe que tem propriedades de associação [Fur98]. Conforme mostrado no esquema da figura 2.2, na UML, uma classe de associação é represemtada por um símbolo de classe anexado a uma associação por uma linha tracejada e, apesar de possuir uma notação gráfica de uma associação e uma classe, corresponde a um único elemento de modelagem.

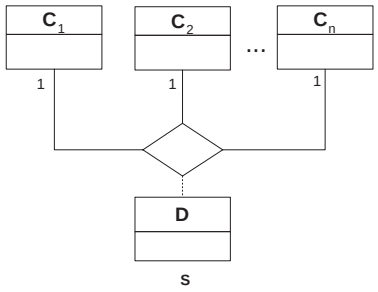

Figura 2.2: Notação de Classe de Associação na UML

Uma classe de associação captura um evento do mundo real que envolve um objeto de cada uma das classes participantes da associação. Isso significa que toda instância da classe de associação está associada com uma única instância de cada uma de suas classes participantes. Formalmente, existe uma ligação da classe de associação para cada classe participante, onde a multiplicidade dessas ligações é 1. Em geral, o nome dessa ligação corresponde ao nome da classe participante. Por exemplo, no esquema da figura 2.3, a classe de associação **Gerência** captura o fato de que um empregado **e** trabalha em um projeto **p** que é gerenciado pelo gerente **g**. Dessa forma, existe uma ligação da classe de associação **Gerência** para cada uma de suas classes participantes (*empregado*: **Gerência** $\rightarrow$ **Empregado**, *projeto*: **Gerência** $\rightarrow$ **Projeto** e *gerente*: **Gerência** $\rightarrow$ **Gerente**) cuja multiplicidade é 1.

Vale a pena ressaltar que as classes de associação adicionam um restrição importante às associações, na qual somente uma instância da classe de associação pode relacionar os mesmos objetos das classes participantes. Isso implica que não posso ter mais de uma instância da classe de associação relacionando os mesmos objetos das classes participantes. Por exemplo, no contexto do exemplo da figura 2.3, temos que só é possível associarmos um empregado a um mesmo projeto e gerente uma única vez.

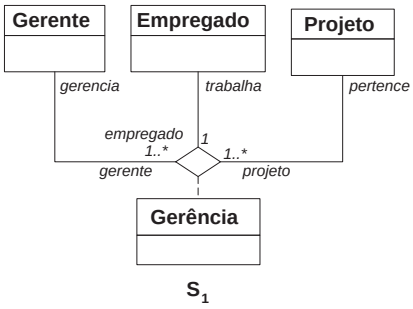

Figura 2.3: Exemplo - Classe de Associação

Agregação

Uma agregação é uma forma especial de associação usada para representar que um objeto é composto, pelo menos em parte, de outro em uma relação de todo-parte [Fur98]. A UML especifica dois tipos de agregação: agregação fraca ou compartilhada (*agregação)* e agregação forte ou composta (*composição*) [Mul99]. Uma agregação permite você modelar um relacionamento todo-parte no qual um objeto possui outro objeto, mas outros objetos também podem possuir aquele objeto. Uma composição permite você modelar o relacionamento todo-parte onde um objeto possui exclusivamente outro objeto.

Na UML, um diamante vazio é anexado ao fim da associação e próximo à classe que representa o todo para indicar uma agregação, enquanto a composição é representada por um diamante preenchido.

O esquema da figura 2.1, ilustra um exemplo de agregação entre as classes **Pedido** e **Item-Pedido**. A agregação descreve a situação em que um pedido agrega um conjunto de itens de pedido, mas um determinado item de pedido também pode aparecer em mais de um pedido. Semanticamente não existe um proprietário. A composição , por outro lado, é mais explícita quanto ao proprietário.

Apenas um papel em uma associação pode possuir uma agregação, ou seja, apenas um objeto pode possuir o outro objeto. Uma associação n-ária não pode ter uma agregação [Rat97]. Uma agregação composta, por definição, deve possuir uma multiplicidade 1, 1..1, ou 0..1 no papel correspondente ao proprietário [Mul99].

2.2.6 Generalização

Uma generalização é um relacionamento entre um elemento mais geral e um elemento mais específico. O elemento mais específico é totalmente consistente com o elemento mais geral e contém informação adicional. Uma instância do elemento mais específico pode ser

usada sempre onde o elemento mais geral é permitido [Rat97].

Na representação gráfica da UML, a generalização é representada por uma linha sólida do elemento mais específico (subclasse) para o elemento mais geral (superclasse), com um triângulo vazio na extremidade próxima ao elemento mais geral.

A figura 2.1, ilustra um exemplo típico de generalização envolvendo as classes **Pessoa Física** e **Pessoa Jurídica**. Essas classes possuem diferenças, mas também possuem similaridades. As similaridades são representadas por uma classe genérica **Cliente** (superclasse) tendo as classes **Pessoa Física** e **Pessoa Jurídica** como subclasses. Conceitualmente, dizemos que a classe **Pessoa Física** é um subtipo da classe **Cliente** se todas as instâncias da classe **Pessoa Física** também são, por definição, instâncias da classe **Cliente**. A idéia chave é a de que todos o atributos, associações e operações da classe **Cliente** são herdados pelas classes **Pessoa Física** e **Pessoa Jurídica**.

2.2.7 Esquema Orientado a Objeto

Um esquema orientado a objetos é uma tupla $\mathcal{S} = (\mathcal{C}, \mathcal{I})$, onde $\mathcal{C}$ representa um conjunto finito de classes e $\mathcal{I}$ é um conjunto de restrições de integridade.

O esquema de um banco de dados é alterado com pouca freqüência, no entanto, um banco de dados muda ao longo do tempo por meio das informações que nele são inseridas ou excluídas. O conjunto de informações contidas em determinado banco de dados em um dado momento é chamado *estado* ou *instância* do esquema do banco de dados.

O estado de uma classe **C** é o conjunto de objetos que pertencem à extensão de **C**, ou seja, o conjunto de objetos que são membros de **C** no estado corrente. O estado corrente de um objeto é definido pelos valores de suas propriedades (atributos e ligações). Considere **c** uma instância da classe **C**. Para qualquer atributo **a** de **C**, **c.a** retorna o valor do atributo **a** para a instância **c** no estado corrente. Para qualquer ligação l: $\mathbf{C} \rightarrow \mathbf{C}'$, $\mathbf{c}.l$ retorna o valor da ligação l para a instância **c** no estado corrente.

2.2.8 Caminho

Os objetos de uma classe também podem estar relacionados indiretamente com os objetos de outras classes através da composição de duas ou mais ligações. Suponha, por exemplo, o diagrama da figura 2.1. De acordo com as ligações definidas entre as classes **Empregado**, **Departamento** e **Gerente**, temos que um empregado está relacionado com o gerente do seu departamento de forma indireta através da composição das ligações *depto* e *ger*,

ou seja, existe um *caminho* entre as classes **Empregado** e **Gerente** (*depto* $\bullet$ *ger*). Uma definição formal de caminho é apresentada a seguir.

Definição 2.1 (*Caminho) Sejam* $\mathbf{C_1}, \mathbf{C_2}, \ldots \mathbf{C_{n+1}}$ *classes de um esquema tais que existe uma ligação* l_i *de* $\mathbf{C_i}$ *para* $\mathbf{C_{i+1}}$ *(*$l_i : \mathbf{C_i} \rightarrow \mathbf{C_{i+1}}$*),* $1 \leq i \leq n$*. Assim sendo,* $\delta = l_1 \bullet l_2 \bullet ... \bullet l_n$ *é um caminho de* $\mathbf{C_1}$*. Isso significa que as instâncias de* $\mathbf{C_1}$ *estão relacionadas com as instâncias de* $\mathbf{C_{n+1}}$ *através do caminho* δ*. Dizemos que* $\mathbf{C_1}$ *está relacionada com* $\mathbf{C_{n+1}}$ *através do caminho* δ*.*

A multiplicidade de um caminho depende das multiplicidades especificadas para as ligações que o compõe. Um caminho é *monovalorado* se todas as ligações que o compõem são monovaloradas. Um caminho é *multivalorado* se pelo menos uma das ligações que o compõe é multivalorada. A seguir definimos formalmente o valor de um caminho.

Definição 2.2 (*Valor de caminho) Sejam* $\mathbf{C_1}, \mathbf{C_2}, \ldots \mathbf{C_{n+1}}$ *classes de um esquema tais que* $\mathbf{C_1}$ *está relacionada com* $\mathbf{C_{n+1}}$ *através do caminho* $\delta = l_1 \bullet l_2 \bullet ... \bullet l_n$*, onde* l_i *é uma ligação de* $\mathbf{C_i}$ *para* $\mathbf{C_{i+1}}$ *(*$l_i : \mathbf{C_i} \rightarrow \mathbf{C_{i+1}}$*),* $1 \leq i \leq n$*. Seja* $\mathbf{o_1}$ *um instância de* $\mathbf{C_1}$*.*

- δ **é monovalorado***:* $\mathbf{o_1} \bullet \delta$ *retorna uma instância* $\mathbf{o_{n+1}}$ *de* $\mathbf{C_{n+1}}$*, tal que existem* $\mathbf{o_2}$ *instância de* $\mathbf{C_2}$*,* $\mathbf{o_3}$ *instância de* $\mathbf{C_3}$*, ...,* $\mathbf{o_n}$ *instância de* $\mathbf{C_n}$ *e* $\mathbf{o_i}.l_i = \mathbf{o_{i+1}}$*,* $1 \leq i \leq n$*.*
- δ **é multivalorado***:* $\mathbf{o_1} \bullet \delta$ *retorna uma coleção de instâncias* $\mathbf{o_{n+1}}$ *de* $\mathbf{C_{n+1}}$*, tal que existem* $\mathbf{o_2}$ *instâncias de* $\mathbf{C_2}$*,* $\mathbf{o_3}$ *instâncias de* $\mathbf{C_3}$*, ...,* $\mathbf{o_n}$ *instâncias de* $\mathbf{C_n}$ *e* $\mathbf{o_i}.l_i = \mathbf{o_{i+1}}$*,* $1 \leq i \leq n$*.*

2.3 Restrições de Integridade

As restrições de integridade impõem restrições nos estados admissíveis de um banco de dados (instâncias do banco de dados). Um instância $\mathcal{D}$ do esquema de um banco de dados é consistente ou é uma instância válida, se e somente se para qualquer restrição de integridade σ, σ é válida em $\mathcal{D}$. Os estados permissíveis (instâncias) de um banco de dados são aqueles que satisfazem sua estrutura e todas as suas restrições. É responsabilidade do projetista identificar essas restrições durante o projeto do banco de dados.

Os construtores básicos de associação, atributo e generalização especificam restrições importantes, mas não conseguem capturar todas as restrições do diagrama de classes. A UML não define uma sintaxe explícita para descrever as restrições, por esse motivo,

além das restrições presentes na própria estrutura do esquema, tais como, restrições de multiplicidade e navegabilidade, nesta seção definimos uma notação para representar os seguintes tipos de restrições de integridade: restrição de chave, dependência funcional, dependência existencial e assertivas de correspondência.

2.3.1 Restrição de Chave

Uma restrição de chave especifica a unicidade nos valores dos atributos de uma classe. Uma classe usualmente possui um atributo cujos valores são distintos para cada instância da classe. Tal atributo é chamado de atributo chave e seus valores podem ser usados para identificar cada instância da classe unicamente. Esse tipo de identificação corresponde ao conceito de chave primária do modelo ER. Por exemplo, na figura 2.1, o atributo **número** é uma chave da classe **Pedido**, porque não é permitido que dois pedidos possuam um mesmo número. A seguir definimos formalmente a restrição de chave.

Definição 2.3 *Seja* $\mathbf{C}$ *uma classe e* $\mathbf{a_1}$, $\mathbf{a_2}$, ..., $\mathbf{a_n}$, *atributos de* $\mathbf{C}$. *A restrição de chave* $\mathbf{a_1}$, $\mathbf{a_2}$, ..., $\mathbf{a_n} \in$ *Chaves(*$\mathbf{C}$*) especifica que para quaisquer instâncias* $\mathbf{c_1}$ *e* $\mathbf{c_2}$ *de* $\mathbf{C}$, *se para todo* $1 \leq i \leq n$, $\mathbf{c_1}.\mathbf{a_i} = \mathbf{c_2}.\mathbf{a_i}$, *então* $\mathbf{c_1} \equiv \mathbf{c_2}$.

2.3.2 Restrição de Dependência Funcional

Segundo [EN00], uma dependência funcional (FD) é uma restrição entre dois conjuntos de atributos de um banco de dados. Os projetistas usam seu conhecimento da semântica dos atributos - como eles estão relacionados uns com os outros - para especificarem as dependências funcionais.

Uma dependência funcional especifica uma restrição sobre os estados possíveis de um banco de dados. As extensões do banco de dados que satisfazem as restrições de dependência funcional são denominadas extensões legais. Portanto, no modelo relacional, o principal uso das dependências funcionais é descrever uma relação especificando restrições sobre seus atributos que devem sempre ser satisfeitas.

O diagrama de classes da UML é constituído de estruturas poderosas para a construção de bons esquemas conceituais, no entanto, não possui uma definição precisa para as restrições de dependência funcional desses esquemas. A seguir propomos uma notação para restrições de dependência funcional que, além de especificarem as restrições entre os atributos de uma classe (relacionamentos intra-classes), também consigam capturar os relacionamentos entre as classes (relacionamentos inter-classes). Essas restrições serão utilizadas no capítulo 4 na reestruturação dos esquemas conceituais.

Uma dependência funcional entre dois conjuntos de propriedades ($\mathcal{X}$ e $\mathcal{Y}$) de uma mesma classe $\mathbf{C}$, denotada por $\mathcal{X} \rightarrow \mathcal{Y}$, (ler-se $\mathcal{X}$ determina funcionalmente $\mathcal{Y}$) especifica que sempre que dois objetos de $\mathbf{C}$ concordam no valor de $\mathcal{X}$, eles devem necessariamente concordar no valor de $\mathcal{Y}$. Por exemplo, suponha o esquema mostrado na figura 2.1. A partir da semântica das propriedades da classe **Pedido**, temos que a restrição de dependência funcional **Pedido**[**número** $\rightarrow$ **data**, **preço**] é uma restrição da classe **Pedido** que especifica que sempre que dois objetos da classe **Pedido** concordarem no valor do atributo **número**, eles necessariamente devem concordar no valor dos atributos **data** e **preço**, ou seja, o número do pedido determina funcionalmente a data e o preço do pedido.

Definição 2.4 (*Dependência Funcional) Seja* $\mathbf{C}$ *uma classe de* $\mathcal{S}$ *e* $\mathbf{p_1}, \mathbf{p_2}, ..., \mathbf{p_n}$, $\mathbf{q}$ *propriedades de* $\mathbf{C}$. *A dependência funcional* $\mathbf{C}[\mathbf{p_1}, \mathbf{p_2}, ..., \mathbf{p_n} \rightarrow \mathbf{q}]$ *especifica que para quaisquer instâncias* $\mathbf{c_1}$ *e* $\mathbf{c_2}$ *de* $\mathbf{C}$, *se* $\mathbf{c_1}.\mathbf{p_i} = \mathbf{c_2}.\mathbf{p_i}$, $1 \leq i \leq n$, *então* $\mathbf{c_1}.\mathbf{q} \equiv \mathbf{c_2}.\mathbf{q}$.

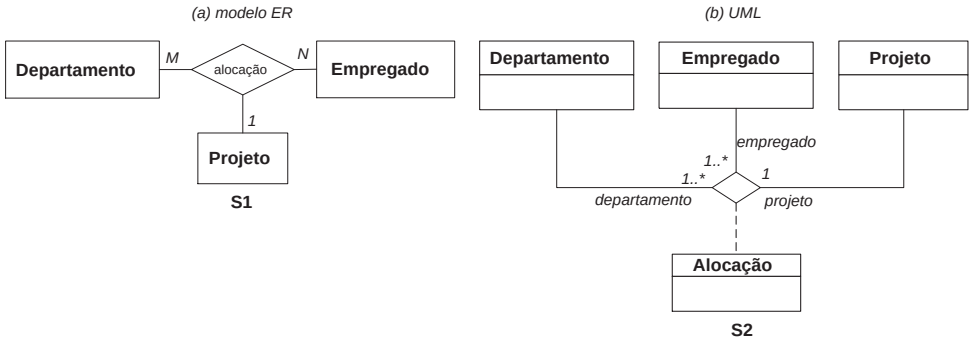

Figura 2.4: Exemplo - Dependência Funcional de Classes de Associação

Vale a pena ressaltar que no modelo entidade e relacionamento, as dependências funcionais podem ser derivadas diretamente das restrições de cardinalidade dos relacionamentos. Por exemplo, no esquema $\mathbf{S_1}$ da figura 2.4 a), a restrição de cardinalidade do relacionamento ternário mostrado é M:N:1. Isso implica que a cardinalidade da entidade **Projeto** é definida pelo par de instâncias das outras duas entidades (**Departamento** e **Empregado**) que estão associadas a uma única instância da entidade **Projeto**, ou seja **Departamento**, **Empregado** $\rightarrow$ **Projeto**. Na UML, as dependências funcionais são definidas explicitamente, independente das restrições estruturais existentes no esquema. Por exemplo, no esquema $\mathbf{S_2}$ da figura 2.4 b), considerando a semântica envolvida, temos que a dependência funcional **Alocação**[**projeto** $\rightarrow$ **departamento**, **empregado**] é uma restrição da classe de associação **Alocação** que deve ser satisfeita por todos os estados válidos do banco de dados.

2.3.3 Restrições de Dependência Existencial

As restrições de dependência existencial (DEs) são importantes para dar suporte à reestruturação de esquemas conceituais, pois permitem expressar formalmente a equivalência semântica de conceitos correspondentes representados de maneiras diferentes. A seguir definimos formalmente os vários tipos de dependência existencial.

Definição 2.5 (*DE de Subconjunto de Extensão) Sejam* $\mathbf{C_1}$ *e* $\mathbf{C_2}$ *classes,* $\mathbf{p_1}, \mathbf{p_2}, ..., \mathbf{p_n}$ *propriedades de* $\mathbf{C_1}$ *e* $\mathbf{q_1}, \mathbf{q_2}, ..., \mathbf{q_n}$ *propriedades de* $\mathbf{C_2}$. *A restrição de dependência existencial* $\mathbf{C_1}[\mathbf{p_1}, \mathbf{p_2}, ..., \mathbf{p_n}] \subset \mathbf{C_2}[\mathbf{q_1}, \mathbf{q_2}, ..., \mathbf{q_n}]$ *especifica que para qualquer instância* $\mathbf{c_1}$ *de* $\mathbf{C_1}$ *existe uma instância* $\mathbf{c_2}$ *de* $\mathbf{C_2}$ *tal que* $\mathbf{c_1.p_i} = \mathbf{c_2.q_i}$, $1 \leq i \leq n$.

Definição 2.6 (*DE de Equivalência de Extensão) Sejam* $\mathbf{C_1}$ *e* $\mathbf{C_2}$ *classes,* $\mathbf{p_1}, \mathbf{p_2}, \ldots, \mathbf{p_n}$ *propriedades de* $\mathbf{C_1}$ *e* $\mathbf{q_1}, \mathbf{q_2}, \ldots, \mathbf{q_n}$ *propriedades de* $\mathbf{C_2}$. *A restrição de dependência existencial* $\mathbf{C_1}\ [\mathbf{p_1}, \mathbf{p_2}, \ldots, \mathbf{p_n}] \equiv \mathbf{C_2}\ [\mathbf{q_1}, \mathbf{q_2}, \ldots, \mathbf{q_n}]$ *especifica que* $\mathbf{C_1}\ [\mathbf{p_1}, \mathbf{p_2}, \ldots, \mathbf{p_n}] \subset \mathbf{C_2}\ [\mathbf{q_1}, \mathbf{q_2}, \ldots, \mathbf{q_n}]$ *e* $\mathbf{C_2}\ [\mathbf{q_1}, \mathbf{q_2}, \ldots, \mathbf{q_n}] \subset \mathbf{C_1}\ [\mathbf{p_1}, \mathbf{p_2}, \ldots, \mathbf{p_n}]$.

Suponha o esquema **S** mostrado na figura 2.5.

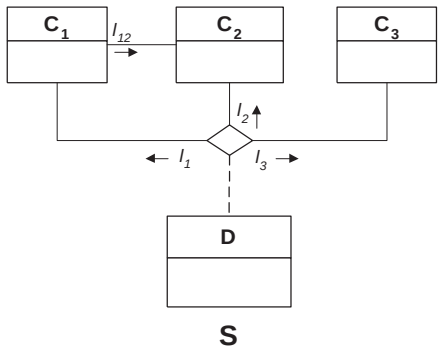

Figura 2.5: Dependência Existencial entre uma Classe de Associação e uma Associação

Definição 2.7 (*DE de Subconjunto entre uma Classe de Associação e uma Associação) Sejam* $\mathbf{d}$, $\mathbf{c_1}$ *e* $\mathbf{c_2}$ *instâncias das classes* $\mathbf{D}$, $\mathbf{C_1}$ *e* $\mathbf{C_2}$ *respectivamente. A restrição de dependência existencial de subconjunto* $\mathbf{C_1}.(\mathbf{D}.l_1)^{-1}.l_2) \subset \mathbf{C_1}.l_{12}$ *especifica que para todo* $\mathbf{c_2} \in \mathbf{c_1}.(\mathbf{d}.l_1)^{-1}.l_2$, *então* $\mathbf{c_2} \in \mathbf{c_1}.l_{12}$.

Definição 2.8 (*DE de Equivalência entre uma Classe de Associação e uma Associação) A restrição de dependência existencial de equivalência* $\mathbf{C_1}.(\mathbf{D}.l_1)^{-1}.l_2) \equiv \mathbf{C_1}.l_{12}$ *especifica que* $\mathbf{C_1}.(\mathbf{D}.l_1)^{-1}.l_2 \subset \mathbf{C_1}.l_{12}$ *e* $\mathbf{C_1}.l_{12} \subset \mathbf{C_1}.(\mathbf{D}.l_1)^{-1}.l_2$.

2.3.4 Assertivas de Correspondência

As *assertivas de correspondência* são tipos especiais de restrições de integridade usadas para especificar a correspondência entre componentes de esquemas. Segundo [Lós98], estas assertivas expressam que a semântica de alguns componentes em um esquema está de alguma forma relacionada com a semântica de alguns componentes em outro esquema.

Existem vários tipos de assertivas de correspondência, dependendo dos elementos envolvidos e da natureza da correspondência. No presente trabalho consideramos três grupos de assertivas de correspondência:

- Assertivas de correspondência de extensão
- Assertivas de correspondência de atributos
- Assertivas de correspondência de caminhos

Assertivas de Correspondência de Extensão

As assertivas de correspondência de extensão representam os diferentes tipos de equivalência semântica existentes entre as extensões das classes de um ou mais esquemas. A tabela 2.1 descreve as definições básicas de correspondências de extensão. Para cada tipo de assertiva é definida a notação utilizada para sua especificação e quais as condições que devem ser satisfeitas para sua validação em um estado específico $\mathcal{D}$ do banco de dados.

As assertivas de correspondência de extensão particionam as classes de um esquema em classes de equivalência. As classes $\mathbf{C_1}$ e $\mathbf{C_2}$ estão na mesma classe de equivalência se existe uma função de mapeamento $\mathcal{F}$ um para um entre as instâncias de $\mathbf{C_1}$ e as instâncias de $\mathbf{C_2}$. Uma função de mapeamento define o conjunto de atributos que será utilizado para definir a correspondência entre objetos semanticamente equivalentes. Em geral, $\mathbf{C_1}$ e $\mathbf{C_2}$ têm um identificador comum, o qual é usado como função de mapeamento. Se uma instância $\mathbf{c_1}$ de $\mathbf{C_1}$ é mapeada na instância $\mathbf{c_2}$ de $\mathbf{C_2}$ ($\mathcal{F}(\mathbf{c_1})$=$\mathbf{c_2}$), então $\mathbf{c_1}$ e $\mathbf{c_2}$ são semanticamente equivalentes, ou seja, $\mathbf{c_1}$ e $\mathbf{c_2}$ referem-se ao mesmo objeto do mundo real. As classes $\mathbf{C_1}$ e $\mathbf{C_2}$ são semanticamente equivalentes quando suas instâncias forem semanticamente equivalentes.

Assertivas de Correspondência de Atributos

As assertivas de correspondência de atributos representam a equivalência semântica de atributos pertencentes a classes semanticamente equivalentes.

Assertivas de Correspondência de Extensão		
Relação	**Notação**	**Condição de validação em $\mathcal{D}$**
Subconjunto	$C_1 \subset C_2$	(i) $\mathcal{D}(C_1) \subset \mathcal{D}(C_2)$ ou (ii) Existe uma função injetiva $f : \mathcal{D}(C_1) \rightarrow \mathcal{D}(C_2)$
Equivalência	$C_1 \equiv C_2$	(i) $\mathcal{D}(C_1) \subset \mathcal{D}(C_2)$ ou (ii) Existe uma função bijetiva $f : \mathcal{D}(C_1) \rightarrow \mathcal{D}(C_2)$
Disjunção	$C_1 \mid C_2$	(i) $\mathcal{D}(C_1) \cup \mathcal{D}(C_2) = \emptyset$
Diferença	$C \equiv C_1 - C_2$	(i) $\mathcal{D}(C) = \mathcal{D}(C_1) - \mathcal{D}(C_2)$ ou (ii) Existe uma função bijetiva $f : \mathcal{D}(C) \rightarrow \mathcal{D}(C_1) - \mathcal{D}(C_2)$
União	$C_i \equiv \cup_{i=1}^{n} C_i$	(i) $\mathcal{D}(C_i) = \cup_{i=1}^{n} \mathcal{D}(C_i)$ ou (ii) Existe uma função bijetiva $f : \mathcal{D}(C) \rightarrow \cup_{i=1}^{n} \mathcal{D}(C_i)$
Interseção	$C_i \equiv \cap_{i=1}^{n} C_i$	(i) $\mathcal{D}(C_i) = \cap_{i=1}^{n} \mathcal{D}(C_i)$ ou (ii) Existe uma função bijetiva $f : \mathcal{D}(C) \rightarrow \cap_{i=1}^{n} \mathcal{D}(C_i)$
Seleção	$C_1 \equiv C_2[\mathsf{p}]$	(i) $\mathcal{D}(C_1) = \mathcal{D}(C_2[\mathsf{p}])$ ou (ii) Existe uma função bijetiva $f : \mathcal{D}(C_1) \rightarrow \mathcal{D}(C_2[\mathsf{p}])$
Sobreposição	$C_1 \cap C_2$	(i) C_1 e C_2 podem ter instâncias em comum ou (ii) Existe uma função injetiva parcial $f : \mathcal{D}(C_1) \rightarrow \mathcal{D}(C_2)$
Generalização	$C \equiv Gen(C_1, ..., C_n)$	(i) $C = \cap_{i=1}^{n} C_i (C \equiv \cap_{i=1}^{n} C_i$ ou (ii) $C = C_i \cup C_j = \emptyset$, para $i \neq j (C_i \mid C_j)$

Tabela 2.1: Assertivas de Correspondência de Extensão

Definição 2.9 (*Atributos com tipos compatíveis) Seja* $\mathbf{a_1}$ *um atributo de* $\mathbf{C_1}$ *e* $\mathbf{a_2}$ *um atributo de* $\mathbf{C_2}$*, onde* $\mathbf{C_1}$ *e* $\mathbf{C_2}$ *são classes semanticamente equivalentes. A assertiva de correspondência de atributos* $\mathbf{a_1} \equiv \mathbf{a_2}$ *especifica que para quaisquer instâncias* $\mathbf{c_1} de \mathbf{C_1}$ *e* $\mathbf{c_2} de \mathbf{C_2}$*, se* $\mathbf{c_1} \equiv \mathbf{c_2}$ *então* $\mathbf{c_1}.\mathbf{a_1} = \mathbf{c_2}.\mathbf{a_2}$.

Deve-se também considerar os atributos com domínios diferentes. Nesse caso, existe uma função de mapeamento de domínio (γ), tal que a restrição de equivalência $\mathbf{a_1} \equiv \mathbf{a_2}$ especifica que para quaisquer instâncias $\mathbf{c_1}$ de $\mathbf{C_1}$ e $\mathbf{c_2}$ de $\mathbf{C_2}$, se $\mathbf{c_1} \equiv \mathbf{c_2}$ então $\mathbf{c_1}.\mathbf{a_1} = \mathbf{c_2}.(\mathbf{a_2} \circ \gamma)$.

Assertivas de Correspondência de Caminhos

Durante a análise dos inter-relacionamentos dos esquemas também devemos considerar a equivalência semântica dos caminhos. As assertivas de correspondência de caminhos especificam que caminhos de classes semanticamente equivalentes são semanticamente equivalentes. A seguir definimos formalmente as assertivas de correspondência de caminhos.

Sejam $\Gamma_1 = l_1 \bullet ... \bullet l_n$ e $\Gamma_2 = l'_1 \bullet ... \bullet l'_n$ caminhos, onde $l_i : \mathbf{C_i} \rightarrow \mathbf{C_{i+1}}$ e $l'_i : \mathbf{C'_i} \rightarrow \mathbf{C'_{i+1}}$, para $1 \leq i \leq n$. $\mathbf{C_1}$ e $\mathbf{C'_1}$ são semanticamente equivalentes, assim como $\mathbf{C_{n+1}}$ e $\mathbf{C'_{n+1}}$.

Definição 2.10 (*Assertiva de correspondência de equivalência de caminhos)* Γ_1 *e* Γ_2 *são monovalorados ou multivalorados: a assertiva de correspondência de equivalência de caminhos* $\Gamma_1 \equiv \Gamma_2$ *especifica que para quaisquer instâncias* $\mathbf{c_1}$ *de* $\mathbf{C_1}$ *e* $\mathbf{c'_1}$ *de* $\mathbf{C'_1}$, *se* $\mathbf{c_1} \equiv \mathbf{c'_1}$ *então* $\mathbf{c_1}.\Gamma_1 \equiv \mathbf{c'_1}.\Gamma_2$.

Definição 2.11 (*Assertiva de correspondência de subconjunto de caminhos)* Γ_1 *e* Γ_2 *são multivalorados: A assertiva de correspondência de subconjunto de caminhos* $\Gamma_1 \subset \Gamma_2$ *especifica que para quaisquer instâncias* $\mathbf{c_1}$ *de* $\mathbf{C_1}$ *e* $\mathbf{c'_1}$ *de* $\mathbf{C'_1}$, *se* $\mathbf{c_1} \equiv \mathbf{c'_1}$ *então* $\mathbf{c_1}.\Gamma_1 \subset \mathbf{c'_1}.\Gamma_2$.

Vale a pena ressaltar que um atributo é um caminho constituído de uma única ligação. Por exemplo, suponha os esquemas $\mathbf{S_1}$ e $\mathbf{S_2}$ mostrados na figura 2.6. A assertiva de correspondência de equivalência de caminhos **Empregado** • **gerente** ≡ **Empregado** • *pertence* • *é-gerenciado* • **nome**, especifica que o atributo **gerente** da classe **Empregado** no esquema $\mathbf{S_1}$ é equivalente ao nome do gerente do departamento daquele empregado no esquema $\mathbf{S_2}$. Portanto, o atributo **nome** também é um caminho.

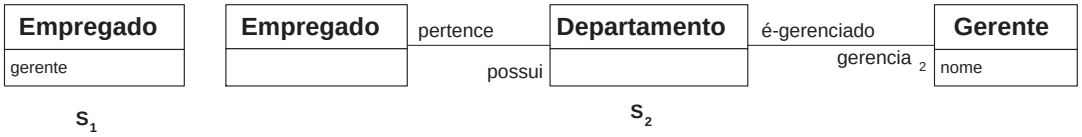

Figura 2.6: Caminhos semanticamente equivalentes

Até agora, vimos os componentes mais relevantes da modelagem conceitual dos dados com a UML. No capítulo 4, utilizaremos os conceitos discutidos aqui na formalização dos padrões desenvolvidos em nosso trabalho para auxiliar nas atividades de integração de visões com a UML.

Capítulo 3

Integração de Visões e Trabalhos Relacionados

3.1 Introdução

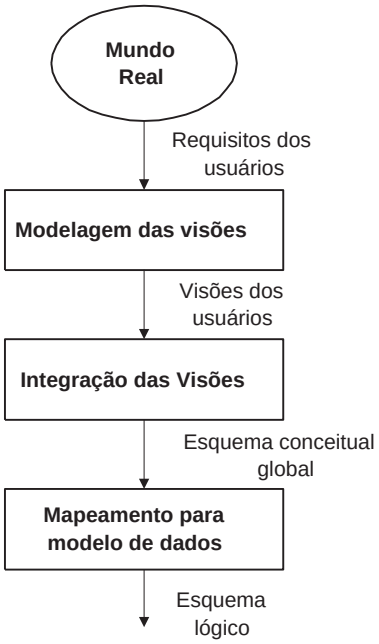

Figura 3.1: Projeto de Banco de Dados Através da Integração de Visões

O projeto de banco de dados pode ser descrito como a atividade de construir uma representação do domínio de uma aplicação em um determinado formalismo. Esta representação ou modelo precisa capturar os objetos que estão envolvidos no domínio da

aplicação e as operações que, de alguma forma, afetam os objetos. Como ilustrado na figura 3.1, durante o projeto do banco de dados, primeiramente cada grupo de usuários analisa seus requisitos e especifica suas visões dos dados usando um modelo semântico. As visões ou esquemas externos permitem o usuário ignorar os dados que não são relevantes para sua aplicação. As visões, além de proteger o acesso aos dados, ajudam a alcançar um certo grau de independência lógica, uma vez que é possível alterar o esquema do banco de dados sem alterar uma visão. Em seguida as várias visões dos usuários são integradas em um esquema conceitual global que satisfaz os requisitos de toda a organização. Esse processo é denominado integração das visões. A integração de visões refere-se à atividade de construir uma estrutura global (esquema integrado) a partir de esquemas individuais (visões).

Segundo [SNL86], dois motivos principais justificam a necessidade de integração de visões durante o projeto de um banco de dados:

1. a estrutura de um banco de dados para grandes aplicações é bastante complexa de ser modelada por um único projetista em uma única visão;

2. em geral, os grupos de usuários trabalham de forma independente nas organizações e possuem seus próprios requisitos dos dados, que podem conflitar com os interesses de outros grupos.

Na seção 3.1, analisamos os principais modelos semânticos usados para a modelagem conceitual dos dados, enfatizando os pontos fortes e pontos fracos de cada um dos modelos apresentados. Na seção 3.2, apresentamos resumidamente algumas metodologias de integração de visões. Na seção 3.3, discutimos outros trabalhos relacionados. Na seção 3.4 apresentamos a metodologia proposta.

3.2 Modelos Semânticos para Modelagem Conceitual dos Dados

A modelagem conceitual é o processo de construir um esquema conceitual utilizando para tal, um modelo de dados de alto nível (modelo semântico). O esquema conceitual descreve a realidade associada a um grupo de usuários e fornece ao projetista uma representação global do problema modelado, independente de aspectos específicos de implementação do sistema de gerenciamento de banco de dados escolhido. Navathe e Schkolnick [NS78] definem formalmente a modelagem conceitual como a modelagem do uso e da estrutura da

informação do mundo real a partir do ponto de vista de diferentes usuários e/ou aplicações (visões).

Qualquer metodologia para integração de visões deve ser baseada em algum modelo de dados (modelo semântico). Um modelo de dados é a base para especificarmos as visões e os relacionamentos semânticos entre os conceitos nas diferentes visões. Neste trabalho, nós adotamos o modelo de objetos da UML descrito no capítulo 2. Nesta seção, nós descrevemos os modelos semânticos mais utilizados no processo de integração de visões: Modelo de Dados Entidade e Relacionamento (modelo ER) de [Che76], Modelo de Abstração dos Dados de Smith e Smith [SS77] e Modelo Navathe e Schkolnick (modelo NS) [NS78].

3.2.1 Modelo Entidade e Relacionamento (Modelo ER)

O modelo ER de [Che76] é provavelmente a abordagem mais amplamente usada para modelagem conceitual dos dados. É um modelo simples e fácil de conceitualizar, empregando apenas três construtores básicos - entidades, atributos e relacionamentos. Uma entidade é *algo* de interesse no banco de dados que pode ser claramente definido [SG88]. Por exemplo, no domínio de uma universidade, uma entidade poderia ser **Estudante**. Um relacionamento é uma associação entre entidades. Por exemplo, **Estudante** *faz* **Curso** é um relacionamento entre as entidades **Estudante** e **Curso**. Atributos são propriedades ou características que podem ser identificadas para entidades e relacionamentos. Por exemplo, no esquema $\mathbf{S_1}$ mostrado na figura 3.2, *num-estudante* é um atributo da entidade **Estudante** e *histórico* um atributo do relacionamento **Estudante** *faz* **Curso**.

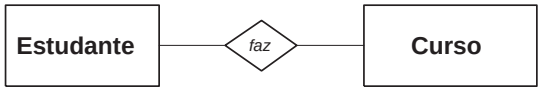

Figura 3.2: Exemplo - Esquema no Modelo ER

No modelo ER, a semântica é capturada através dos conceitos de conjunto de entidades e conjunto de relacionamentos. Ser membro de um desses conjuntos indica que a entidade ou o relacionamento satisfaz certas condições que são características do conjunto.

Existem, contudo, situações onde o modelo ER falha ao capturar a semântica. Temos observado que a atual modelagem conceitual através do ER está voltada principalmente para o conceito de entidades. Tal fato torna-se evidente pela existência de vários construtores especiais para entidades tais como, entidades fracas, entidades compostas e hierarquias de generalização. Os relacionamentos não têm recebido o mesmo grau de atenção.

O modelo não permite a representação de relacionamentos entre relacionamentos ou entre uma entidade e um relacionamento e também não permite que distinções sejam feitas entre diferentes tipos de relacionamento. Relacionamentos ternários e de grau superior, também têm recebido uma análise pouco rigorosa. A justificativa dada para tal fato é que relacionamentos de grau alto são raros nas aplicações do mundo real e, quando ocorrem, podem ser representados por múltiplos relacionamentos binários, o que nem sempre é verdade. A escolha entre um relacionamento n-ário e vários relacionamentos binários é uma tarefa complexa e depende essencialmente da semântica da aplicação a ser modelada. Além disso, quando o modelo ER é utilizado, é requerida uma identificação e distinção bem definida dos principais construtores. Existem, contudo, situações em que pode ser difícil determinar se um objeto do mundo real deve ser representado como uma entidade, como um relacionamento, ou como um atributo de uma entidade ou de um relacionamento. Por exemplo, *casamento* deve ser representado como uma entidade ou como um relacionamento entre duas entidades **Pessoa**?

3.2.2 Modelo de Abstração dos dados

No modelo de abstração dos dados de Smith e Smith, tanto entidades como relacionamentos são representados como objetos. Dois tipos diferentes de relacionamentos são representados - associação de entidades similares e associação de entidades relacionadas, conhecidos respectivamente como generalização e agregação. Ambos são usados para construir uma hierarquia multi-dimensional de objetos que representam a visão dos usuários.

Em geral, uma abstração é um modelo de um sistema no qual alguns detalhes são propositadamente omitidos, visando focalizar a atenção nas características mais significantes. Em uma abstração de agregação, o relacionamento entre objetos é feito em um nível mais alto. Por exemplo, **Reserva** é uma abstração de um relacionamento particular entre uma pessoa, um hotel e uma data. Podemos pensar na reserva sem ter que nos preocupar com todos os detalhes subtendidos, tal como a duração da estadia.

A generalização toma uma classe de entidades similares e a considera como um objeto genérico. Por exemplo, um conjunto de pessoas que dirigem caminhão pode ser imaginado como um objeto genérico **Caminhoneiro**. Esse novo objeto torna-se então uma primitiva para definir modelos do mundo real. Caso seja necessário modelar o objeto abstrato **Empregado**, este deve ser inicialmente decomposto junto ao plano de generalização dos objetos genéricos - por exemplo, **Caminhoneiro**, **Secretária** e **Engenheiro**, como mostrado no esquema $\mathbf{S_2}$ da figura 3.3. O objeto **Empregado** deve então ser decomposto junto ao plano de agregação para incluir os atributos de empregado, tais como, número

de identificação do empregado, nome e idade. Assim, o objeto abstrato **Empregado** é a interseção de uma hierarquia de generalização e agregação.

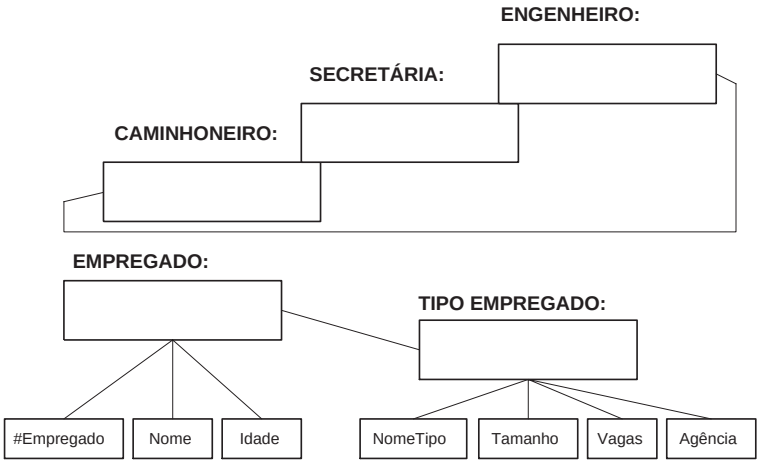

Figura 3.3: Exemplo - Esquema no Modelo de Abstração dos Dados

Essa abordagem da abstração dos dados tem sido descrita como elegante conceitualmente, contudo, tem sido criticada quando envolve problemas de grande escala, pois exige uma grande habilidade por parte dos projetistas durante a construção do esquema conceitual.

3.2.3 Modelo Navathe e Schkolnick (Modelo NS)

Navathe e Schkolnick aprofundaram os conceitos da metodologia de abstração dos dados para desenvolverem seu próprio modelo. O objetivo do modelo NS é capturar uma grande quantidade de conteúdo semântico e modelar os requisitos dos usuários da forma mais explícita possível. O modelo NS possui dois tipos de construtores - objetos e conectores - com tipos objeto sendo divididos em tipos entidade e associação.

Associações são relacionamentos n-ários entre entidades, entre entidades e associações, ou entre associações. Elas representam fatos, idéias, ou aspectos específicos de um relacionamento.

Um esquema no modelo NS é representado como um grafo que possui entidades e associações em seus nós e arestas ligando os nós. As arestas são chamadas conectores e são representadas como tuplas com uma seta sobre um dos membros da tupla indicando a direção da associação. Por exemplo, *Empregado Trabalha Função* poderia ser

representado como três nós, conforme mostrado no esquema $\mathbf{S_3}$ da figura 3.4. O conector entre **Empregado** e **Trabalha** poderia ser escrito como (**Empregado**, **Trabalha**) com uma seta sobre **Empregado** para representar a direção do conector. Similarmente, um conector entre **Trabalha** e **Função** poderia ser (**Trabalha**, **Função**) com uma seta sobre **Trabalha**, indicando que a direção da associação é de **Trabalha** para **Função**. Uma associação é descrita por uma tupla que contém os nomes das entidades e atributos participantes tais como **Trabalha** (**Empregado**, **Função**).

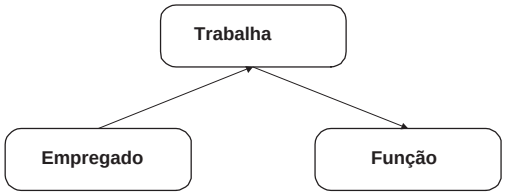

Figura 3.4: Exemplo - Esquema no Modelo NS

Entidades e associações possuem conjuntos descritores correspondentes. Conjuntos descritores são listas de atributos, ou grupos de atributos, no caso de atributos repetidos, que podem ser associados a uma instância de um objeto no banco de dados. Por exemplo, o objeto **Empregado** no Esquema $\mathbf{S_3}$, poderia possuir como conjunto descritor **Empregado** = {EMPREGADO#, Dep#, {Escola, {Grau de instrução}}}. Para o objeto **Função**, o conjunto descritor correspondente seria **Função** = {FUNÇÃO#, Nome-Função}.

Toda entidade deve ter um identificador. No modelo NS, o tipo do identificador é definido explicitamente. Pode ser interno ou externo, com um identificador interno total ou parcial. Quando um elemento ou vários elementos de um conjunto descritor são usados para identificar uma entidade, existe um identificador interno. Um identificador interno total provê uma completa identificação da entidade, contudo uma identificação interna parcial precisa de alguma identificação externa para identificar unicamente uma instância da entidade. As associações identificadas ajudam a estabelecer regras para inserir e remover instâncias de entidades e identificar associações. Isso aumenta a semântica do modelo.

As associações ainda são divididas em associações simples e identificadoras. Três tipos de associação simples - categorização, seleção e subconjunto - provêm informação semântica adicional.

A categorização é uma associação entre as entidades proprietária e membro. Está

fortemente relacionada ao conceito de generalização de Smith e Smith exceto pelo fato de eliminar a restrição de categorias mutualmente exclusivas. No exemplo de Navathe e Schkolnick, um estudante pode ser categorizado em graduação, graduado, especial, residente, etc. Os mapeamentos definidos pelo modelo NS dão a cada estudante um subconjunto próprio de categorias, tal que não têm que ser definidas várias categorias diferentes para assegurar a exclusão mútua. Categorização é usada num contexto semântico em conexão com atividades de inserção / remoção.

A seleção é uma associação binária entre entidades. Uma associação de seleção **A**(**X**,**Y**) provê uma associação onde uma instância de **X** está associada a um subconjunto de instâncias de **Y**. Por exemplo, seja **A**(**X**,**Y**) uma seleção de associação: **A**(**País**, **Estudante**), como mostrado no esquema $\mathbf{S_4}$ figura 3.5. Isso é interpretado como uma seleção de um estudante de um país onde país é o proprietário da seleção de associação.

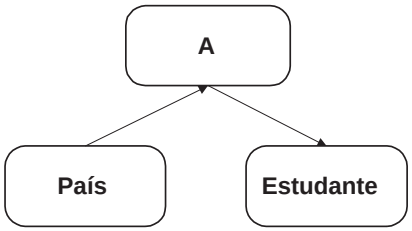

Figura 3.5: Exemplo - Seleção

A seleção difere de categorização em nível de instanciação. Se o exemplo **Estudante-País** fosse modelado usando categorização para cada estudante, uma instância adicional, tal como **Estudante-Canadense** ou **Estudante-Americano** seria requerido para cada categoria.

Em termos de conteúdo semântico, inserir um conteúdo semântico é inserir uma instância proprietária **X** para uma seleção **A** (**X**,**Y**) implica que a função de seleção correspondente deve ser modificada, instâncias de **Y** devem ser particionadas novamente, e uma nova instância de **A** criada se necessário.

O subconjunto é uma associação unária que provê um significado de associação do nome **A** a um subconjunto de instâncias de **B**. Suponha, por exemplo, o esquema mostrado na figura 3.6. **Orientador-de** é uma associação entre **Professores** e **Estudantes**, enquanto **Orientador-phd** é uma associação de subconjunto na qual **Orientador-de** é um componente. Subconjunto permite um subconjunto de instâncias de associação serem agregadas e nomeadas.

Algumas das vantagens do modelo NS são:

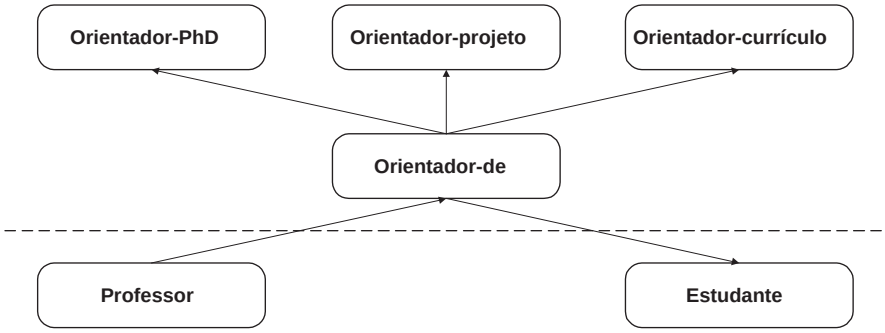

Figura 3.6: Exemplo - Subconjunto

- Tipos associação podem ser definidos sobre tipos entidade, tipos associação, ou uma combinação de ambos, o que o difere do modelo ER.
- Dependências existenciais entre entidades são modeladas separadamente dos relacionamentos entre entidades, diferentemente do método de abstração dos dados, logo, caminhos de identificação são definidos claramente.
- Os relacionamentos são modelados tanto em nível de esquema quanto em nível de instâncias, diferentemente do modelo ER e do método de abstração dos dados.

3.3 Metodologias para Integração de Visões

Uma das tarefas mais complexas do projeto de banco de dados é a análise e a integração dos requisitos dos vários grupos de usuários da aplicação. Uma vez que os requisitos de cada grupo de usuários tenham sido descritos formalmente pelos esquemas de visões, as metodologias de integração de visões são usadas para construir o esquema conceitual global do banco de dados, de forma que este consiga suportar as visões de todos os esses diferentes grupos [Sch87].

Segundo [SNL86], o processo de integração de visões pode ocorrer em diferentes momentos do projeto de um banco de dados. [Kah79] aplica a integração de visões entre o levantamento de requisitos e o projeto conceitual. Realizar a integração durante a fase de análise dos requisitos é uma tarefa árdua, pois os requisitos dos usuários estão geralmente mal estruturados dificultando o uso de uma metodologia formal. As metodologias de [AFS81], [CV83] e [Y$^+$82] realizam a integração de visões como fase do projeto lógico do banco de dados, na fase de implementação. Realizar a integração durante a fase de implementação é uma tarefa bastante complexa porque, nesse ponto, as representações

não permitem o uso efetivo de abstrações. Sendo assim, a integração dos esquemas deve ser realizada após a análise dos requisitos, mas antes da implementação. Portanto, a fase ideal para a integração é a fase do projeto conceitual, onde o uso da abstração é bastante útil na comparação e na integração das diferentes percepções do domínio da aplicação por diferentes grupos de usuários. Esta abstração não compromete o esquema integrado com nenhum modelo de dados lógico, deixando livre a implementação do banco de dados. As metodologias de [BL84], [ELN87], [NG82], [TF82] e [WE79] aplicam a integração de visões durante o projeto conceitual.

A seguir, apresentamos resumos dos principais trabalhos que abordam a integração de visões durante o projeto conceitual do banco de dados, enfatizando as características mais importantes e estratégias de integração utilizada nesses trabalhos [SNL86]. Os resumos abaixo ressaltam o fato de que, apesar de todas as metodologias descritas terem um mesmo objetivo, o processo de integração de visões pode variar bastante. A terminologia dos autores é utilizada sem modificações.

3.3.1 ElMasri et al. [1987]

A metodologia para integração de visões proposta em [ELN87], está baseada no Modelo Entidade-Categoria-Relacionamento (modelo ECR) que reconhece entre entidades e relacionamentos o conceito de categoria. As categorias são usadas para dois propósitos: (i) mostrar uma generalização de uma super-entidade e uma sub-entidade e (ii) permitir a definição de um subconjunto de uma entidade baseada em algum predicado. Como entrada, [ELN87] utiliza n esquemas que representam as visões dos usuários representadas no modelo ERC, assertivas de equivalência de atributos e assertivas de extensão de classes objeto e produz como resultado o esquema integrado da aplicação, junto a um conjunto de mapeamentos entre os esquemas.

Estratégia de Integração

1. Se necessário, transformar os esquemas existentes em esquemas ERC;
2. Efetuar a pré-integração, que consiste em uma análise e modificação intermediária dos esquemas;
3. Integrar as classes de objeto;
4. Integrar as classes de relacionamento;
5. Gerar os mapeamentos.

3.3.2 Batini e Lenzerini [1984]

A metodologia para integração de visões proposta em [BL84], está baseada no Modelo Entidade e Relacionamento (modelo ER). O modelo está descrito brevemente na seção 2.3. Como entrada, [BL84] utiliza os esquemas dos usuários (esquemas componentes), o esquema da organização e os pesos dos esquemas e produz como resultado o esquema global ou esquema integrado da aplicação.

Estratégia de Integração

1. Escolha o esquema base
2. Enquanto houver novos esquemas para serem integrados, faça:
   (a) escolha um novo esquema;
   (b) encontre conflitos entre os dois esquemas;
   (c) deixe os esquemas em conformidade;
   (d) efetue a fusão dos esquemas;
   (e) analise o diagrama do esquema integrado com o objetivo de descobrir redundâncias e simplificar a representação.

3.3.3 Navathe e Gadgil [1982]

A metodologia para integração de visões proposta em [NG82], está baseada no Modelo Navathe e Schkolnick (modelo NS) [NS78]. O modelo NS está descrito brevemente na seção 2.3. Como entrada, [NG82] utiliza a visão da organização, as visões locais, as assertivas inter-visões e intra-visões e os requisitos de processamento. Como resultado, produz uma visão global junto às regras de mapeamento, conflitos não resolvidos e assertivas modificadas.

Estratégia de Integração

1. Dividir as visões em classes de visões equivalentes, visões idênticas e visões simples;
2. Integrar as classes verificando conflitos e solucionando esses conflitos, baseado nas assertivas ou ordem de preferência;
3. Executar novas integrações entre visões intermediárias, enquanto houver novas operações de assertivas de visões;

4. Gerar regras de mapeamento que determinam como cada uma das visões componentes pode ser obtida a partir da visão integrada.

3.3.4 Teorey e Fry [1982]

A metodologia para integração de visões proposta em [TF82], está baseada no Modelo Hierárquico Semântico. Os principais construtores desse modelo são as classes de objeto, abstrações de agregação entre objetos pelos quais um objeto é visto como uma abstração de um conjunto de propriedades e abstrações de generalização. Como entrada, [TF82] utiliza as perspectivas de informação (esquemas componentes), aplicação, evento, corporação e regras de uso. Como resultado, produz uma estrutura de informação global (esquema integrado) e os conflitos.

Estratégia de Integração

1. Ordenar as visões locais de acordo com a importância específica para os objetivos do projeto;
2. Consolidar as visões locais duas a duas (a ordem é determinada pelo passo 2);
3. Resolver conflitos que surjam no passo 2.

3.3.5 Wiederhold e ElMasri [1979]

A metodologia para integração de visões proposta em [WE79], está baseada no Modelo Estrutural. Esse modelo é construído a partir de relações que são usadas para representar classes de entidade e vários tipos de relacionamentos entre classes de entidade. Outros tipos de relacionamentos são representados por conexões entre relações. Como entrada, [WE79] utiliza dois modelos de dados (esquemas componentes). Como resultado, produz o chamado modelo do banco de dados integrado (esquema integrado) e os sub-modelos do banco de dados.

Estratégia de Integração

1. Encontre todos os pares de relações compatíveis;
2. Para cada par de relação, integre a conexão entre eles;
3. Integre as relações compatíveis.

3.4 Outros Trabalhos Relacionados

Nos últimos anos, temos observado que o enfoque do projeto de banco de dados saiu do domínio lógico para o nível conceitual. Desde então, a modelagem conceitual tornou-se a fase de maior importância do processo de projeto de um banco de dados. Os projetistas perceberam que erros introduzidos no esquema conceitual podem se estender para as próximas fases do projeto, tornando-se cada vez mais caros e difíceis de serem removidos. Além disso, sabemos que a fase ideal para a integração é a fase do projeto conceitual, onde o uso da abstração é bastante útil na comparação e na integração das diferentes percepções do domínio da aplicação por diferentes grupos de usuários. Esta abstração não compromete o esquema integrado com nenhum modelo de dados lógico, deixando livre a implementação do banco de dados. Por este motivo, a construção de esquemas conceituais bons e corretos tem sido alvo de diversos trabalhos, destacando-se dentre eles: [Sto88], [Sto91], [DSB97], [Lar97], [Fow97a], [WSW99] e [Vid94], os quais são descritos a seguir.

Segundo [Sto88], um elemento crítico do projeto de um banco de dados é assegurar que os dados necessários para os usuários estejam armazenados no banco de dados e que estes sejam facilmente acessados. Assim, um componente chave do processo de projeto de um banco de dados está em obter dos usuários uma definição clara dos requisitos dos dados. O *View Creation System* descrito em [Sto88] é um sistema especialista desenvolvido com o propósito de extrair os requisitos dos usuários de forma precisa. O sistema então formaliza e automatiza o processo de criação das visões, aumentando a qualidade e consistência do projeto de banco de dados. O sistema também formaliza, como um conjunto de regras, uma metodologia para gerar as visões dos usuários. Essas regras foram implementadas como a base de conhecimento do *View Creation System*.

[Sto91] descreve uma metodologia para projeto de banco de dados relacional baseado no modelo de dados Entidade e Relacionamento (modelo ER). A metodologia proposta começa com a identificação dos construtores básicos do modelo ER. Também são discutidas formas de capturar a semântica da aplicação através da abstração dos dados e de procedimentos para detectar problemas de projeto. À medida que percorre as várias fases da metodologia, [Sto91] descreve detalhadamente os construtores do modelo de ER. Além disso, propõe regras de projeto com o objetivo de ajudar os projetistas a usar corretamente esses construtores. O processo é completado com o mapeamento do modelo ER em um conjunto de relações já normalizadas.

[DSB97] propõe algumas regras de projeto voltadas para representação e análise dos relacionamentos no enfoque do modelo ER. [DSB97] acredita que a ausência de uma

análise mais expressiva dos relacionamentos e a pouca familiaridade dos projetistas com esse construtor sejam responsáveis pela presença de grande parte dos erros encontrados nos esquemas conceituais. Nesse esforço, [DSB97] desenvolve um tratamento uniforme para todos os tipos de relacionamentos, reduzindo significantemente a ocorrência de tais erros, aumentando assim, a capacidade de se extrair informações do mundo real e de se obter melhores projetos conceituais. Dentre as principais contribuições deste estudo destaca-se o conjunto de regras de projeto para identificação e representação dos relacionamentos no modelo ER. As regras propostas estão definidas formalmente, de modo a serem incorporadas à base conhecimento do *View Creation System*, um sistema especialista para projeto de banco de dados automatizado proposto anteriormente em [SG88].

Existem muitos livros no mercado que tratam da análise e projeto orientado a objetos. No entanto, grande parte desses livros se concentra em apresentar apenas a notação ou o processo de modelagem orientado a objeto. [Lar97] e [Fow97a] apresentam abordagens diferentes para o assunto. Em seu livro, [Lar97] descreve uma metodologia orientada a objeto para a construção de aplicações robustas e fáceis de manter. A metodologia descrita propõe uma seqüência de passos a serem seguidos, partindo dos requisitos do domínio do problema modelado, passando pela análise e projeto, e terminando na fase de implementação do sistema. Para ilustrar as fases de análise e projeto, [Lar97] utiliza a notação fornecida pela UML. [Lar97] finaliza mostrando como refinar os projetos desenvolvidos através dos padrões de projeto propostas em [**?**]. Já [Fow97a], focaliza os resultados obtidos dos processos de modelagem. Seu livro discute padrões que refletem estruturas conceituais inseridas em domínios específicos. Grande parte de seu livro discute padrões relacionados a determinadas áreas de negócios, como por exemplo, financeira, saúde, etc. Segundo [Fow97a], esses padrões são importantes porque eles podem ajudar os projetistas a entenderem como as pessoas vêem o mundo. Além disso, os padrões sugeridos podem ser usados para traduzir os modelos conceituais em programas, pois também discutem aspectos específicos de implementação.

[WSW99] apresenta uma definição formal do significado dos construtores da modelagem conceitual no contexto da ontologia. A ontologia é uma área da filosofia que trata com modelos de realidade. Baseada neste estudo, [WSW99] deriva várias regras de modelagem conceitual que têm como objetivo assistir os projetistas no emprego dos relacionamentos durante a modelagem conceitual utilizando o modelo ER. Além disso, [WSW99] discute como as regras propostas resolvem problemas relacionados às ambigüidades existentes atualmente no modelo ER e como elas podem enriquecer a capacidade do modelo ER extrair as informações relevantes do domínio das aplicações modeladas.

Em [Vid94], é proposta um metodologia para integração de visões durante o projeto

conceitual de um banco de dados. Nessa metodologia, o processo de integração de visões é dividido basicamente em três passos: combinação, reestruturação e otimização. Durante a combinação, as várias visões dos usuários são analisadas e comparadas para determinar correspondência entre conceitos, descobrir relacionamentos e detectar possíveis conflitos. No passo de reestruturação, as visões são compatibilizadas de forma que a fusão das partes comuns seja possível e durante a otimização, o tamanho do esquema é reduzido através da fusão de partes comuns. Além disso, [Vid94] define um conjunto de primitivas de transformação que permitem que a integração seja realizada de forma segura e eficiente.

3.5 Metodologia Proposta

Nesta seção, descrevemos nosso enfoque para tratar do processo de integração de visões dos usuários durante o projeto do banco de dados. Na seção 3.5.1, apresentamos as características gerais da metodologia adotada. Na seção 3.5.2, detalhamos o processo de integração de visões proposto.

3.5.1 Características Gerais da Metodologia Proposta

No nosso trabalho, o problema da integração das visões é tratado durante a fase do projeto conceitual. Nesta seção, descrevemos a abordagem proposta para o processo de integração de visões durante o projeto de um banco de dados. Como mostrado na figura 3.7, o enfoque utilizado para o processo de integração de visões consiste dos passos abaixo:

(i) Primeiramente, cada grupo de usuários analisa seus requisitos de dados e especifica suas visões dos dados como um esquema.

(ii) Em seguida, as visões individuais são combinadas em um esquema integrado e a definição conceitual das visões é efetuada. O esquema integrado descreve todos os dados que serão armazenados no banco de dados. A definição conceitual consiste dos esquemas das visões originais e de um conjunto de assertivas de correspondência que especifica como cada visão é definida em termos do esquema conceitual integrado obtido. Usamos as assertivas de correspondência para especificar o relacionamento das visões dos usuários com o esquema obtido. É importante notar que o relacionamento entre cada visão e o esquema conceitual reflete exatamente os relacionamentos entre as visões como definido no modelo conceitual. Esse passo é bastante complexo, devido às diversas representações utilizadas pelos projetistas na modelagem das visões. Essa diversidade se deve ao fato de que cada grupo de usuários possui

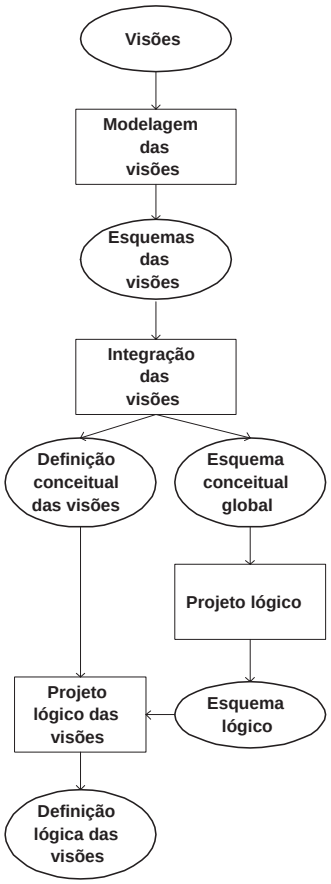

Figura 3.7: Projeto de Banco de Dados Através da Integração de Visões

seus próprios requisitos de dados, resultando em diferentes percepções da estrutura dos dados.

(iii) Após a integração das visões, o esquema integrado é mapeado para um esquema lógico. O esquema lógico é expresso em termos do modelo de dados específico do sistema de gerenciamento de banco de dados adotado.

(iv) Após o mapeamento do esquema conceitual no esquema lógico, as definições conceituais das visões são traduzidas em definições em termos do esquema lógico obtido. As assertivas de correspondência também são mapeadas do esquema conceitual para o esquema lógico. Esse passo é abordado em [Nor01]. [Nor01] realiza o mapeamento do esquema integrado resultante do nosso processo de integração para o modelo objeto-relacional.

3.5.2 O Processo de Integração de Visões

Nesta seção detalhamos o processo de integração de visões proposto, que constitui uma das principais contribuições do nosso trabalho. Nossa metodologia propõe dividir o processo de integração de visões nas etapas abaixo:

(i) **Decomposição**: certas formas de dependências funcionais sugerem a presença de classes embutidas no esquema conceitual. Durante a etapa de decomposição, essas dependências funcionais são removidas de modo a eliminar a redundância do esquema e dessa forma, minimizar as anomalias de inserção, remoção e atualização. As novas classes adicionadas são então combinadas no passo de combinação a seguir.

(ii) **Combinação I**: durante a etapa de combinação I, as várias visões dos usuários são analisadas e comparadas para determinar a correspondência entre extensões das classes das visões dos usuários. O resultado desse processo é o esquema combinado inicial, o qual contém todas as visões originais e um conjunto de assertivas de correspondência de extensão. Essas assertivas de correspondência de extensão representam os diferentes tipos de relacionamento semântico existentes entre as extensões das classes de uma ou mais visões.

(iii) **Otimização I**: uma vez identificadas as equivalências entre as extensões das classes das visões dos diversos grupos de usuários, aplicamos ao esquema combinado inicial uma série de transformações de modo a obter um esquema global otimizado. O objetivo principal dessa etapa é reduzir redundâncias e o tamanho do esquema através da fusão das classes e atributos comuns.

(iv) **Combinação II**: em seguida, o esquema é novamente analisado. O objetivo dessa segunda etapa de combinação é identificar relacionamentos semânticos entre classes de associação e entre associações. O resultado da fase de combinação II é um novo esquema combinado, composto por todas as visões originais e um conjunto de assertivas de correspondência que expressam de que maneira as classes de associação e associações estão relacionadas nas várias visões dos usuários.

(v) **Otimização II**: tendo sido identificados os relacionamentos semânticos entre classes de associação e entre associações, faz-se necessário reestruturar novamente o esquema de modo a eliminar as redundâncias identificadas no passo (iv).

Cada etapa da nossa metodologia de integração de visões está baseada no uso de um conjunto de padrões os quais permitem que a integração de esquemas seja realizada

de forma segura e eficiente. Esses padrões estão agrupados em um catálogo de padrões apresentado no capítulo 4.

Apesar da forma seqüencial da apresentação, as etapas descritas acima tendem a ser interativas, baseadas no refinamento sucessivo do esquema conceitual.

O uso de um modelo de dados com grande poder de abstração não é suficiente para tornar fácil os processos de combinação das visões em aplicações de grande porte. Embora o modelo semântico da UML adotado permita ao projetista representar a semântica das diversas formas de propriedades inter-esquemas, ele não pode evitar situações em que tais relacionamentos não tenham sido detectados. Muitos pesquisadores, entre eles [EN00], têm investigado o uso de ferramentas interativas para ajudar a identificar relacionamentos entre visões. Dentre os problemas abordados, destaca-se a detecção de possíveis conflitos entre os esquemas. Os critérios utilizados para o estabelecimento de equivalência entre as diferentes representações são descritos a seguir.

1. **Conflitos Semânticos**

   - Homonímia: diferentes conceitos possuem o mesmo nome.
   - Sinonímia: conceitos similares possuem nomes diferentes.
   - Restrição de Modelagem: conjuntos equivalentes de conceitos são agrupados seguindo diferentes critérios.
   - Descritivos: diferentes perspectivas aplicadas a um mesmo conceito.

2. **Conflitos Estruturais**

   - Modelagem: um mesmo conceito é modelado com uma estrutura diferente.
   - Agregação: elementos do universo de discurso são agrupados de forma distinta.
   - Domínio de atributos: diferenças na forma de representação no que diz respeito ao domínio de um atributo.

Já nas fases de otimização, um dos problemas encontrados está no fato de que algumas formas de redundâncias capturadas não podem ser diretamente integradas. Esse problema pode ser resolvido através de transformações de esquemas que compatibilizem as visões, resolvam conflitos e tratem das similaridades. No entanto, na medida que as transformações de esquemas ocorrem, as assertivas de correspondência identificadas devem ser mapeadas para o contexto do esquema transformado.

Capítulo 4

Catálogo de Padrões

4.1 Introdução

O projeto de sistemas tem sofrido verdadeiras revoluções nos últimos anos. Cada nova geração de projetistas ignora o trabalho feito anteriormente e constrói uma nova forma de pensar e executar seus projetos [Mul99]. No entanto, recentemente, os projetistas começaram a aceitar a idéia de que existem alguns aspectos do projeto que não devem mudar. Eles perceberam que a experiência e o conhecimento adquirido ao longo dos anos pode ser adaptado e reutilizado na solução de problemas diversos. O resultado desse processo que enfatiza o reuso do conhecimento no projeto de sistemas foi o surgimento do conceito de padrão.

Na verdade, os padrões surgiram da iniciativa individual de alguns pesquisadores, destacando-se dentre eles Kent Beck e Ward Cunningham, dois dos pioneiros da linguagem Smalltalk, que no início da década de 90 divulgaram a idéia de padrões apresentada pelo arquiteto e filósofo Christopher Alexander. Além destes, em 1992, Jim Coplien descreveu em seu livro *Advanced C++ Programming Styles and Idioms* [Cop92], um conjunto de idiomas bastante úteis para os usuários da linguagem C++. Alguns desses pesquisadores vieram constituir o grupo chamado *Hillside Group*, cujo objetivo era aprofundar o conhecimento relacionado aos padrões. Uma maior divulgação dos padrões, no entanto, só surgiu a partir da conferência **PLOP** - *Pattern Language of Programming* - lançada pelo *Hillside Group* em 1994 e com a publicação do livro da *Gang of Four*, em 1995.

Por tratar-se de uma tecnologia relativamente recente, temos observado discussões acirradas na comunidade dos padrões, envolvendo discussões sobre o que é exatamente um padrão. Esta situação dificulta a adoção de uma definição única [Fow97a]. Neste trabalho, definimos um padrão como *uma regra que expressa uma relação entre um determinado contexto, um problema e uma solução* [A+77].

Os padrões são muito populares na comunidade da engenharia de software e, nos últimos anos, também vêm despertando um interesse crescente na comunidade que trabalha com a orientação a objeto. Contudo, temos observado que pouco esforço tem sido dedicado a um estudo mais aprofundado da aplicabilidade e utilidade dos padrões na modelagem conceitual dos bancos de dados. Neste capítulo apresentamos os padrões desenvolvidos em nosso estudo para auxiliar os projetistas nas atividades de modelagem conceitual, em especial, nos passos do processo de integração de visões.

Na modelagem conceitual tradicional, o projetista descreve as características gerais de uma aplicação, abstraindo-se dos detalhes de implementação [Bro84]. Os padrões, por sua vez, podem complementar esse processo através de exemplos e experiências adquiridas. O projetista é induzido a pensar em termos de problemas e soluções. Além disso, os padrões suportam o reuso de soluções de projeto para problemas similares e também permitem especificar propriedades genéricas do esquema conceitual de uma aplicação, resultando em projetos conceituais mais abstratos [G+99].

Na seção 4.2, discutimos o formato adotado para a estrutura dos padrões. Na seção 4.3, mostramos a classificação efetuada com padrões. Na seção 4.4, alocamos os padrões nas principais atividades do processo de integração de visões. Na seção 4.5, apresentamos o catálogo de padrões proposto.

4.2 O Formato dos Padrões

Em geral, os padrões são expressos utilizando o formato denominado *Formato de Alexander* que define o que um padrão deve conter [Cop98]. Os padrões desenvolvidos em nosso estudo adotam uma variação desse formato e estão definidos utilizando a estrutura descrita a seguir:

- *nome do padrão* - nome usado para identificar univocamente o padrão;
- *contexto* - um padrão soluciona um problema inserido em um contexto específico;
- *problema* - descreve o problema que o padrão soluciona;
- *solução* - propõe a solução para o problema descrito;
- *exemplo* - os exemplos utilizados têm o objetivo de facilitar o entendimento dos padrões;
- *usos conhecidos* - mostra onde o problema abordado pelo padrão já foi tratado;

- *padrões relacionados* - mostra os diversos relacionamentos que podem ocorrer entre os padrões. Algumas vezes, um problema abordado por um padrão pode ser uma especialização, generalização ou uma combinação de problemas já solucionados por outros padrões.

4.3 Classificação dos Padrões

Nesta seção apresentamos o catálogo dos padrões desenvolvidos em nosso estudo para modelagem conceitual utilizando o diagrama de classes da UML. Agrupamos os padrões em três categorias principais, classificadas de acordo com a etapa do processo de integração de visões que eles tratam (ver seção 4.4): *C1.Padrões de Decomposição*, *C2.Padrões de Combinação*, *C3.Padrões de Otimização*. Além dos padrões específicos para o processo de integração de visões, reunimos na categoria *C4.Padrões de Refinamento* os padrões que propõem algumas técnicas de modelagem conceitual que podem ser aplicadas em qualquer ponto do processo de integração com o objetivo de refinar as representações encontradas.

C1. Padrões de Decomposição

P1. *Decompondo Classes para Eliminar Dependências Funcionais Indesejáveis* - identifica no esquema conceitual a presença de classes embutidas representadas por restrições de dependência funcional. As classes são então decompostas de forma a eliminar as redundâncias capturadas pelas dependências funcionais identificadas.

C2. Padrões de Combinação

C2.1 Identificação de Equivalência Semântica entre Extensões de Classes

P2. *Identificando Correspondências entre Extensões de Classes* - identifica relacionamentos semânticos entre extensões das classes de um esquema conceitual.

P3. *Identificando Associações Ocultas* - identifica associações ocultas a partir da análise do esquema conceitual.

C2.2 Identificação de Equivalência Semântica entre Classes de Associação

P4. *Identificando Correspondências entre Classes de Associação* - identifica restrições de dependência existencial entre classes de associação.

C2.3 Identificação de Equivalência Semântica entre Classes de Associação e Classes

P5. *Identificando Correspondências entre uma Classe de Associação e uma Classe Participante da Classe de Associação* - identifica relacionamentos semânticos entre extensões de uma classe de associação e uma das classes participantes da classe de associação.

C2.4 Identificação de Equivalência Semântica entre Associações

P6. *Identificando Correspondências entre Associações* - identifica restrições de dependência existencial entre associações.

P7. *Identificando Associações Derivadas* - verifica se uma associação é derivável da composição de duas ou mais associações.

C2.5 Identificação de Equivalência Semântica entre Classes de Associação e Associações

P8. *Identificando Correspondências entre uma Classe de Associação e uma Associação* - identifica relacionamentos semânticos entre uma classe de associação e uma associação definida entre duas classes participantes da classe de associação.

P9. *Identificando Classes de Associação Derivadas* - verifica se uma classe de associação é derivável da composição das associações existentes entre as classes participantes da classe de associação.

C3. Padrões de Otimização

C3.1 Reestruturação de Esquemas a partir das Assertivas de Correspondência

P10. *Integrando Classes Equivalentes* - reestrutura o esquema conceitual de modo a capturar as assertivas de correspondência de equivalência de extensão entre as classes.

P11. *Eliminando Assertivas de Correspondência de Subconjunto entre Extensões de Classes* - reestrutura o esquema conceitual de modo a capturar as assertivas de correspondência de subconjunto entre extensões das classes.

P12. *Representando Correspondências entre uma Classe de Associação e uma Classe Participante da Classe de Associação* - reestrutura o esquema conceitual a partir das correspondências semânticas identificadas entre as extensões de uma classe de associação e uma das classes participantes da classe de associação.

P13. *Integrando Associações Equivalentes* - reestrutura o esquema conceitual capturando as assertivas de correspondência de equivalência entre associações.

P14. *Eliminando Assertivas de Correspondência de Subconjunto entre Associações* - reestrutura o esquema conceitual capturando as assertivas de correspondência de subconjunto entre associações.

P15. *Removendo Associações Derivadas* - reestrutura o esquema conceitual de modo a remover associações deriváveis da composição de duas ou mais associações.

P16. *Removendo Classes de Associação Derivadas* - reestrutura o esquema conceitual de modo a remover classes de associação deriváveis da composição das associações existentes entre as classes participantes da classe de associação.

C3.2 Reestruturação de Esquemas a partir das Restrições de Dependência Existencial

P17. *Integrando Classes de Associação Equivalentes* - reestrutura o esquema conceitual visando remover as restrições de dependência existencial de equivalência entre classes de associação.

P18. *Eliminando Dependências Existenciais de Subconjunto entre Classes de Associação* - reestrutura o esquema conceitual visando remover as restrições de dependência existencial de subconjunto entre classes de associação.

P19. *Integrando Classes de Associação e Associações Equivalentes* - reestrutura o esquema conceitual para capturar as restrições de dependência existencial de equivalência entre classes de associação e associações.

P20. *Eliminando Dependências Existenciais de Subconjunto entre uma Classe de Associação e uma Associação* - reestrutura o esquema conceitual para capturar as restrições de dependência existencial de subconjunto entre classes de associação e associações.

P21. *Adicionando Associações Ocultas* - reestrutura o esquema conceitual adicionando associações ocultas identificadas.

C4. Padrões de Refinamento

P22. *Associações Mandatórias X Associações Opcionais* - reestrutura o esquema conceitual de modo a eliminar do esquema conceitual a redundância causada pelas associações opcionais.

P23. *Classe de Associação X Classe* - reestrutura o esquema conceitual representando uma classe de associação binária como uma classe plena.

P24. *Adicionando Classes Descritivas* - refina o esquema conceitual fazendo uso das classes descritivas.

P25. *Modelando Quantidades* - reestrutura o esquema conceitual, visando propor uma modelagem mais eficiente para os atributos que armazenam quantidades.

P26. *Modelando Históricos* - reestrutura o esquema conceitual para representar de forma mais adequada o histórico dos objetos.

A figura 4.1 apresenta a classificação geral dos padrões de modelagem desenvolvidos, assim como seus inter-relacionamentos. As elipses representam respectivamente as categorias dos problemas abordados. Os padrões são representados por retângulos. As setas implicam na existência de um relacionamento entre os padrões, ou seja, implica que um padrão usa ou depende dos resultados obtidos com a aplicação do outro.

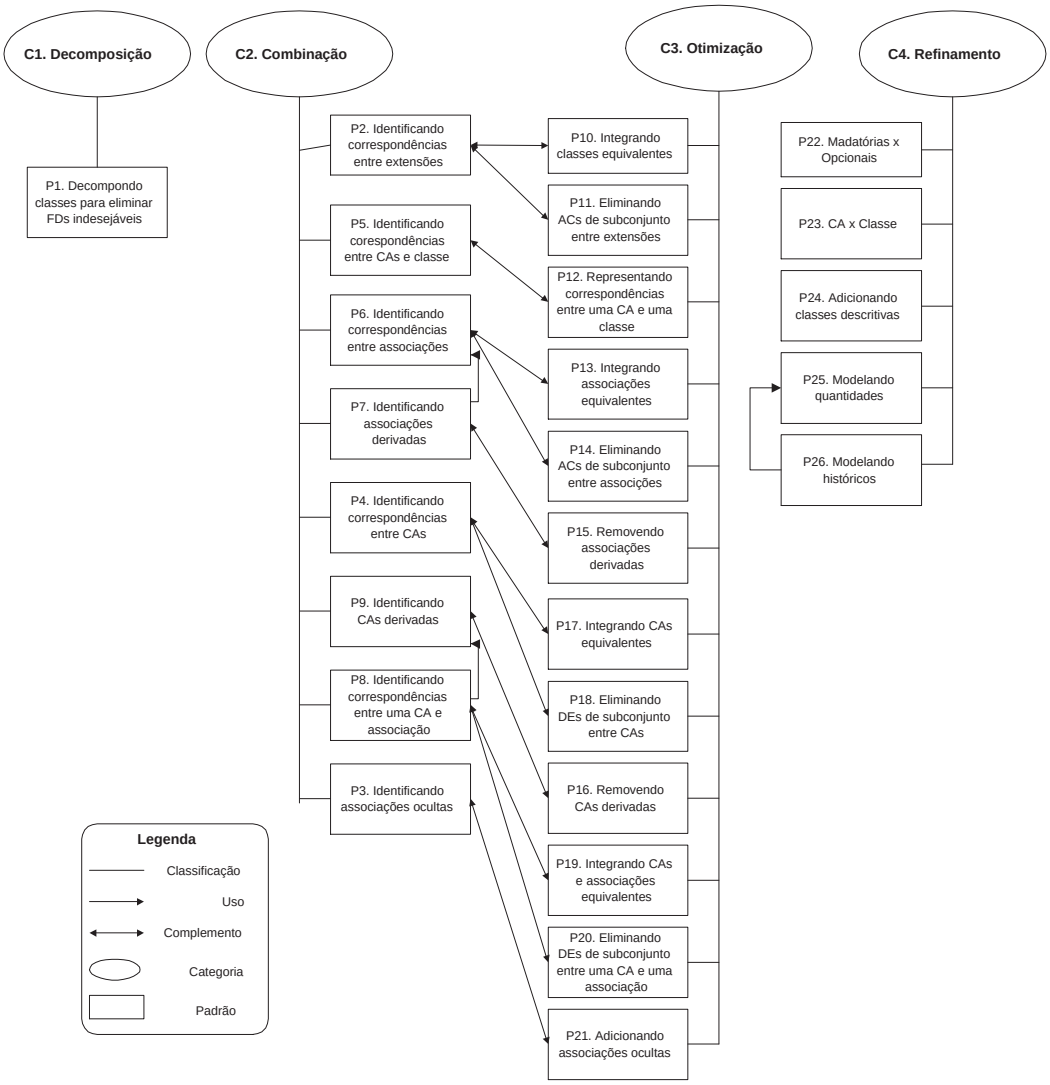

Figura 4.1: Classificação dos Padrões para Integração de Visões Modeladas com a UML

4.4 Usando os Padrões Durante a Integração de Visões

Os padrões propostos são usados durante a modelagem e integração das visões. Nos itens de (i) a (v), definimos o uso desses padrões nas diversas etapas do processo de integração.

(i) **Decomposição**: o padrão **P1** é utilizado na etapa de decomposição com o objetivo de remover dependências funcionais indesejáveis, de modo a minimizar a redundância do esquema.

(ii) **Combinação I**: os padrões **P2** e **P3** são utilizados durante essa etapa para determinar assertivas de correspondência entre extensões das classes das visões dos usuários.

(iii) **Otimização I**: os padrões **P10**, **P11** e **P21** são usados durante a fase de otimização I para reduzir o tamanho do esquema através da fusão de classes e atributos comuns.

(iv) **Combinação II**: os padrões **P4**, **P5**, **P6**, **P7**, **P8** e **P9** são usados durante a etapa de combinação II com o objetivo de identificar relacionamentos semânticos entre classes de associação, entre associações e entre classes de associação e associações.

(v) **Otimização II**: os padrões **P12**, **P13**, **P14**, **P15** ,**P16**, **P17**, **P18**, **P19** e **P20** são utilizados para reestruturar novamente o esquema de forma a eliminar as redundâncias identificadas na etapa (iv).

Além dos padrões específicos para o processo de integração de visões, propomos alguns padrões que podem ser aplicados em qualquer ponto do processo de integração com o objetivo de refinar as representações encontradas. Estão agrupados nessa categoria os padrões **P22**, **P23**, **P24**, **P25**, **P26**.

4.4.1 Transformações de Esquemas

Grande parte dos padrões desenvolvidos em nosso trabalho sugere algum tipo de transformação de esquema com o objetivo de propor uma representação alternativa que seja ao mesmo tempo mais eficiente e equivalente à representação original. Segundo [Vid94], uma transformação de esquema deve preservar a semântica do esquema original.

No nosso enfoque, o processo de transformação de esquemas está baseado no conceito de equivalência de esquemas. Em termos gerais, dois esquemas são equivalentes se preservam a informação e as restrições de integridade. Uma transformação preserva a informação quando o esquema original e o esquema transformado têm capacidades de informação equivalentes [MIR93].

Definição 4.1 *Considere na figura 4.2 o esquema original* $\mathbf{S_1}$ *e o esquema transformado* $\mathbf{S_2}$*, onde* ϕ *especifica como instâncias de* $\mathbf{S_1}$ *são mapeadas em instâncias de* $\mathbf{S_2}$ *e* σ *especifica como instâncias de* $\mathbf{S_2}$ *são mapeadas em instâncias de* $\mathbf{S_1}$*.* $\mathbf{S_1}$ *e* $\mathbf{S_2}$ *possuem capacidades de informação equivalentes se e somente se:*

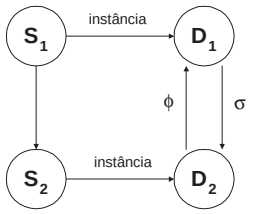

Figura 4.2: A Transformação Preserva a Informação

(i) para qualquer instância $\mathbf{D_1}$ *de* $\mathbf{S_1}$*,* $\phi(\mathbf{D_1})$ *é uma instância de* $\mathbf{S_2}$ *e* $\mathbf{D_1} = \sigma(\phi(\mathbf{D_1}))$

(ii) para qualquer instância $\mathbf{D_2}$ *de* $\mathbf{S_2}$*,* $\sigma(\mathbf{D_2})$ *é uma instância de* $\mathbf{S_1}$

Nesse trabalho, usamos as assertivas de correspondência para especificar formalmente o mapeamento entre o esquema original e o esquema transformado. O uso desse formalismo é importante porque através dele poderemos provar que as regras de transformação propostas em nossos padrões de fato preservam a informação.

Dependendo dos elementos envolvidos e da natureza da correspondência, as assertivas de correspondência podem ser classificadas em vários tipos. Como discutido na seção 2.3.4, no presente trabalho consideramos três grupos principais de assertivas de correspondência: assertivas de correspondência de extensão, assertivas de correspondência de atributos, assertivas de correspondência de caminhos.

4.5 Catálogo de Padrões

Nesta seção, apresentamos os padrões desenvolvidos em nosso trabalho para auxiliar nas tarefas de integração de visões e modelagem conceitual com a UML.

C1. Padrões de Decomposição

Nesta categoria, mostramos como identificar e remover do esquema restrições de dependência funcional indesejáveis.

P1. Decompondo Classes para Eliminar Dependências Funcionais Indesejáveis

Contexto: Suponha o esquema **S** da figura 4.3. Seja **A** uma classe, `At(A)` o conjunto de atributos da classe **A** e `Lg(A)` o conjunto de ligações da classe **A**. Considere que $\{a, x_1, x_2, ...x_n\ ,\ y_1, y_2, ...y_n\} \in$ `At(A)`, tal que $x_1, x_2, ...x_n \rightarrow y_1, y_2, ...y_n$ e $\{x_1, x_2, ...x_n\} \notin$ `Chaves(A)`.

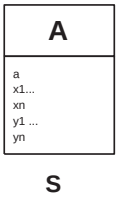

Figura 4.3: Classe Embutida

Problema: Remover a redundância representada pela dependência funcional $x_1, x_2, ...x_n \rightarrow y_1, y_2, ...y_n$.

Solução: O esquema **S** da figura 4.3 deve ser transformado no esquema **S'** como mostrado na figura 4.4.

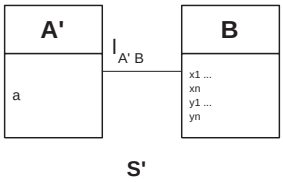

Figura 4.4: Removendo Classe Embutida

No esquema **S'**, a classe **A** foi decomposta nas classes **A'** e **B** e uma ligação $l_{A'B}$ foi definida entre elas. Os atributos e ligações de **S'** são distribuídos como segue:

(i) `At(A')` = `At(A)` - $\{x_1, x_2, ...x_n\ , y_1, y_2, ...y_n\}$

(ii) `At(B)` = $\{x_1, x_2, ...x_n\ , y_1, y_2, ...y_n\}$

(iii) `Lg(A')` = `Lg(A)` + $l_{A'B}$

O mapeamento entre o esquema transformado **S'** e o esquema original **S** é formalmente especificado pelas assertivas de correspondência abaixo:

(i) $\texttt{A} \equiv \texttt{A'}$

(ii) $\texttt{A}.x_i \equiv \texttt{A'}.l_{A'B}.x_i$

(iii) $\texttt{A}.y_i \equiv \texttt{A'}.l_{A'B}.y_i, \ 1 \leq \texttt{i} \leq \texttt{n}$

(iv) os demais atributos de **A** são mapeados diretamente em atributos de **A'**.

Exemplo: Suponha o esquema $\mathbf{S_1}$ mostrado na figura 4.5. Analisando a semântica dos atributos da classe **Livro**, verificamos que as seguintes dependências funcionais são válidas para todas as suas instâncias:

$\mathbf{FD_1}$) `#livro` $\rightarrow$ `título, #editora`

$\mathbf{FD_2}$) `#editora` $\rightarrow$ `nome-editora`

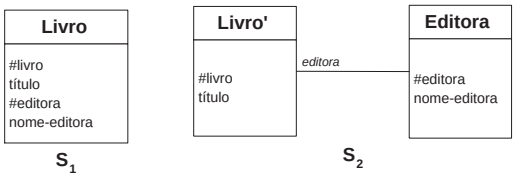

Figura 4.5: Exemplo - Classe Embutida

A dependência funcional $\mathbf{FD_1}$ especifica que o atributo **#livro** determina unicamente os atributos **título** e **#editora**. A dependência funcional $\mathbf{FD_2}$ especifica que o atributo **#editora** determina unicamente o atributo **nome-editora**. Como o atributo **#editora** não faz parte da chave da classe **Livro**, a assertiva $\mathbf{FD_2}$ está indicando que o atributo **nome-editora**, na verdade, é um atributo de uma outra classe que está embutida na classe **Livro**, cuja chave é o atributo **#editora**. Isso implica que a classe **Livro** está representando mais de um objeto do mundo real. Nesse caso, propomos reestruturar o esquema $\mathbf{S_1}$ como mostrado no esquema $\mathbf{S_2}$ da figura 4.5. No esquema $\mathbf{S_2}$, a classe **Livro** foi decomposta nas classes **Livro'** e **Editora**. Os atributos **#editora** e **nome-editora** foram transferidos para a nova classe **Editora** e a ligação *editora* foi definida entre as classes **Livro'** e **Editora**.

O mapeamento entre o novo esquema $\mathbf{S_2}$ e o esquema original $\mathbf{S_1}$ é formalmente especificado pelas assertivas de correspondência abaixo:

(i) `Livro` $\equiv$ `Livro'`

(ii) `Livro.#livro` $\equiv$ `Livro'.#livro`

(iii) `Livro.título` $\equiv$ `Livro'.título`

(iv) `Livro.#editora` $\equiv$ `Livro'.`*editora*`.#editora`

(v) `Livro.nome-editora` $\equiv$ `Livro'.`*editora*`.nome-editora`

Usos Conhecidos: Uma discussão no enfoque do modelo ER relacionada a esse padrão pode ser encontrada em [Lós98]. [Lin85] define as formas normais para o modelo ER.

C2. Padrões de Combinação

Nesta categoria, reunimos os padrões que abordam as atividades envolvidas nas etapas de combinação I e II do processo de integração de visões. Durante a combinação I, as várias visões dos usuários são analisadas e comparadas para determinar a correspondência entre extensões das classes das visões dos usuários. O resultado desse passo é o esquema combinado inicial. Esse esquema é composto por todas as visões originais e um conjunto de assertivas de correspondência que expressam de que maneira as classes nas várias visões estão relacionadas. Durante a combinação II, o esquema é novamente analisado. O objetivo dessa segunda etapa de combinação é identificar relacionamentos semânticos entre classes de associação e entre associações. O resultado da fase de combinação II é o esquema combinado composto por todas as visões originais e um conjunto de assertivas de correspondência que expressam de que maneira as classes, classes de associação e associações estão relacionadas nas várias visões dos usuários.

C2.1 Identificação de Equivalência Semântica entre Extensões de Classes

Nesta seção, discutimos como estabelecer equivalências semânticas entre extensões de classes pertencentes a um ou mais esquemas conceituais. A vantagem em especificar essas correspondências semânticas durante a modelagem conceitual está no fato de conseguirmos identificar redundâncias indesejáveis, de forma que possam ser posteriormente eliminadas dos esquemas conceituais.

P2. Identificando Correspondências entre Extensões de Classes

Contexto: Algumas situações de modelagem sugerem a possibilidade da existência de relacionamentos semânticos entre extensões de classes. Nesse padrão consideramos duas situações distintas: classes com nomes semelhantes e classes com atributos similares.

Problema: Identificar correspondências semânticas entre extensões de classes.

Solução: A semântica da aplicação deve ser analisada para podermos verificar a presença de um relacionamento semântico entre as extensões dessas classes. Esse relacionamento semântico pode ser formalmente capturado pelas assertivas de correspondência de extensão discutidas na seção 2.3.4.

Exemplo:

- **Caso 1:** *Classes com nomes semelhantes*

Suponha o esquema $\mathbf{S_1}$ da figura 4.6. Analisando o esquema mostrado, identificamos duas classes com nomes sinônimos: as classes **Empregado** e **Funcionário**. Essa situação indica que pode existir uma correspondência semântica entre as classes **Empregado** e **Funcionário**.

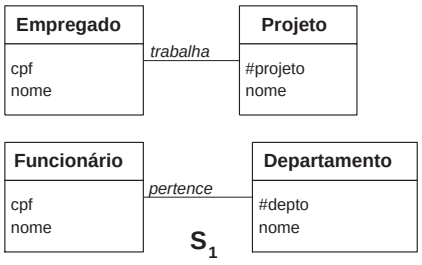

Figura 4.6: Exemplo - Classes com Nomes Semelhantes

Tendo identificado no esquema conceitual classes com nomes semelhantes, o próximo passo é analisar a semântica do mundo real para podermos afirmar se de fato existe um relacionamento semântico entre as extensões dessas classes. No contexto do exemplo mostrado na figura 4.6, temos que todo funcionário é um empregado e vice-versa. Existe um mapeamento um para um entre as instâncias das classes **Empregado** e **Funcionário** que especifica que **e** é uma instância de **Empregado** se e somente se **f** é uma instância de **Funcionário** e **e.cpf** = **f.cpf**. Isso significa que para qualquer instância **e** de **Empregado** existe uma instância **f** de **Funcionário** tal que **e** $\equiv$ **f**. Logo, podemos concluir que:

(i) `Empregado` $\equiv$ `Funcionário`

(ii) `Empregado.cpf` $\equiv$ `Funcionário.cpf`

(iii) `Empregado.nome` $\equiv$ `Funcionário.nome`

- **Caso 2:** *Classes com atributos similares*

Considere o esquema $\mathbf{S_2}$ mostrado na figura 4.7. Analisando as classes **Empregado** e **Gerente**, verificamos que elas possuem um conjunto de atributos em comum: os atributos **cpf**, **nome** e **salário**. Essa situação sugere a existência de um relacionamento semântico entre as extensões das classes **Empregado** e **Gerente**.

Para podermos assegurar a presença de uma correspondência semântica entre as extensões das classes **Empregado** e **Gerente**, também é necessário verificar a semântica

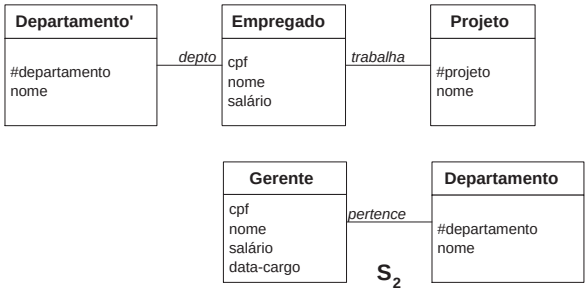

Figura 4.7: Exemplo - Classes com Atributos Similares

associada ao esquema $\mathbf{S_2}$. No contexto do exemplo mostrado na figura 4.7, identificamos que todo gerente é um empregado, ou seja, para toda instância **g** de **Gerente** existe uma instância **e** de **Empregado**, tal que existe um mapeamento um para um entre as instâncias das classes **Gerente** e **Empregado** que especifica que **g.cpf** = **e.cpf**. Essa restrição é formalmente capturada pelas assertivas de correspondência:

(i) `Gerente` $\subset$ `Empregado`

(ii) `Departamento'` $\equiv$ `Departamento`

(iii) `Gerente.cpf` $\equiv$ `Empregado.cpf`

(iv) `Gerente.nome` $\equiv$ `Empregado.nome`

(v) `Gerente.salário` $\equiv$ `Empregado.salário`

(vi) `Gerente.`$gerencia$ $\equiv$ `Empregado.`$depto$

(vii) `Departamento'.#departamento` $\equiv$ `Departamento.#departamento`

(viii) `Departamento'.nome` $\equiv$ `Departamento.nome`

Usos Conhecidos: Uma discussão relacionada à identificação de correspondências semânticas no esquema ER pode ser encontrada em [Sto91].

Padrões Relacionados: Nos padrões **P10** e **P11**, propomos a reestruturação dos esquemas conceituais de forma a capturar as assertivas de correspondência de extensão identificadas.

P3. Identificando Associações Ocultas

Contexto: Os projetistas ficam geralmente confusos se devem representar associações entre classes como um atributo de uma classe ou como uma associação entre classes.

Suponha o esquema **S** da figura 4.8.

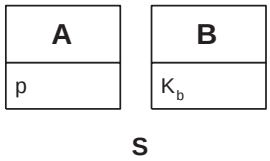

Figura 4.8: Associação Oculta entre Classes

Sejam **A** e **B** classes, `At(A)` o conjunto de atributos da classe **A**, `At(B)` o conjunto de atributos da classe **B** e $\mathbf{K_b}$ chave da classe **B**. Algumas situações de modelagem que sugerem a possibilidade de existir uma restrição de dependência existencial entre as classes **A** e **B** são:

1. Existe um atributo $\mathbf{p} \in$ `At(A)` com nome similar ao nome da classe **B**.
2. Existe um atributo $\mathbf{p} \in$ `At(A)` com nome e tipo similar ao nome e tipo do atributo chave $\mathbf{K_b}$ da classe **B**.

Problema: Verificar se existe uma restrição de dependência existencial entre as classes **A** e **B** envolvendo os atributos **p** e $\mathbf{K_b}$.

Solução: A semântica deve ser verificada. De acordo com a análise da semântica, temos que existe uma restrição entre as classes **A** e **B**, dada por $\mathbf{A[p]} \subset \mathbf{B[K_b]}$, se para toda instância **a** de **A** existe uma instância **b** de **B** tal que $\mathbf{a.p} = \mathbf{b.K_b}$.

É importante notar que, apesar de tratar apenas de chaves simples, a solução proposta pode ser estendida para tratar de chaves compostas.

Exemplo: Suponha o esquema $\mathbf{S_1}$ mostrado na figura 4.9. No esquema $\mathbf{S_1}$, verificamos que a classe **Livro** possui um atributo **autor** que corresponde ao mesmo nome da classe **Autor**. Pela semântica do mundo real, temos que existe uma restrição entre as classes **Livro** e **Autor** que especifica que para toda instância **l** de **Livro**, existe uma instância **a** de **Autor**, tal que **l.autor** = **a.nome**. Essa restrição é capturada pela assertiva de dependência existencial `Livro[autor]` $\subset$ `Autor[nome]`. Isso significa que o atributo **autor**, em vez de representar uma propriedade da classe **Livro**, na verdade representa uma associação oculta existente entre as classes **Livro** e **Autor**.

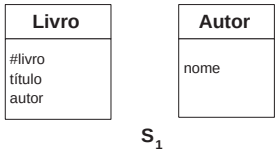

Figura 4.9: Exemplo - Dependência Existencial entre Classes

Usos Conhecidos: Uma discussão relacionada a esse padrão pode ser encontrada em [Lós98].

Padrões Relacionados: No padrão **P21**, discutimos como capturar no próprio esquema conceitual as restrições de dependência existencial identificadas.

C2.2 Identificação de Equivalência Semântica entre Classes de Associação

Nesta subcategoria abordamos a identificação de equivalências semânticas entre classes de associação.

P4. Identificando Correspondências entre Classes de Associação

Contexto: Uma restrição de dependência existencial entre classes de associação expressa como as instâncias de uma classe de associação dependem da existência de instâncias de uma outra classe de associação.

Suponha o esquema **S** mostrado na figura 4.10.

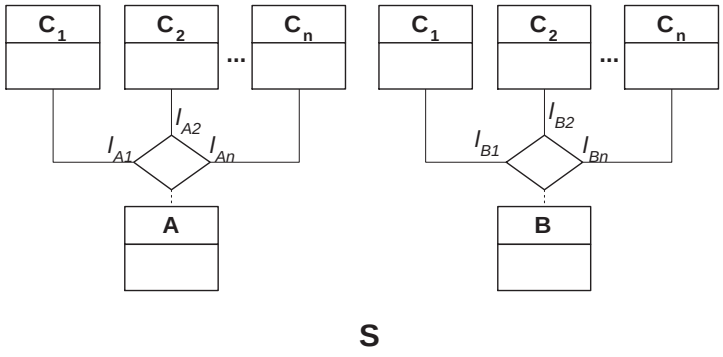

Figura 4.10: DE entre Classes de Associação

Sejam **A** e **B** classes de associação. Suponha que $\mathbf{C_1}$, $\mathbf{C_2}$, ..., $\mathbf{C_n}$ sejam classes em comum participantes das classes de associação **A** e **B**, $\mathbf{l_{Ai}}$, $1 \leq i \leq n$ ligações da classe de associação **A** e $\mathbf{l_{Bi}}$, $1 \leq i \leq n$ ligações da classe de associação **B**.

Problema: Verificar se existe uma restrição de dependência existencial entre as classes de associação **A** e **B**.

Solução: A semântica deve ser verificada. Pela análise da semântica, temos que existem dois tipos possíveis de relacionamento semântico entre as classes de associação **A** e **B**:

1. Existe uma restrição de dependência existencial de subconjunto entre **A** e **B**, dada por $\mathbf{A}[\mathbf{l_{A1}}, \mathbf{l_{A2}}, ..., \mathbf{l_{An}}] \subset \mathbf{B}[\mathbf{l_{B1}}, \mathbf{l_{B2}}, ..., \mathbf{l_{Bn}}]$, se a existência de uma instância **a** de **A** requer a existência de uma instância **b** de **B**, tal que $\mathbf{a}.\mathbf{l_{Ai}} = \mathbf{b}.l_{Bi}$, $1 \leq i \leq n$.

Se $[\mathbf{l_{A1}}, \mathbf{l_{A2}}, ..., \mathbf{l_{An}}]$ contém a chave de $\mathbf{A}$ e $[\mathbf{l_{B1}}, \mathbf{l_{B2}}, ..., \mathbf{l_{Bn}}]$ contém a chave de $\mathbf{B}$, então existe uma assertiva de correspondência de subconjunto de extensão entre $\mathbf{A}$ e $\mathbf{B}$ dada por $\mathbf{A} \subset \mathbf{B}$.

2. Existe uma restrição de dependência existencial de equivalência entre $\mathbf{A}$ e $\mathbf{B}$, dada por $\mathbf{A}[\mathbf{l_{A1}}, \mathbf{l_{A2}}, ..., \mathbf{l_{An}}] \equiv \mathbf{B}[\mathbf{l_{B1}}, \mathbf{l_{B2}}, ..., \mathbf{l_{Bn}}]$, se $\mathbf{a}$ é uma instância de $\mathbf{A}$ se e somente se existe uma instância $\mathbf{b}$ de $\mathbf{B}$, tal que $\mathbf{a}.\mathbf{l_{Ai}} = \mathbf{b}.l_{Bi}$, $1 \leq \mathrm{i} \leq \mathrm{n}$.

   Se $[\mathbf{l_{A1}}, \mathbf{l_{A2}}, ..., \mathbf{l_{An}}]$ contém a chave de $\mathbf{A}$ e $[\mathbf{l_{B1}}, \mathbf{l_{B2}}, ..., \mathbf{l_{Bn}}]$ contém a chave de $\mathbf{B}$, então existe uma assertiva de correspondência de equivalência de extensão entre $\mathbf{A}$ e $\mathbf{B}$ dada por $\mathbf{A} \equiv \mathbf{B}$.

Exemplo:

- **Caso 1:** *Dependência existencial de subconjunto entre classes de associação*

Considere os esquemas $\mathbf{S_1}$ e $\mathbf{S_2}$ da figura 4.11. As classes de associação **Oferta** e **Matrícula** possuem as classes **Disciplina** e **Professor** em comum. Temos que a existência de uma instância da classe de associação **Matrícula** associando uma disciplina $\mathbf{d}$ e um professor $\mathbf{p}$, requer a existência de uma associação de um professor $\mathbf{p}$ e uma disciplina $\mathbf{d}$ na classe de associação **Oferta**. Formalmente, para todo $\mathbf{m} \in$ **Matrícula**, existe um $\mathbf{o} \in$ **Oferta**, tal que $\mathbf{m}.\mathit{disciplina} = \mathbf{o}.\mathit{disciplina}$ e $\mathbf{m}.\mathit{professor} = \mathbf{o}.\mathit{professor}$. Essa restrição é capturada pela assertiva de dependência existencial de subconjunto `Matrícula[disciplina, professor]` $\subset$ `Oferta[disciplina, professor]`.

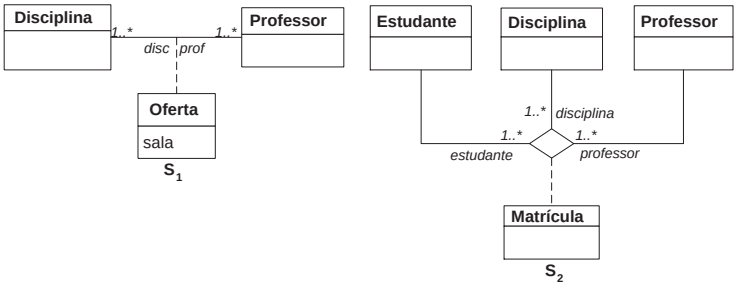

Figura 4.11: Exemplo - DE de Subconjunto entre Classes de Associação

- **Caso 2:** *Dependência existencial de equivalência entre classes de associação*

Suponha os esquemas $\mathbf{S_3}$ e $\mathbf{S_4}$ da figura 4.12. As classes de associação **Alocação** e **Matrícula** relacionam um mesmo conjunto de classes: as classes **Semestre**, **Disciplina** e **Professor**.

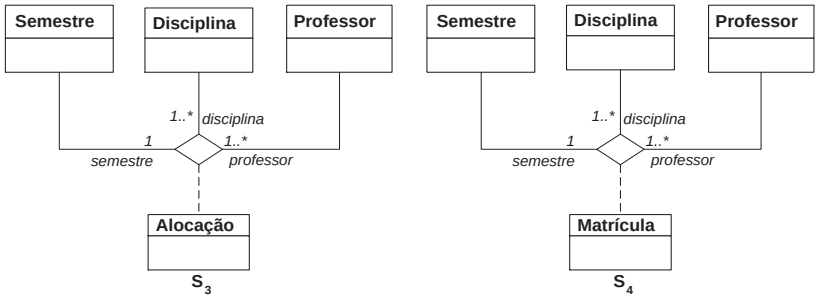

Figura 4.12: Exemplo - DE de Equivalência entre Classes de Associação

Analisando a semântica do exemplo mostrado na figura 4.12, temos que a existência de uma instância da classe de associação **Alocação** associando um semestre **s**, uma disciplina **d** e um professor **p**, requer a existência de uma instância da classe de associação **Matrícula** associando um semestre **s**, uma disciplina **d** e um professor **p** e vice-versa. Essa restrição é formalmente capturada pela assertiva de dependência existencial de equivalência `Alocação[semestre, disciplina, professor]` $\equiv$ `Matrícula[semestre, disciplina, professor]`.

Usos Conhecidos: A identificação de restrições de dependência existencial pode ser encontrada em [Vid94].

Padrões Relacionados: Os esquemas conceituais mostrados neste padrão não capturam as restrições de dependência existencial identificadas. Sugerimos reestruturá-los como discutido nos padrões **P17** e **P18**.

C2.3 Identificação de Equivalência Semântica entre Classes de Associação e Classes

Nesta subcategoria agrupamos os padrões que tratam da identificação de relacionamentos semânticos entre uma classe de associação e uma classe.

P5. Identificando Correspondências entre uma Classe de Associação e uma Classe Participante da Classe de Associação

Contexto: Uma classe de associação pode ser semanticamente equivalente ou ser uma subclasse de uma de suas classes participantes. Para formalizar os relacionamentos semânticos identificados utilizamos as assertivas de correspondência de extensão discutidas no capítulo 2.3.4.

Suponha o esquema **S** da figura 4.13.

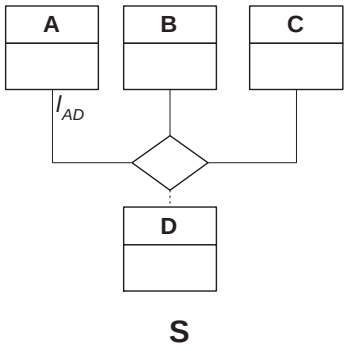

Figura 4.13: AC em uma Classe de Associação

Seja **D** uma classe de associação, **A** uma classe participante de **D** e l_{AD} uma ligação definida entre **A** e **D** (l_{AD}: **A** $\rightarrow$ **D**). Existe um mapeamento um para um entre as instâncias de **A** e **D**.

Problema: Verificar se existe um relacionamento semântico entre a classe de associação **D** e a classe **A** participante da classe de associação **D**.

Solução: O relacionamento semântico entre **A** e **D**, depende da multiplicidade associada à ligação l_{AD}. A seguir analisamos duas possibilidades para a multiplicidade de l_{AD}:

1. Existe uma assertiva de correspondência de equivalência entre as extensões de **A** e **D**, dada por **A** $\equiv$ **D**, se a multiplicidade de l_{AD} é 1.

2. Existe uma assertiva de correspondência de subconjunto entre as extensões de **A** e **D**, dada por $\mathbf{A} \subset \mathbf{D}$, se a multiplicidade de l_{AD} é 0..1.

Exemplo:

- **Caso 1:** *A classe de associação é semanticamente equivalente a uma de suas classes participantes*

Suponha o esquema $\mathbf{S_1}$ mostrado na figura 4.14. No esquema $\mathbf{S_1}$, temos que a classe de associação **Gerência** captura o fato de que um empregado **e** trabalha em um projeto **p** que é gerenciado pelo gerente **g**. Pela definição de classe de associação dada no capítulo 2, temos que toda instância da classe de associação **Gerência** está associada com uma única instância de cada uma das classes participantes **Gerente**, **Empregado** e **Projeto**. Formalmente, existe uma ligação de uma instância em **Gerência** para cada uma das classes participantes **Gerente**, **Empregado** e **Projeto**, cuja multiplicidade é 1 e cujo nome corresponde ao nome das classes participantes: *gerente*, *empregado* e *projeto*. Analisando o esquema $\mathbf{S_1}$, verificamos que a multiplicidade associada à ligação *trabalha*: **Empregado** $\rightarrow$ **Gerência** é 1. Isso implica que toda instância na classe **Empregado** pode estar associada com uma única instância da classe de associação **Gerência**. Daí, verificamos a existência de um mapeamento um para um entre as instâncias das classes **Empregado** e **Gerência**, sendo possível definir uma função bijetiva *f* dada por,

f: **Empregado** $\rightarrow$ **Gerência**.

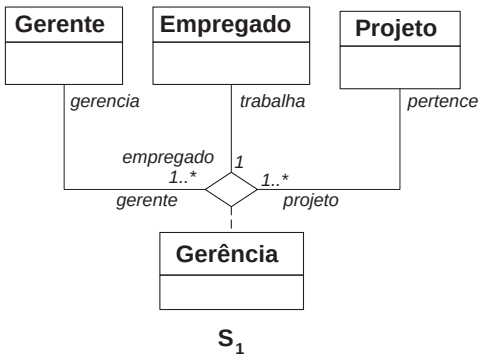

Figura 4.14: Exemplo - AC de Equivalência em uma Classe de Associação

Essa restrição é formalmente capturada pela assertiva de correspondência de equivalência de extensão `Empregado` $\equiv$ `Gerência`. Isso significa que as classes **Empregado** e **Gerência** contêm os mesmos objetos do mundo real. A restrição identificada entre a classe de associação **Gerência** e a classe **Empregado** sinaliza a necessidade de uma reestruturação no esquema conceitual, a qual será tratada no padrão **P12**.

- **Caso 2:** *A classe de associação é uma subclasse de uma de suas classes participantes*

Suponha o esquema $\mathbf{S_2}$ mostrado na figura 4.15. No esquema $\mathbf{S_2}$, temos que a classe de associação **Orientação** representa o fato de que um estudante **e** participa de um projeto **j** que é supervisionado pelo professor **p**. Pela definição de classe de associação, temos que toda instância da classe de associação **Orientação** está associada com uma única instância de cada uma das classes participantes **Professor**, **Projeto** e **Estudante**. Formalmente, existe uma ligação de uma instância em **Orientação** para cada uma das classes participantes **Professor**, **Projeto** e **Estudante**, cuja multiplicidade é 1 e cujo nome corresponde ao nome das classes participantes: *professor*, *projeto* e *estudante*. Além disso, analisando o esquema $\mathbf{S_2}$, identificamos que a multiplicidade associada à ligação *participa*: **Estudante** $\rightarrow$ **Orientação** é 0..1. Isso significa que toda instância da classe **Estudante** pode participar de 0 ou 1 associações com a classe de associação **Orientação**. Logo, verificamos a existência de um mapeamento um para um entre as instâncias das classes **Orientação** e **Estudante**, sendo possível definir uma função injetiva g dada por,

g: **Orientação** $\rightarrow$ **Estudante**.

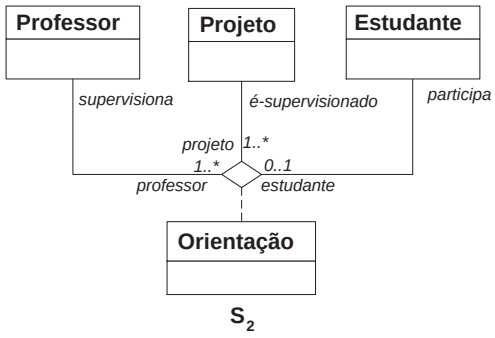

Figura 4.15: Exemplo - AC de Subconjunto em uma Classe de Associação

Essa restrição é formalmente capturada pela assertiva de correspondência de subconjunto de extensão `Orientação` $\subset$ `Estudante`. Essa restrição sugere a necessidade de uma reestruturação no esquema conceitual, a qual será tratada no padrão **P12**.

Usos Conhecidos: Uma discussão abordando esse assunto no enfoque do modelo entidade e relacionamento pode ser encontrada em [DSB97].

Padrões Relacionados: No padrão **P12**, mostramos como capturar diretamente na estrutura do esquema conceitual as assertivas de correspondência identificadas.

C2.4 Identificação de Equivalência Semântica entre Associações

Os padrões agrupados nessa subcategoria mostram como identificar correspondências semânticas entre associações.

P6. Identificando Correspondências entre Associações

Contexto: Nesse padrão, procuramos identificar no esquema conceitual associações diferentes relacionando classes em comum. Para especificarmos formalmente os relacionamentos semânticos identificados, utilizamos as assertivas de correspondência de caminho discutidas na seção 2.3.4.

Suponha o esquema **S** mostrado na figura 4.16.

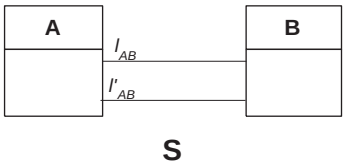

Figura 4.16: Correspondência Semântica entre Associações

Identificamos no esquema conceitual **S** duas ligações distintas (l_{AB} e l'_{AB}) relacionando as classes **A** e **B**. Essa situação sugere a existência de um relacionamento entre l_{AB} e l'_{AB}.

Problema: Verificar a existência de um relacionamento semântico entre as ligações l_{AB} e l'_{AB}.

Solução: A semântica do mundo real deve ser verificada. Pela análise da semântica, existem dois tipos possíveis de relacionamentos entre l_{AB} e l'_{AB}:

1. Existe uma assertiva de correspondência de subconjunto dada por $l_{AB} \subset l'_{AB}$, se para toda instância **a** de **A** e para toda instância **b** de **B**, se $\mathbf{b} \in \mathbf{a}.l_{AB}$, então $\mathbf{b} \in \mathbf{a}.l'_{AB}$.

2. Existe uma assertiva de correspondência de equivalência dada por $l_{AB} \equiv l'_{AB}$, se para toda instância **a** de **A** e para toda instância **b** de **B**, $\mathbf{b} \in \mathbf{a}.l_{AB}$ se e somente se $\mathbf{b} \in \mathbf{a}.l'_{AB}$.

Exemplo:

- **Caso 1:** *Assertiva de correspondência de subconjunto entre caminhos*

Suponha o esquema $\mathbf{S_1}$ da figura 4.17. Verificamos que as classes **Professor** e **Curso** estão associadas por meio de duas ligações distintas: através da ligação *ensina* e da ligação *pode-ensinar*.

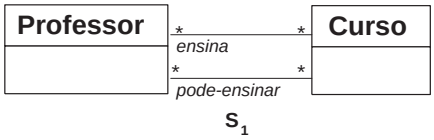

Figura 4.17: Exemplo - AC de Subconjunto entre Associações

Tendo identificado diferentes associações relacionando classes em comum, o próximo passo é verificar, a partir da análise da semântica do mundo real, se existe alguma restrição semântica entre as ligações *ensina* e *pode-ensinar*. De acordo com a semântica associada ao exemplo da figura 4.17, temos que um professor só ensina um determinado curso se estiver habilitado para tal. Formalmente, para toda instância **c** de **Curso** e para toda instância **p** de **Professor**, se $\mathbf{c} \in \mathbf{p}.ensina$ então $\mathbf{c} \in \mathbf{p}.pode\text{-}ensinar$. Essa restrição é capturada pela assertiva de correspondência de subconjunto de caminhos `ensina` $\subset$ `pode-ensinar`.

- **Caso 2:** *Assertiva de correspondência de equivalência entre caminhos*

Suponha o esquema $\mathbf{S_2}$ mostrado na figura 4.18. Analisando as associações mostradas, verificamos que as classes **Empregado** e **Projeto** estão relacionadas por meio de duas ligações distintas: as ligações *trabalha* e *é-requerido*.

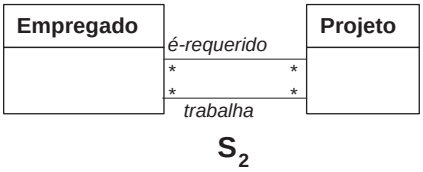

Figura 4.18: Exemplo - AC de Equivalência entre Associações

Devemos agora analisar a semântica da aplicação para podermos assegurar a existência de alguma restrição semântica entre as associações *trabalha* e *é-requerido*. No contexto do exemplo da figura 4.18, temos que um empregado trabalha em um determinado projeto se e somente se ele é requerido por esse projeto. Formalmente, para toda instância **p** de

Projeto e para toda instância **e** de **Empregado**, $\mathbf{p} \in \mathbf{e}.$*trabalha* se e somente se $\mathbf{p} \in \mathbf{e}.$*é-requerido*. Essa restrição é capturada pela assertiva de correspondência de equivalência de caminhos `trabalha` $\equiv$ `é-requerido`.

Usos Conhecidos: [SNL86] aborda o problema de correspondências semânticas entre relacionamentos no contexto do modelo ER.

Padrões Relacionados: As restrições de dependência existencial entre associações identificadas neste padrão não estão capturadas no esquema conceitual. Nos padrões **P13** e **P14** reestruturamos os esquemas de modo a capturar as restrições identificadas diretamente na estrutura desses esquemas.

É importante notar que também é possível estabelecer uma equivalência semântica entre uma associação e uma composição de duas ou mais associações. No padrão **P7**, mostramos em que situações esse tipo especial de equivalência entre associações pode ser identificada.

P7. Identificando Associações Derivadas

Contexto: Este padrão ajuda a verificar a existência de associações redundantes no esquema conceitual. Uma associação derivada ou redundante é aquela cujas instâncias podem ser inferidas a partir de instâncias de outras associações. A importância de identificarmos as associações derivadas no esquema conceitual está no modo como elas poderão influenciar a implementação do banco de dados, independente do modelo lógico adotado.

Suponha o esquema **S** da figura 4.19.

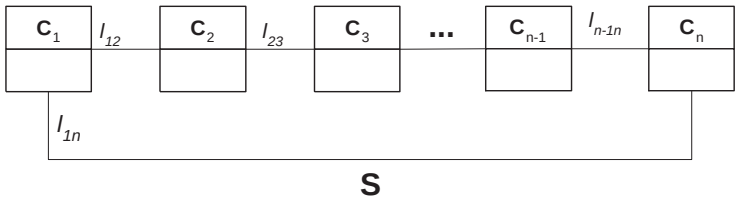

Figura 4.19: Associação Derivada

Existem dois caminhos ligando as classes $\mathbf{C_1}$ e $\mathbf{C_n}$: o caminho l_{1n} e o caminho $l_{12} \bullet ... \bullet l_{n-1n}$. Identificamos, portanto, a presença de um ciclo nas associações. Existe um ciclo em um esquema quando existe mais de um caminho ligando duas classes no esquema conceitual. Ao detectar a existência de um ciclo, é necessário verificar a compatibilidade das multiplicidades que compõem esse ciclo. As multiplicidades de dois caminhos são compatíveis se os valores mínimo e máximo das multiplicidades desses caminhos são iguais. Se existe um ciclo e as multiplicidades desse ciclo são compatíveis, então pode existir uma associação derivada no esquema conceitual.

Problema: Verificar se a ligação l_{1n} é derivável da composição das ligações $l_{12} \bullet ... \bullet l_{n-1n}$.

Solução: É necessário verificar a semântica associada ao esquema. Pela análise da semântica de **S**, temos que l_{1n} é derivada da composição dos caminhos $l_{12} \bullet ... \bullet l_{n-1n}$ se para qualquer instância $\mathbf{c_1}$ de $\mathbf{C_1}$, então $\mathbf{c_1}.l_{1n} = \mathbf{c_1}.(l_{12} \bullet ... \bullet l_{n-1n})$. Nesse caso, existe uma assertiva de correspondência de equivalência de caminhos dada por $l_{1n} \equiv (l_{12} \bullet ... \bullet l_{n-1n})$.

Exemplo: Considere o esquema $\mathbf{S_1}$ mostrado na figura 4.20. Existem dois caminhos ligando as classes **Peça** e **Fornecedor**: o caminho *é-fornecida* e o caminho *está-contida* • *é-preenchido*. Identificamos, portanto, a presença de um ciclo nas associações. Existe um ciclo em uma associação quando existe mais de um caminho ligando duas classes no esquema conceitual.

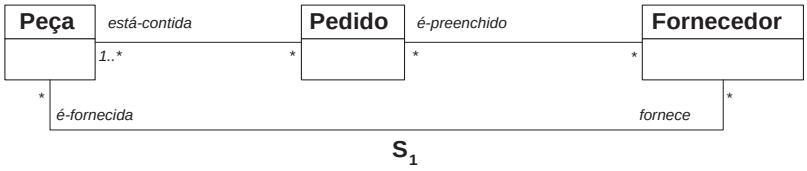

Figura 4.20: Exemplo - Associação derivada

Ao detectar a existência de um ciclo no esquema conceitual, é necessário verificar a compatibilidade das multiplicidades que compõem esse ciclo. As multiplicidades de dois caminhos são compatíveis se os valores mínimo e máximo das multiplicidades desses caminhos são iguais. Por exemplo, as multiplicidades do ciclo da figura 4.20 são compatíveis, pois uma peça pode estar associada a zero ou mais fornecedores pelo caminho *é-fornecida* e pode estar associada a zero ou mais fornecedores pelo caminho *está-contida* • *é-preenchido*.

As condições estruturais discutidas até o momento são apenas necessárias, mas não suficientes para determinar se uma associação é derivada de fato. Também é necessário verificarmos se os caminhos representam o mesmo fato do mundo real. No contexto do exemplo mostrado na figura 4.20, temos que o caminho *é-fornecida* é derivável da composição dos caminhos *está-contida* e *é-preenchido*, pois para qualquer instância **p** de **Peça**, então $\mathbf{p}.\textit{é-fornecida} = \mathbf{p}.(\textit{está-contida} \bullet \textit{é-preenchido})$. Esta restrição é formalmente capturada pela assertiva de correspondência de equivalência caminhos `é-fornecida` $\equiv$ `está-contida • é-preenchido`.

Usos Conhecidos: Uma discussão relacionada à existência de derivação em esquemas conceituais pode ser encontrada em [Fow97b].

Padrões Relacionados: No padrão **P15** mostramos como remover associações derivadas do esquema conceitual.

C2.5 Identificação de Equivalência Semântica entre Classes de Associação e Associações

Nesta seção, focalizamos os relacionamentos semânticos existentes entre classes de associação e associações.

P8. Identificando Correspondências entre uma Classe de Associação e uma Associação

Contexto: Suponha o esquema **S** mostrado na figura 4.21. Analisando o esquema **S**, verificamos a presença da associação l_{AB} entre as classes **A** e **B** participantes da classe de associação **D**. Essa situação sugere a ocorrência de um relacionamento semântico entre a associação l_{AB} e a classe de associação **D**.

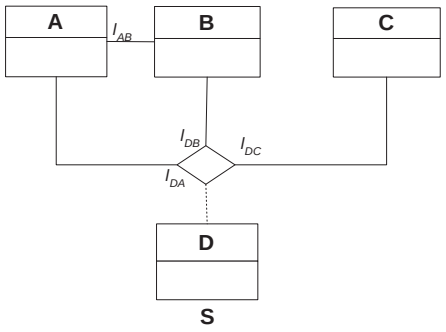

Figura 4.21: DE de Equivalência entre uma Classe de Associação e uma Associação

Problema: Verificar a existência de um relacionamento semântico entre a classe de associação **D** e a associação l_{AB} definida entre as classes **A** e **B** participantes da classe de associação **D**.

Solução: Deve-se verificar a semântica associada ao esquema **S**. Pela análise da semântica, existem dois tipos possíveis de relacionamento semântico entre l_{AB} e **D**:

1. Existe uma restrição de dependência existencial de subconjunto dada por $\mathtt{A}.(D.l_{DA})^{-1}.l_{DB} \subset \mathtt{A}.l_{AB}$, se a existência de uma instância na classe de associação **D**, associando uma instância **a** de **A** e uma instância **b** de **B**, requer a existência de uma instância na associação l_{AB} relacionando **a** e **b**.

2. Existe uma restrição de dependência existencial de equivalência dada por $\mathtt{A}.(D.l_{DA})^{-1}.l_{DB} \equiv \mathtt{A}.l_{AB}$, se existe uma instância na classe de associação **D** relacionando uma instância **a** de **A** e uma instância **b** de **B** se e somente se existe uma instância na associação l_{AB} relacionando **a** e **b**.

Exemplo:

- **Caso 1:** *Dependência existencial de subconjunto entre uma classe de associação e uma associação definida entre as classes participantes da classe de associação*

Considere o esquema $\mathbf{S_1}$ da figura 4.22. Verificamos a presença da associação *ensina* definida entre as classes **Professor** e **Disciplina** participantes da classe de associação **Matrícula**. Semanticamente, essa situação pode ter dois significados distintos: (i) a associação *ensina* e a classe de associação **Matrícula** não estão semanticamente relacionadas ou (ii) existe uma restrição de dependência existencial entre elas. Nesse padrão consideramos apenas o caso (ii).

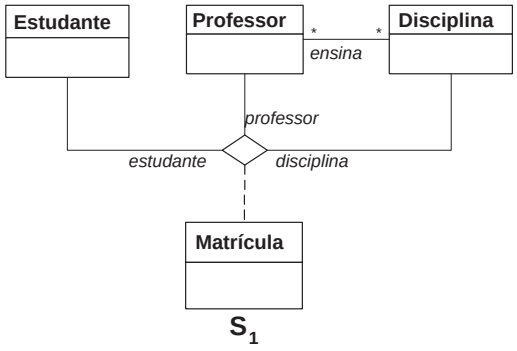

Figura 4.22: Exemplo - DE de Subconjunto entre uma Classe de Associação e uma Associação

Analisando a semântica do esquema $\mathbf{S_1}$ mostrado na figura 4.22, temos que um professor é associado a uma determinada disciplina em uma matrícula apenas se este professor é capaz de ensinar a disciplina referida. Portanto, identificamos uma relação de dependência semântica entre a classe de associação **Matrícula** e a associação *ensina*. Formalmente, temos que a existência de uma instância na classe de associação **Matrícula** associando uma disciplina **d** um estudante **e** e um professor **p**, requer a existência de uma instância na associação *ensina* relacionando um professor **p** e uma disciplina **d**. Essa restrição é capturada pela assertiva de dependência existencial de subconjunto $\mathtt{Professor}.(\mathtt{Matrícula}.professor)^{-1}.disciplina \subset \mathtt{Professor}.ensina$

- **Caso 2:** *Dependência existencial de equivalência entre a classe de associação e a associação definida entre as classes participantes da classe de associação*

No esquema $\mathbf{S_3}$ mostrado na figura 4.23, identificamos a associação *pertence* definida entre as classes **Empregado** e **Departamento** participantes da classe de associação **Alocação**. Pela semântica do mundo real, temos que existe uma instância na classe de associação **Alocação** relacionando um empregado **e** e um departamento **d** se e somente se existe uma instância na associação *pertence* relacionando um mesmo empregado **e** e um mesmo departamento **d**. Essa restrição é formalmente capturada pela assertiva de dependência existencial de equivalência $\texttt{Empregado}.(\texttt{Alocação}.\mathit{empregado})^{-1}.\mathit{departamento} \equiv \texttt{Empregado}.\mathit{pertence}$.

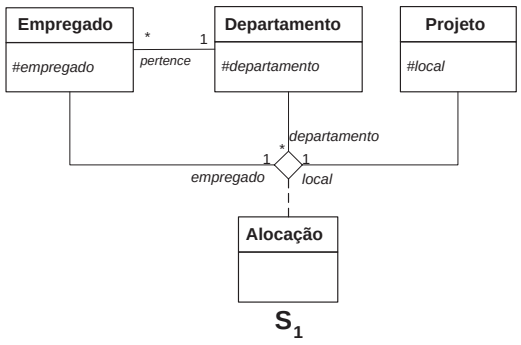

Figura 4.23: Exemplo - DE de Equivalência entre uma Classe de Associação e uma Associação

Usos Conhecidos: Relacionamentos semânticos entre componentes de um esquema conceitual é abordado em [SNL86].

Padrões Relacionados: Para capturar as restrições de dependência existencial identificadas neste padrão, sugerimos reestruturar o esquema conceitual como proposto nos padrões **P19** e **P20**.

Vale a pena ressaltar que uma classe de associação pode ser semanticamente equivalente à composição das associações que relacionam as classes participantes da classe de associação. Nesses casos, dizemos que a classe de associação é redundante ou derivada. No padrão **P9**, mostramos como identificar esse tipo de equivalência semântica a partir da análise do esquema conceitual.

P9. Identificando Classes de Associação Derivadas

Contexto: Em algumas situações verificamos ser possível inferir as instâncias de uma classe de associação a partir das instâncias das associações existentes entre as classes que compõem a classe de associação. Nesses casos, dizemos que a classe de associação é derivada ou redundante. A presença de classes de associação derivadas representa a existência de redundância no esquema conceitual, podendo vir a causar problemas de projeto durante a fase de implementação do banco de dados, tais como, anomalias de atualização e inconsistências. Neste padrão ajudamos o projetista na identificação das classes de associação derivadas a partir da análise do esquema conceitual.

Suponha o esquema **S** mostrado figura 4.24.

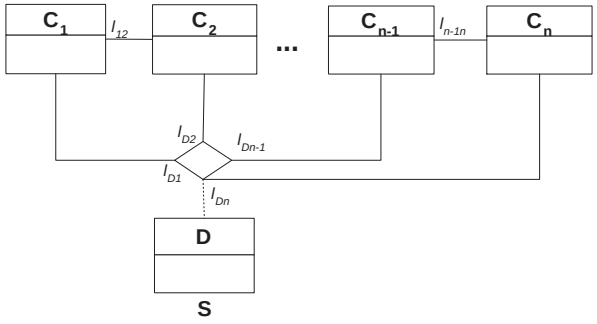

Figura 4.24: Classe de Associação Derivada

Problema: Verificar se a classe de associação **D** é derivável da composição das ligações l_{12}, ..., l_{n-1n}.

Solução: A classe de associação **D** é derivável da composição dos caminhos l_{12}, ..., l_{n-1n}, se para qualquer instância **d** de **D**, onde $\mathbf{d}.l_{D1} = \mathbf{c_1}$, $\mathbf{d}.l_{D2} = \mathbf{c_2}$, ..., $\mathbf{d}.l_{Dn} = \mathbf{c_n}$, então $\mathbf{c_2} \in \mathbf{c_1}.l_{12}$, ..., $\mathbf{c_n} \in \mathbf{c_{n-1}}.l_{n-1n}$.

Exemplo: Inicialmente devemos identificar no esquema conceitual classes de associação cujas classes participantes estejam interligadas entre si através de associações. Suponha o esquema $\mathbf{S_1}$ mostrado na figura 4.25. As classes **Projeto**, **Peça** e **Fornecedor**, participantes da classe de associação **Fornecimento**, estão interligadas entre si através das ligações *usa* e *é-fornecida*. O próximo passo é verificar se de fato a classe de associação é semanticamente equivalente à composição das associações que relacionam suas classes participantes. Voltando ao exemplo do esquema $\mathbf{S_1}$, temos que a classe de associação **Fornecimento** é derivável da composição das ligações *usa* e *é-fornecida*, pois para toda

instância **i** de **Fornecimento**, onde **i**.*projeto* = **j**, **i**.*peça* = **p** e **i**.*fornecedor* = **f**, então **p** ∈ **j**.*usa* e **f** ∈ **p**.*é-fornecida*. Esta restrição é formalmente capturada pela assertiva de correspondência de equivalência caminhos `Fornecimento` ≡ `(usa • é-fornecida`.

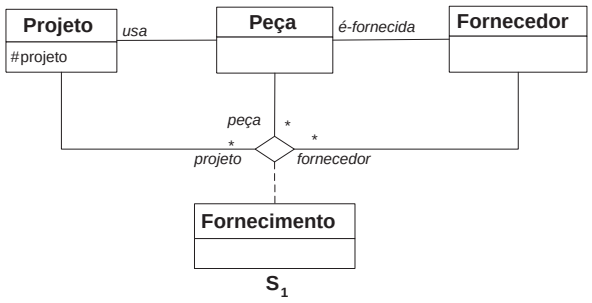

Figura 4.25: Exemplo - Classe de Associação Derivada

Usos Conhecidos: Uma discussão relacionada à existência de derivação em esquemas conceituais pode ser encontrada em [Fow97b].

Padrões Relacionados: No padrão **P16** mostramos como remover as classes de associação derivada identificadas no esquema conceitual.

C3. Padrões de Otimização

Nesta categoria, agrupamos os padrões que abordam a otimização dos esquemas conceituais.

C3.1 Reestruturação de Esquemas a partir das Assertivas de Correspondência

Como discutido na seção 2.3.4, as assertivas de correspondência são tipos especiais de restrições de integridade usadas para especificar a correspondência entre componentes de esquemas. Nesta seção, reunimos os padrões que reestruturam os esquemas tomando por base as assertivas de correspondência.

P10. Integrando Classes Equivalentes

Contexto: No padrão **P2**, mostramos como estabelecer assertivas de correspondência de equivalência entre extensões de classes de um ou mais esquemas conceituais.

Suponha o esquema **S** mostrado na figura 4.26.

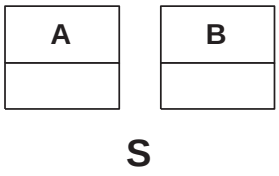

Figura 4.26: AC de Equivalência entre Extensões de Classes

Sejam **A** e **B** classes, `At(A)` o conjunto de atributos da classe **A**, `At(B)` o conjunto de atributos da classe **B**, `Lg(A)` o conjunto de ligações da classe **A** e `Lg(B)` o conjunto de ligações da classe **B**. Existe uma assertiva de correspondência de equivalência entre as extensões de **A** e **B** dada por $\mathbf{A} \equiv \mathbf{B}$.

Problema: Eliminar do esquema conceitual a redundância existente entre as extensões das classes **A** e **B**.

Solução: As classes **A** e **B** são integradas em uma nova classe **C**, como mostrado no esquema **S'** da figura 4.27.

Os atributos e ligações de **S'** são definidos como segue:

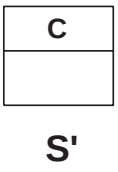

Figura 4.27: Removendo AC de Equivalência entre Extensões de Classes

- `At(C) = At(A) ∪ At(B)`, onde ∪ (união de conjuntos) integra todos os atributos sinônimos existentes entre as classes **A** e **B**.

- `Lg(C) = Lg(A) ∪ Lg(B)`, onde ∪ (união de conjuntos) integra todas as ligações sinônimas existentes entre as classes **A** e **B**.

O mapeamento entre o esquema original **S** e o esquema transformado **S'** é formalmente especificado pelas assertivas de correspondência abaixo:

(i) `A ≡ C`

(ii) `B ≡ C`

(iii) os atributos de **A** que não possuem atributos sinônimos em **B** são mapeados diretamente em **C**

(iv) os atributos de **B** que não possuem atributos sinônimos em **A** são mapeados diretamente em **C**

(v) os atributos de **A** que possuem atributos sinônimos em **B** são integrados em um único atributo em **C**

Exemplo: Suponha o esquema $\mathbf{S_1}$ mostrado na figura 4.28.

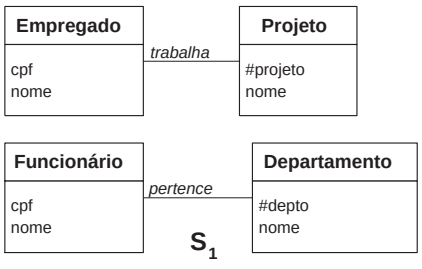

Figura 4.28: Exemplo - AC de Equivalência entre Extensões de Classes

De acordo com o padrão **P2**, no esquema $\mathbf{S_1}$ temos as assertivas de correspondência abaixo:

(i) `Empregado` $\equiv$ `Funcionário`

(ii) `Empregado.cpf` $\equiv$ `Funcionário.cpf`

(iii) `Empregado.nome` $\equiv$ `Funcionário.nome`

De forma a minimizar a redundância identificada, sugerimos reestruturar o esquema $\mathbf{S_1}$ como mostrado no esquema $\mathbf{S_2}$ da figura 4.29. No esquema $\mathbf{S_2}$, definimos uma nova classe **Empregado'** cujos atributos correspondem à união dos atributos das classes **Empregado** e **Funcionário** e cujas ligações correspondem à união das ligações das classes **Empregado** e **Funcionário**. O mapeamento entre o esquema transformado $\mathbf{S_2}$ e o esquema original $\mathbf{S_1}$ é formalmente especificado pelas assertivas de correspondência abaixo:

(i) `Empregado` $\equiv$ `Empregado'`

(ii) `Funcionário` $\equiv$ `Empregado'`

(iii) `Empregado.cpf` $\equiv$ `Empregado'.cpf`

(iv) `Empregado.nome` $\equiv$ `Empregado'.nome`

(iii) `Funcionário.cpf` $\equiv$ `Empregado'.cpf`

(iv) `Funcionário.nome` $\equiv$ `Empregado'.nome`

(v) `Empregado.`*trabalha* $\equiv$ `Empregado'.`*trabalha*

(v) `Funcionário.`*pertence* $\equiv$ `Empregado'.`*pertence*

Usos Conhecidos: Uma discussão relacionada a esse assunto pode ser encontrada em [SNL86].

Padrões Relacionados: No padrão **P2** mostramos como identificar assertivas de correspondência de equivalência entre extensões de classes.

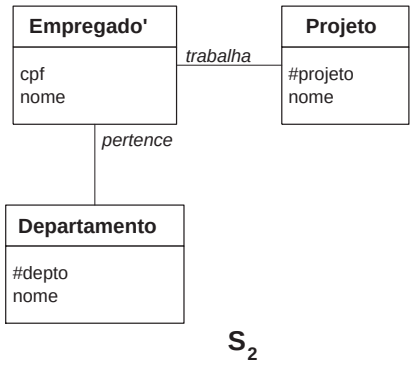

Figura 4.29: Exemplo - Capturando AC de Equivalência entre Extensões de Classes

P11. Eliminando Assertivas de Correspondência de Subconjunto entre Extensões de Classes

Contexto: No padrão **P2**, mostramos como identificar assertivas de correspondência de subconjunto entre extensões das classes de um ou mais esquemas conceituais. Nesse padrão propomos a reestruturação dos esquemas, de modo a capturar as assertivas de correspondência de subconjunto identificadas.

Suponha o esquema **S** mostrado na figura 4.30.

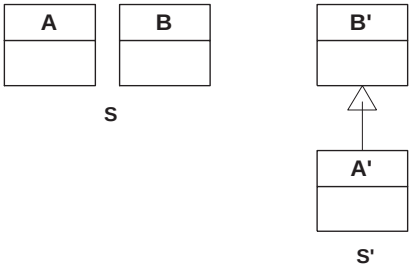

Figura 4.30: Eliminando AC de Subconjunto entre Extensões de Classes

Sejam **A** e **B** classes, `At(A)` o conjunto de atributos da classe **A**, `At(B)` o conjunto de atributos da classe **B**, `Lg(A)` o conjunto de ligações da classe **A** e `Lg(B)` o conjunto de ligações da classe **B**. Existe uma assertiva de correspondência de subconjunto entre as extensões de **A** e **B**, dada por $\texttt{A} \subset \texttt{B}$.

Problema: Remover do esquema conceitual a redundância existente entre as extensões das classes **A** e **B**.

Solução: O esquema **S** deve ser transformado no esquema **S'** como mostrado na figura 4.30. Os atributos e ligações de **A'** e **B'** são distribuídos como segue:

- `At(B') = At(B)`

- `At(A') = At(A) - At(B)`, onde - (diferença de conjuntos) significa que os atributos da classe **A'** são aqueles atributos de **A** que não possuem atributos semanticamente equivalentes em **B**

- `Lg(B') = Lg(B)`

- `Lg(A') = Lg(A) - Lg(B)`, onde - (diferença de conjuntos) significa que as ligações da classe **A'** são aquelas ligações de **A** que não possuem ligações semanticamente equivalentes em **B**

O mapeamento entre o esquema original **S** e o esquema transformado **S'** é formalmente especificado pelas assertivas de correspondência abaixo:

(i) `A` $\equiv$ `A'`

(ii) `B` $\equiv$ `B'`

(iii) os atributos de **B** são mapeados diretamente em atributos de **B'**

(iv) os atributos de **A** que não possuem atributos semanticamente equivalentes em **B** são mapeados diretamente em atributos de **A'**

(v) os atributos de **A** que possuem atributos semanticamente equivalentes em **B** são integrados em **C**

Exemplo: Suponha o esquema $\mathbf{S_1}$ mostrado na figura 4.31.

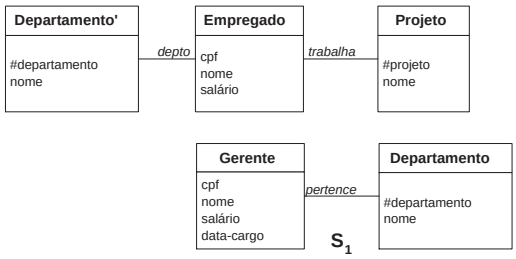

Figura 4.31: Exemplo - AC de Subconjunto entre Extensões de Classes

De acordo com o padrão **P2**, no esquema $\mathbf{S_1}$ temos as assertivas de correspondência abaixo:

(i) `Gerente` $\subset$ `Empregado`

(ii) `Gerente.cpf` $\equiv$ `Empregado.cpf`

(iii) `Gerente.nome` $\equiv$ `Empregado.nome`

(iv) `Gerente.salário` $\equiv$ `Empregado.salário`

(v) `Gerente.`*pertence* $\equiv$ `Empregado.`*depto*

De acordo com (i), temos uma generalização implícita entre as classes **Gerente** e **Empregado** que não está devidamente representada no esquema $\mathbf{S_1}$. Nesse caso, propomos reestruturar o esquema $\mathbf{S_1}$ como sugerido no esquema $\mathbf{S_2}$ da figura 4.32. No esquema $\mathbf{S_2}$, definimos duas novas classes **Empregado'** e **Gerente'** e definimos uma generalização entre essas classes. Os atributos e ligações da nova classe **Empregado'** correspondem aos atributos e ligações da classe **Empregado**. Os atributos da classe **Gerente'** correspondem aos atributos da classe **Gerente** que não possuem atributos semanticamente equivalentes em **Empregado**. As ligações de **Gerente'** correspondem às ligações de **Gerente** que não possuem ligações semanticamente equivalentes em **Empregado**. O mapeamento entre o esquema original $\mathbf{S_1}$ e o esquema transformado $\mathbf{S_2}$ é formalmente especificado pelas assertivas de correspondência abaixo:

(i) `Empregado` $\equiv$ `Empregado'`

(ii) `Gerente` $\equiv$ `Gerente'`

(iii) `Empregado.cpf` $\equiv$ `Empregado'.cpf`

(iv) `Empregado.nome` $\equiv$ `Empregado'.nome`

(v) `Empregado.salário` $\equiv$ `Empregado'.salário`

(vi) `Gerente.data-cargo` $\equiv$ `Gerente'.data-cargo`

(vii) `Empregado.`*trabalha* $\equiv$ `Empregado'.`*trabalha*

(viii) `Gerente.`*pertence* $\equiv$ `Empregado'.`*depto*

(ix) `Gerente.cpf` $\equiv$ `Empregado'.cpf`

(x) `Gerente.nome` $\equiv$ `Empregado'.nome`

(xi) `Gerente.salário` $\equiv$ `Empregado'.salário`

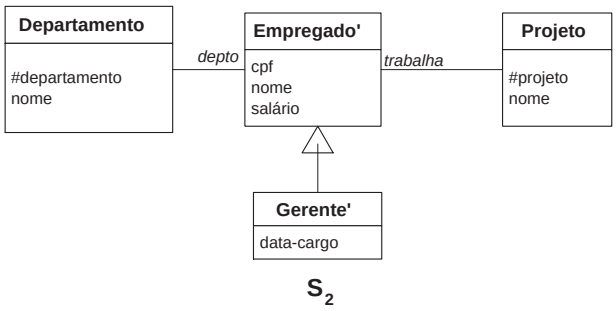

Figura 4.32: Exemplo - Eliminando AC de Subconjunto entre Extensões de Classes

Usos Conhecidos: Uma discussão relacionada a esse assunto pode ser encontrada em [SNL86].

Padrões Relacionados: No padrão **P2** mostramos como identificar assertivas de correspondência de subconjunto entre extensões de classes.

P12. Representando Correspondências entre uma Classe de Associação e uma Classe Participante da Classe de Associação

Contexto: No padrão **P5**, mostramos que uma classe de associação pode ser semanticamente equivalente a uma de suas classes participantes ou ser uma subclasse de uma de suas classes participantes.

Suponha o esquema **S** da figura 4.33. Seja **D** uma classe de associação e **A** uma classe participante de **D**.

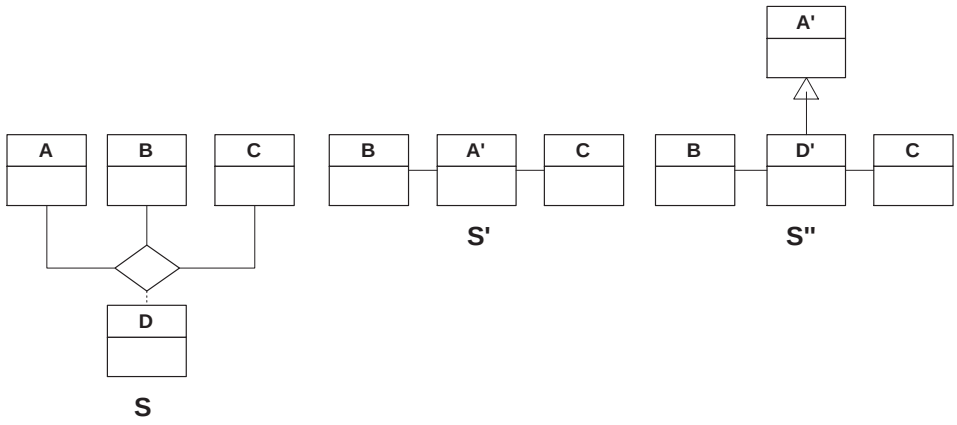

Figura 4.33: Capturando Relacionamento Semântico em uma Classe de Associação

Problema: Reestruturar o esquema conceitual a partir da correspondência semântica existente entre a classe de associação **D** a classe **A** participante da classe de associação **D**.

Solução:

1. Se existe uma assertiva de correspondência de equivalência de extensão entre **D** e **A** dada por **D** $\equiv$ **A**, então o esquema **S** deve ser transformado no esquema **S'** como mostrado na figura 4.33. Os atributos e ligações do esquema **S'** são definidos como segue:

   - `At(A') = At(D)` $\cup$ `At(A)`, onde $\cup$ (união de conjuntos) integra todos os atributos sinônimos existentes entre **A** e **D**
   - `Lg(A') = Lg(D)` $\cup$ `Lg(A)`, onde $\cup$ (união de conjuntos) integra todas as ligações sinônimas existentes entre **A** e **D**

O mapeamento entre o esquema original **S** e o esquema transformado **S'** é formalmente especificado pelas assertivas de correspondência abaixo:

(i) `A` $\equiv$ `A'`

(ii) `D` $\equiv$ `A'`

(iii) os atributos a **A** são mapeados diretamente em atributos de **A'**

(iv) os atributos de **D** são mapeados diretamente em atributos de **A'**

2. Se existe uma assertiva de correspondência de subconjunto entre as extensões de **A** e **D** dada por $\mathbf{D} \subset \mathbf{A}$, então o esquema **S** deve ser transformado no esquema **S"** como mostrado na figura 4.33. Os atributos e ligações do esquema **S"** são distribuídos como segue:

   - `At(A') = At(A)`
   - `At(D') = At(D) - At(A)`, onde - (diferença de conjuntos) significa que os atributos da classe **D'** são aqueles atributos de **D** que não possuem atributos semanticamente equivalentes em **A**
   - `Lg(A') = Lg(A)`
   - `Lg(D') = Lg(D) - Lg(A)`, onde - (diferença de conjuntos) significa que as ligações da classe **D'** são aquelas ligações de **D** que não possuem ligações semanticamente equivalentes em **A**

   O mapeamento entre o esquema original **S** e o esquema transformado **S"** é formalmente especificado pelas assertivas de correspondência abaixo:

   (i) `A` $\equiv$ `A'`

   (ii) `D` $\equiv$ `D'`

   (iii) os atributos a **A** são mapeados diretamente em atributos de **A'**

   (iv) os atributos de **D** são mapeados diretamente em atributos de **D'**

Exemplo:

- **Caso 1:** *A classe de associação é semanticamente equivalente a uma de suas classes participantes*

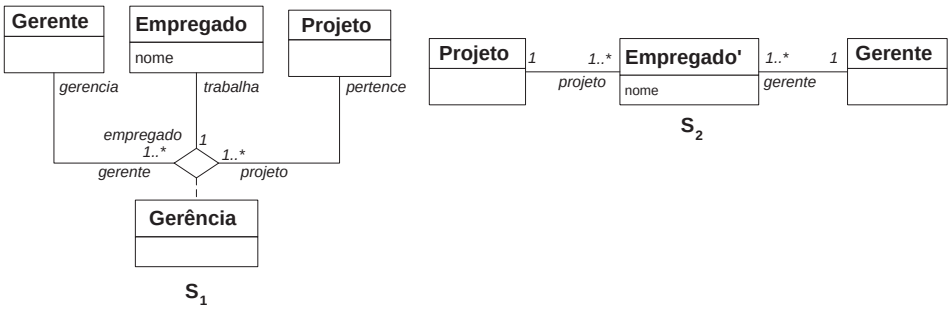

Figura 4.34: Exemplo - Representando AC de Equivalência em uma Classe de Associação

Suponha o esquema $\mathbf{S_1}$ mostrado na figura 4.34. De acordo com o padrão **P5** temos uma assertiva de correspondência de equivalência de extensão entre as classes **Empregado** e **Gerência** dada por `Empregado` $\equiv$ `Gerência`.

De modo a eliminar a redundância causada pela equivalência semântica existente entre as classes **Empregado** e **Gerência**, o esquema $\mathbf{S_1}$ deve ser reestruturado como mostrado no esquema $\mathbf{S_2}$ da figura 4.34. No esquema $\mathbf{S_2}$, definimos a classe **Empregado'** cujas instâncias correspondem à união das instâncias das classes **Empregado** e **Gerência**. A classe **Empregado'** deve conter todas as ligações e atributos da classe de associação **Gerência**. O mapeamento entre o esquema original $\mathbf{S_1}$ e o esquema transformado $\mathbf{S_2}$ é formalmente especificado pelas assertivas de correspondência abaixo:

(i) `Empregado` $\equiv$ `Empregado'`

(ii) `Gerência` $\equiv$ `Empregado'`

(iii) `Gerência`.*gerente* $\equiv$ `Empregado'`.*gerente*

(iv) `Gerência`.*projeto* $\equiv$ `Empregado'`.*projeto*

(v) `Gerência`.*empregado* $\equiv$ `Empregado'.nome`

- **Caso 2:** *A classe de associação é uma subclasse de uma de suas classes participantes*

Suponha o esquema $\mathbf{S_1}$ da figura 4.35. De acordo com o padrão **P5**, temos uma assertiva de correspondência de subconjunto de extensão entre as classes **Orientação** e **Estudante** dada por `Orientação` $\subset$ `Estudante`.

Para remover a redundância identificada, o esquema $\mathbf{S_1}$ deve ser reestruturado como sugerido no esquema $\mathbf{S_2}$ da figura 4.35. No esquema $\mathbf{S_2}$, definimos duas novas classes **Orientando** e **Empregado'**. As instâncias de **Orientando** correspondem às instâncias da classe **Estudante** que estão relacionadas com um projeto e com um professor. Em

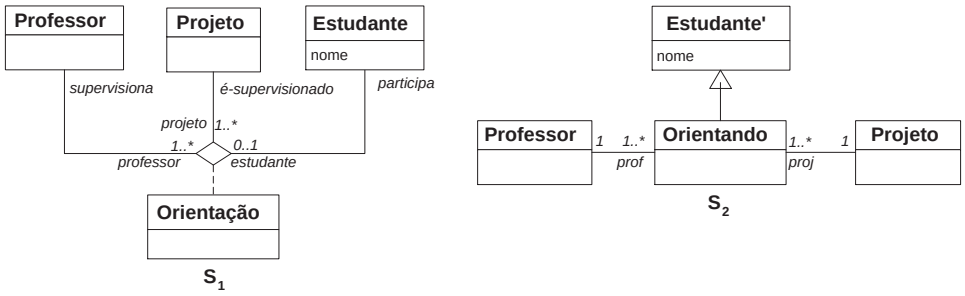

Figura 4.35: Exemplo - Representando AC de Subconjunto em uma Classe de Associação

seguida, estabelecemos uma generalização entre as classes **Estudante'** e **Orientando**. A nova classe **Orientando** possui todas as ligações e atributos da classe de associação **Orientação**.

Analisando os esquemas $\mathbf{S_1}$ e $\mathbf{S_2}$, verificamos ser possível estabelecer um mapeamento um para um entre as instâncias das classes **Orientando** e **Orientação** e, portanto, definir uma função bijetiva f entre suas extensões dada por,

$$f\text{: } \mathbf{Orientando} \rightarrow \mathbf{Orientação}.$$

Isso significa que a classe de associação **Orientação** e a classe **Orientando** são semanticamente equivalentes (`Orientando` $\equiv$ `Orientação`). O mapeamento entre o esquema original $\mathbf{S_1}$ e o esquema transformado $\mathbf{S_2}$ é formalmente especificado pelas assertivas de correspondência abaixo:

(i) `Orientação` $\equiv$ `Orientando`

(ii) `Estudante` $\equiv$ `Estudante'`

(iii) `Orientando` $\subset$ `Estudante'`

(iv) `Orientação`.*professor* $\equiv$ `Orientando`.*prof*

(v) `Orientação`.*projeto* $\equiv$ `Orientando`.*proj*

(vi) `Orientação`.*estudante* $\equiv$ `Estudante'.nome`

Usos Conhecidos: [DSB97] aborda a reestruturação de relacionamentos n-ários no contexto do modelo ER.

Padrões Relacionados: No padrão **P5** discutimos como identificar assertivas de correspondência entre extensões de uma classe de associação e uma das classes participantes da classe de associação.

P13. Integrando Associações Equivalentes

Contexto: Conforme discutido no padrão **P6**, podemos identificar assertivas de correspondência de equivalência de caminho entre associações que relacionam um conjunto de classes em comum.

Suponha o esquema **S** mostrado na figura 4.36. Sejam **A** e **B** classes e l_{AB} e l'_{AB} associações definidas entre as classes **A** e **B**. Temos que $l_{AB} \equiv l'_{AB}$.

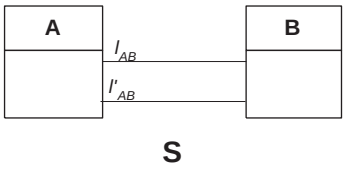

Figura 4.36: AC de Equivalência entre Associações

Problema: Representar adequadamente no esquema conceitual a assertiva de correspondência de equivalência de caminho existente entre as associações $l_{AB} \equiv l'_{AB}$.

Solução: O esquema **S** deve ser transformado no esquema **S'** como mostrado na figura 4.37. No esquema **S'**, selecionamos a associação l para permanecer no esquema integrado.

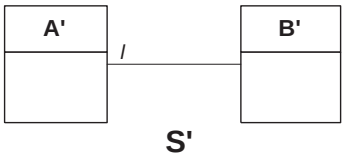

Figura 4.37: Removendo AC de Equivalência entre Associações

O mapeamento entre o esquema original **S** e o esquema transformado **S'** é formalmente especificado pelas assertivas de correspondência abaixo:

(i) `A` $\equiv$ `A'`

(ii) `B` $\equiv$ `B'`

(iii) `A`$.l_{AB} \equiv$ `A'`$.l$

(iv) `A`$.l'_{AB} \equiv$ `A'`$.l$

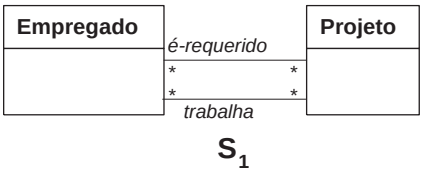

Figura 4.38: Exemplo - AC de Equivalência entre Associações

Exemplo: Considere o esquema $\mathbf{S_1}$ da figura 4.38.

De acordo com o padrão **P6**, temos que *trabalha* $\equiv$ *é-requerido*. Para eliminar a redundância identificada, sugerimos reestruturar o esquema $\mathbf{S_1}$ como sugerido no esquema $\mathbf{S_2}$ da figura 4.39. No esquema $\mathbf{S_2}$ selecionamos a associação *trabalha* para permanecer no esquema integrado.

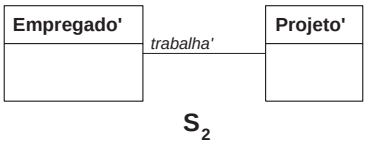

Figura 4.39: Exemplo - Representando AC de Equivalência entre Associações

O mapeamento entre o esquema original $\mathbf{S_1}$ e o esquema transformado $\mathbf{S_2}$ é formalmente especificado pelas assertivas de correspondência abaixo:

(i) `Empregado` $\equiv$ `Empregado'`

(ii) `Projeto` $\equiv$ `Projeto'`

(iii) `Empregado`. *é-requerido* $\equiv$ `Empregado'`. *trabalha'*

(iv) `Empregado`. *trabalha* $\equiv$ `Empregado'`. *trabalha'*

Usos Conhecidos: Equivalência de relacionamentos é abordado no contexto do modelo ER em [SNL86].

Padrões Relacionados: No padrão **P6** discutimos como identificar restrições de dependência existencial entre associações.

P14. Eliminando Assertivas de Correspondência de Subconjunto entre Associações

Contexto: Conforme discutido no padrão **P6**, podemos identificar assertivas de correspondência de subconjunto entre associações que relacionam um conjunto de classes em comum.

Suponha o esquema **S** mostrado na figura 4.40. Sejam **A** e **B** classes e l_{AB} e l'_{AB} associações definidas entre as classes **A** e **B**. Temos que $l_{AB} \subset l'_{AB}$.

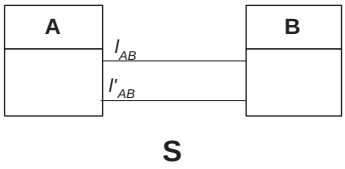

Figura 4.40: AC de Subconjunto entre Associações

Problema: Representar adequadamente no esquema conceitual a assertiva de correspondência de subconjunto existente entre as ligações l_{AB} e l'_{AB}.

Solução: O esquema **S** deve ser transformado no esquema **S'** como mostrado na figura 4.41. No esquema **S'**, duas novas classes de associação $\mathbf{L_{AB}}$ e $\mathbf{L'_{AB}}$ foram criadas, tal que a extensão da classe de associação $\mathbf{L_{AB}}$ é uma especialização da extensão da classe de associação $\mathbf{L'_{AB}}$.

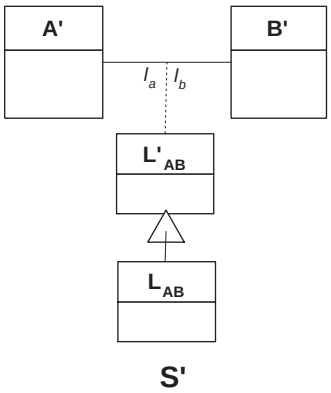

Figura 4.41: Removendo AC de Subconjunto entre Associações

O mapeamento entre o esquema original $\mathbf{S}$ e o esquema transformado $\mathbf{S}'$ é formalmente especificado pelas assertivas de correspondência abaixo:

(i) $\texttt{A} \equiv \texttt{A'}$

(ii) $\texttt{B} \equiv \texttt{B'}$

(iii) $\texttt{A}.l_{AB} \equiv \texttt{A'}.(L_{AB}.l_a)^{-1}$

(iv) $\texttt{A}.l'_{AB} \equiv \texttt{A'}.(L'_{AB}.l_a)^{-1}$

Exemplo: Considere o esquema $\mathbf{S_1}$ mostrado na figura 4.42. De acordo com o padrão **P6**, temos que `ensina` $\subset$ `pode-ensinar`. Essa restrição mostra a existência de uma generalização implícita entre as associações *ensina* e *pode-ensinar* que não está capturada no esquema $\mathbf{S_1}$.

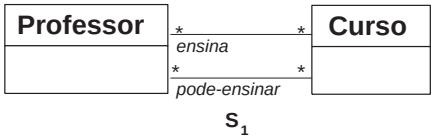

Figura 4.42: Exemplo - AC de Subconjunto entre Associações

Para capturar a restrição identificada, sugerimos reestruturar o esquema $\mathbf{S_1}$ como mostrado no esquema $\mathbf{S_2}$ da figura 4.43. No esquema $\mathbf{S_2}$, as associações *ensina* e *pode-ensinar* estão representadas como classes de associação. Além disso, para capturar a generalização existente entre as associações, definimos um relacionamento de generalização entre as novas classes de associação **Pode-ensinar** e **Ensina**.

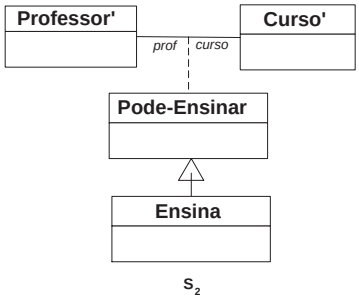

Figura 4.43: Exemplo - Representando AC de Subconjunto entre Associações

O mapeamento entre o esquema original $\mathbf{S_1}$ e o esquema transformado $\mathbf{S_2}$ é formalmente especificado pelas assertivas de correspondência abaixo:

(i) `Professor` $\equiv$ `Professor'`

(ii) `Curso` $\equiv$ `Curso'`

(iii) `Professor`$.$*pode-ensinar* $\equiv$ `Professor'.(Pode-Ensinar.`*prof*$)^{-1}$

(iv) `Professor`$.$*ensina* $\equiv$ `Professor'.(Ensina.`*prof*$)^{-1}$

Usos Conhecidos: A identificação de correspondências semânticas entre relacionamentos é abordada em [SNL86].

Padrões Relacionados: No padrão **P6** discutimos como identificar restrições de dependência existencial entre associações.

P15. Removendo Associações Derivadas

Contexto: No padrão **P7**, mostramos como identificar a existência de associações derivadas da composição de duas ou mais associações.

Suponha o esquema **S** da figura 4.44.

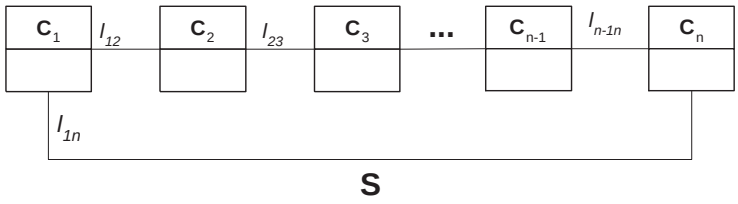

Figura 4.44: Associação Derivada

Pela análise da semântica de **S**, temos que l_{1n} é derivada da composição dos caminhos $l_{12} \bullet ... \bullet l_{n-1n}$ ($l_{1n} \equiv l_{12} \bullet ... \bullet l_{n-1n}$).

Problema: Reestruturar o esquema conceitual de modo a remover a associação derivada l_{1n}. As associações derivadas representam informação redundante que pode ou não permanecer no esquema conceitual, dependendo da implementação escolhida pelo projetista.

Solução: O esquema **S** deve ser transformado no esquema **S'** como mostrado na figura 4.45.

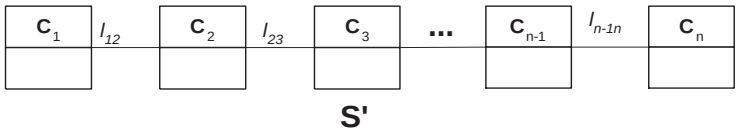

Figura 4.45: Removendo Associação Derivada

O mapeamento entre o esquema original **S** e o esquema transformado **S'** é formalmente especificado pelas assertivas de correspondência abaixo.

(i) $C_i \equiv C'_i$, $\mathtt{1} \leq \mathtt{i} \leq \mathtt{n}$

(ii) $C_1.l_{1n} \equiv C'_1.(l_{12} \bullet ... \bullet l_{n-1n})$

(iii) $C_i.l_{ii+1} \equiv C'_i.l_{ii+1}$, $\mathtt{1} \leq \mathtt{i} \leq \mathtt{n}$

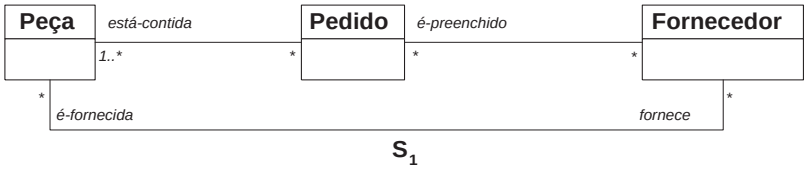

Figura 4.46: Exemplo - Associação Derivada

Exemplo: Considere o esquema $\mathbf{S_1}$ mostrado na figura 4.46. De acordo com o padrão **P7**, temos que a associação *é-fornecida* é derivável da composição das associações *está-contida* e *é-preenchido* (`é-fornecida` $\equiv$ `está-contida` $\bullet$ `é-preenchido`).

De forma a remover a redundância identificada, sugerimos reestruturar o esquema $\mathbf{S_1}$ como mostrado no esquema $\mathbf{S_2}$ da figura 4.47. No esquema $\mathbf{S_2}$, removemos a associação *é-fornecida* do esquema, já que esta pode ser obtida a partir da composição das associações *está-contida* e *é-preenchido*. Vale a pena ressaltar que o projetista pode decidir permanecer com a associação redundante no esquema durante a implementação do banco de dados por motivos relacionados ao desempenho das operações a serem executadas sobre o banco de dados.

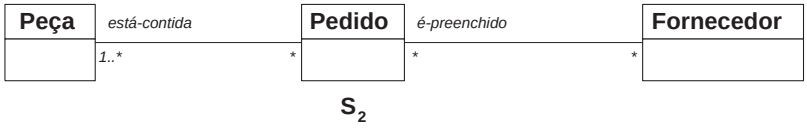

Figura 4.47: Exemplo - Removendo Associação Derivada

O mapeamento entre o esquema original $\mathbf{S_1}$ e o esquema transformado $\mathbf{S_2}$ é formalmente especificado pelas assertivas de correspondência abaixo:

(i) `Peça` $\equiv$ `Peça'`

(ii) `Pedido` $\equiv$ `Pedido'`

(iii) `Fornecedor` $\equiv$ `Fornecedor'`

(iv) `Peça`. *é-fornecida* $\equiv$ `Peça'`.(*está-contida* $\bullet$ *é-preenchida*)

(v) `Peça`. *está-contida* $\equiv$ `Peça'`. *está-contida*

(vi) `Pedido`. *é-preenchido* $\equiv$ `Pedido'`. *é-preenchido*

Usos Conhecidos: [Fow97b] aborda o problema de associações derivadas no esquema conceitual.

Padrões Relacionados: No padrão **P7** discutimos como identificar associações derivadas no esquema conceitual.

P16. Removendo Classes de Associação Derivadas

Contexto: No padrão **P9** discutimos como identificar classes de associação derivadas a partir da análise da semântica do esquema conceitual.

Suponha o esquema **S** mostrado figura 4.48.

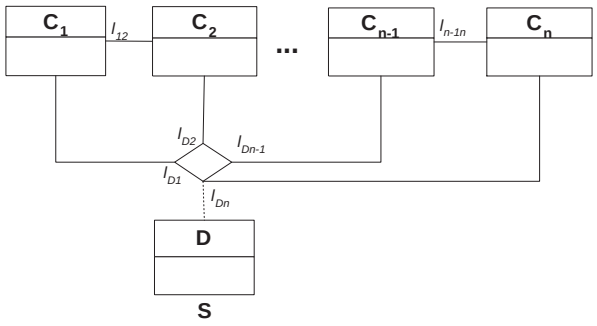

Figura 4.48: Classe de Associação Derivada

Pela análise da semântica de **S** temos que a classe de associação **D** é derivável da composição dos caminhos l_{12}, ..., l_{n-1n}.

Problema: Remover do esquema conceitual a classe de associação derivada **D**. A presença de classes de associação derivadas representa a existência de redundância no esquema conceitual.

Solução: O esquema **S** deve ser transformado no esquema **S'** como mostrado na figura 4.49.

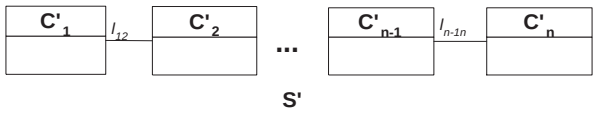

Figura 4.49: Removendo Classe de Associação Derivada

O mapeamento entre o esquema original **S** e o esquema transformado **S'** é formalmente especificado pelas assertivas de correspondência abaixo:

(i) $C_i \equiv C'_i$, $\mathtt{1} \leq \mathtt{i} \leq \mathtt{n}$

(ii) $C_i.l_{ii+1} \equiv C'_i.l_{ii+1}$, $\mathtt{1} \leq \mathtt{i} \leq \mathtt{n}$

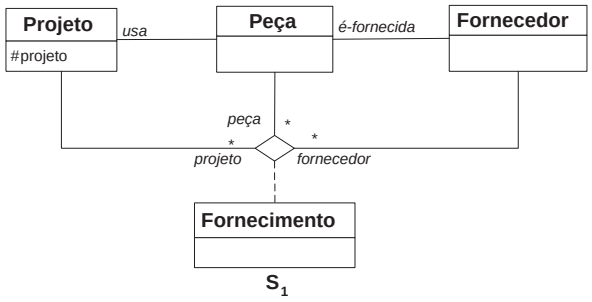

Figura 4.50: Exemplo - Classe de Associação Derivada

Exemplo: Suponha o esquema $\mathbf{S_1}$ mostrado na figura 4.50.

De acordo com o padrão **P9** temos que a classe de associação **Fornecimento** é derivável da composição das ligações *usa* e *é-fornecida*. Para remover a redundância identificada, sugerimos reestruturar o esquema $\mathbf{S_1}$ como mostrado no esquema $\mathbf{S_2}$ da figura 4.51.

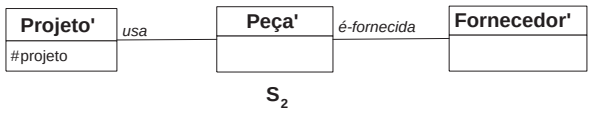

Figura 4.51: Exemplo - Removendo Classe de Associação Derivada

O mapeamento entre o esquema transformado $\mathbf{S_2}$ e o esquema original $\mathbf{S_1}$ é formalmente especificado pelas assertivas de correspondência abaixo:

(i) `Peça` $\equiv$ `Peça'`

(ii) `Projeto` $\equiv$ `Projeto'`

(iii) `Fornecedor` $\equiv$ `Fornecedor'`

(iv) `Fornecimento.`*fornecedor* $\equiv$ `Projeto'.(`*usa* $\bullet$ *é-fornecida*`)`

(v) `Fornecimento.`*peça* $\equiv$ `Projeto'.`*peça*

(vi) `Fornecimento.`*projeto* $\equiv$ `Projeto'.#projeto`

Usos Conhecidos: [Fow97b] aborda o problema de associações derivadas no esquema conceitual.

Padrões Relacionados: No padrão **P9** discutimos como identificar classes de associação derivadas no esquema conceitual.

C3.2 Reestruturação de Esquemas a partir das Restrições de Dependência Existencial

Esta seção reúne os padrões que reestruturam os esquemas conceituais com o objetivo de eliminar restrições de dependência existencial indesejadas.

P17. Integrando Classes de Associação Equivalentes

Contexto: No padrão **P4** discutimos em que circunstâncias podemos identificar restrições de dependência existencial de equivalência entre classes de associação.

Suponha o esquema **S** da figura 4.52.

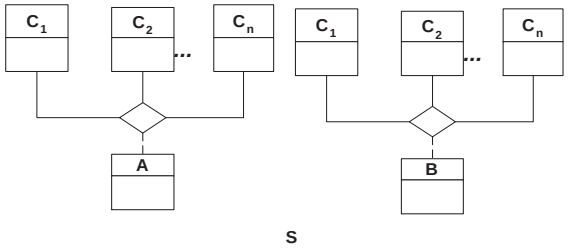

Figura 4.52: Exemplo - DE de Equivalência entre Classes de Associação

Sejam **A** e **B** classes de associação, $\mathbf{C_i}$, $1 \leq i \leq n$ classes participantes de **A** e **B**, `At(A)` o conjunto de atributos de **A**, `At(B)` o conjunto de atributos de **B**, `Lg(A)` o conjunto de ligações de **A** e `Lg(B)` o conjunto de ligações de **B**. Existe uma restrição de dependência existencial entre as classes de associação **A** e **B** dada por $\mathbf{A} \equiv \mathbf{B}$.

Problema: Remover do esquema conceitual a redundância causada pela restrição de dependência existencial de equivalência existente entre as classes de associação **A** e **B**.

Solução: O esquema **S** deve ser transformado no esquema **S'** como mostrado na figura 4.53.

Os atributos e ligações de **D** são distribuídos como segue:

- `At(D) = At(A)` $\cup$ `At(B)`, onde $\cup$ (união de conjuntos) integra os atributos sinônimos existentes entre **A** e **B**

- `Lg(D) = Lg(A)` $\cup$ `Lg(B)`, onde $\cup$ (união de conjuntos) integra as ligações sinônimas existentes entre **A** e **B**

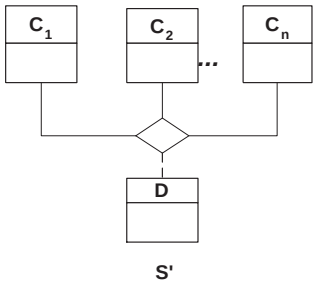

Figura 4.53: Exemplo - Removendo DE de Equivalência entre Classes de Associação

O mapeamento entre o esquema transformado **S'** e o esquema original **S** é formalmente especificado pelas assertivas de correspondência abaixo:

(i) `A` $\equiv$ `D`

(ii) `B` $\equiv$ `D`

(iii) os atributos de **A** que não possuam atributos sinônimos em **B** são mapeados diretamente em **D**

(iv) os atributos de **B** que não possuam atributos sinônimos em **A** são mapeados diretamente em **D**

(v) os atributos de **A** que possuam atributos sinônimos em **B** são integrados em um único atributo em **D**

Exemplo: Considere os esquemas $\mathbf{S_1}$ e $\mathbf{S_2}$ da figura 4.54. De acordo com o padrão **P4**, temos uma restrição de dependência existencial de equivalência entre as classes de associação **Alocação** e **Matrícula** dada por `Alocação[semestre, disciplina, professor]` $\equiv$ `Oferta[semestre, disciplina, professor]`.

De forma a eliminar a redundância identificada pela dependência existencial, devemos reestruturar os esquemas $\mathbf{S_1}$ e $\mathbf{S_2}$ como mostrado no esquema $\mathbf{S_3}$ da figura 4.55. No esquema $\mathbf{S_3}$, definimos uma nova classe de associação **Oferta'** cujos atributos correspondem à união dos atributos das classes de associação **Alocação** e **Oferta**, e cujas ligações correspondem à união das ligações das classes de associação **Alocação** e **Oferta**.

O mapeamento entre o novo esquema $\mathbf{S_3}$ e os esquemas originais $\mathbf{S_1}$ e $\mathbf{S_2}$ é formalmente especificado pelas assertivas de correspondência abaixo:

(i) `Oferta` $\equiv$ `Oferta'`

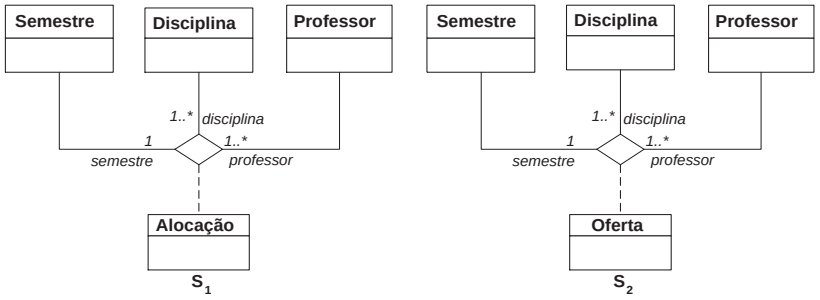

Figura 4.54: Exemplo - Dependência Existencial de Equivalência entre Classes de Associação

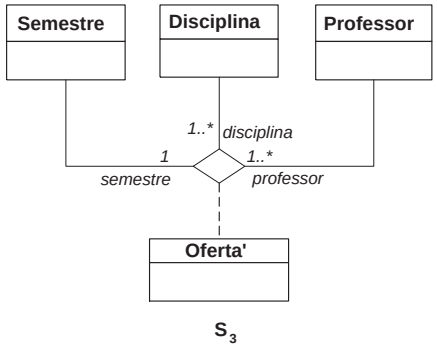

Figura 4.55: Exemplo - Capturando DE de Equivalência entre Classes de Associação

(ii) `Alocação` $\equiv$ `Oferta'`

(iii) `Alocação`.*semestre* $\equiv$ `Oferta'`.*semestre*

(iv) `Alocação`.*disciplina* $\equiv$ `Oferta'`.*disciplina*

(v) `Alocação`.*professor* $\equiv$ `Oferta'`.*professor*

(vi) `Oferta`.*semestre* $\equiv$ `Oferta'`.*semestre*

(vii) `Oferta`.*disciplina* $\equiv$ `Oferta'`.*disciplina*

(viii) `Oferta`.*professor* $\equiv$ `Oferta'`.*professor*

Usos Conhecidos: Restrições de dependência existencial são discutidas em [Vid94].

Padrões Relacionados: No padrão **P4** mostramos como identificar, a partir da análise da semântica do mundo real, as restrições de dependência existencial de equivalência entre classes de associação.

P18. Eliminando Dependências Existenciais de Subconjunto entre Classes de Associação

Contexto: No padrão **P4** discutimos em que circunstâncias podemos identificar restrições de dependência existencial de subconjunto entre classes de associação.

Suponha o esquema **S** mostrado na figura 4.56.

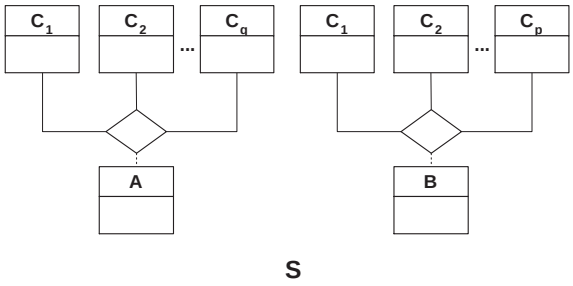

Figura 4.56: DE de Subconjunto entre Classes de Associação

Sejam **A** e **B** classes de associação, `At(A)` o conjunto de atributos de **A**, `At(B)` o conjunto de atributos de **B**, `Lg(A)` o conjunto de ligações de **A** e `Lg(B)` o conjunto de ligações de **B**. Existe uma restrição de dependência existencial entre **A** e **B** dada por `A` $\subset$ `B`.

Problema: Remover do esquema conceitual a redundância causada pela restrição de dependência existencial de subconjunto existente entre as classes de associação **A** e **B**.

Solução: O esquema **S** deve ser transformado no esquema **S'** como mostrado na figura 4.57.

Os atributos e ligações das novas classes de associação **A** e **B** são distribuídos como segue:

- `At(B') = At(B)`
- `At(A') = At(A)`
- `Lg(B') = Lg(B)`
- `Lg(A') = Lg(A) - Lg(B)`, onde - (diferença de conjuntos) significa que as ligações da classe **A'** são aquelas ligações de **A** que não possuem ligações semanticamente equivalentes em **B**.

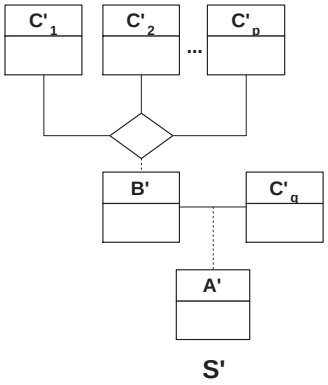

Figura 4.57: Removendo DE de Subconjunto entre Classes de Associação

O mapeamento entre o esquema original **S** e o esquema transformado **S'** é formalmente especificado pelas assertivas de correspondência abaixo:

(i) `A` $\equiv$ `A'`

(ii) `B` $\equiv$ `B'`

(iii) $C_i \equiv C'_i$, `1` $\leq$ `i` $\leq$ `n`

(iii) os atributos de **A** são mapeados diretamente em atributos de **A'**

(iv) os atributos de **B** são mapeados diretamente em atributos de **B'**

Exemplo: Considere os esquemas $\mathbf{S_1}$ e $\mathbf{S_2}$ da figura 4.58. De acordo com o padrão **P4** temos uma restrição de dependência existencial entre as classes de associação **Matrícula** e **Oferta** dada por `Matrícula[disciplina, professor]` $\subset$ `Oferta[disciplina, professor]`.

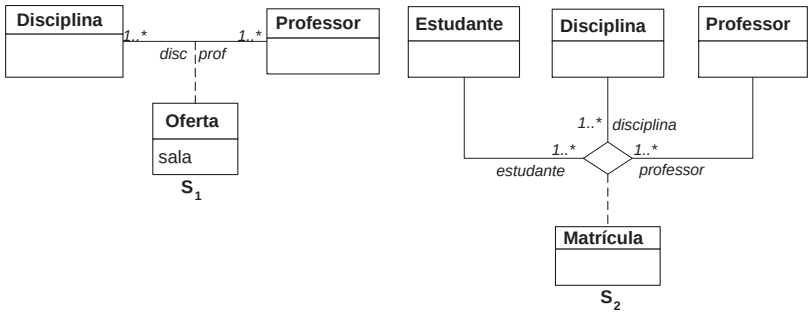

Figura 4.58: Exemplo - DE de Subconjunto entre Classes de Associação

Nesse caso, sugerimos reestruturar os esquemas $\mathbf{S_1}$ e $\mathbf{S_2}$ como mostrado no esquema $\mathbf{S_3}$ da figura 4.59. No esquema integrado $\mathbf{S_3}$, definimos as classes de associação **Matrícula'** e **Oferta'**. Os atributos e ligações da classe de associação **Oferta'** correspondem aos atributos e ligações da classe de associação **Oferta**. Os atributos da classe de associação **Matrícula'** correspondem aos atributos de **Matrícula**. As ligações de **Matrícula'** correspondem às ligações de **Matrícula** que não possuem ligações semanticamente equivalentes em **Oferta**.

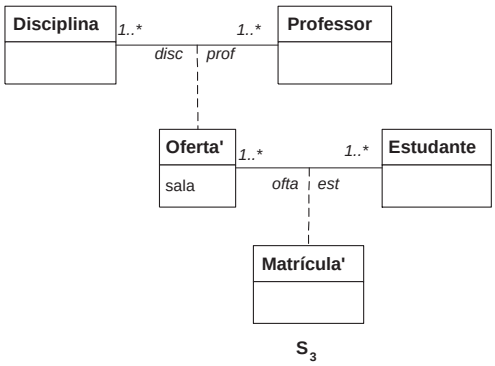

Figura 4.59: Exemplo - Representando DE de Subconjunto entre Classes de Associação

O mapeamento entre o novo esquema $\mathbf{S_3}$ e os esquemas originais $\mathbf{S_1}$ e $\mathbf{S_2}$ é formalmente especificado pelas assertivas de correspondência abaixo:

(i) `Oferta` $\equiv$ `Oferta'`

(ii) `Matrícula` $\equiv$ `Matrícula'`

(iii) `Matrícula`$.estudante \equiv$ `Matrícula'`$.est$

(iv) `Matrícula`$.disciplina \equiv$ `Matrícula'`$.ofta'.disc$

(v) `Matrícula`$.professor \equiv$ `Matrícula'`$.ofta'.prof$

(vi) `Oferta`$.disc \equiv$ `Oferta'`$.disc$

(vii) `Oferta`$.prof \equiv$ `Oferta'`$.prof$

Usos Conhecidos: Restrições de dependência existencial são abordadas em [Vid94].

Padrões Relacionados: No padrão **P4** mostramos como identificar, a partir da análise da semântica do mundo real, as restrições de dependência existencial entre classes de associação.

P19. Integrando Classes de Associação e Associações Equivalentes

Contexto: No padrão **P8**, identificamos restrições de dependência existencial de equivalência entre uma classe de associação e uma associação definida entre as classes participantes da classe de associação.

Suponha o esquema **S** mostrado na figura 4.60.

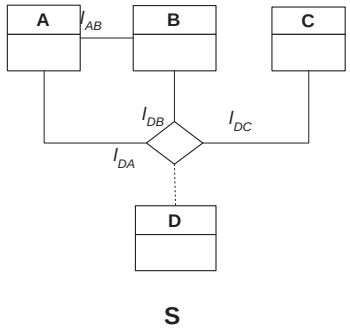

Figura 4.60: DE de Equivalência entre uma Classe de Associação e uma Associação

Existe uma restrição de dependência existencial de equivalência entre a classe de associação **D** e a associação l_{AB} dada por $\mathbf{A}.(\mathrm{D}.l_{DA})^{-1}.l_{DB} \equiv \mathbf{A}.l_{AB}$.

Problema: Representar no esquema conceitual a restrição de dependência existencial de equivalência identificada entre a classe de associação **D** e da associação l_{AB}.

Solução: O esquema **S** deve ser transformado no esquema **S'** como mostrado na figura 4.61.

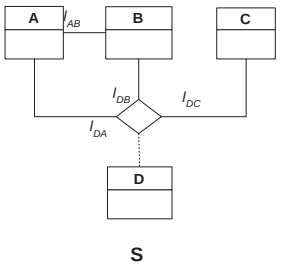

Figura 4.61: Removendo DE de Equivalência entre uma Classe de Associação e uma Associação

Os atributos e ligações do esquema **S'** devem ser distribuídos como se segue:

- At(D') = At(D)

- Lg(D') = Lg(D)

O mapeamento entre o esquema transformado **S'** e o esquema original **S** é formalmente especificado pelas assertivas de correspondência abaixo:

(i) D $\equiv$ D'

(ii) os atributos e ligações de **D** são mapeados diretamente em atributos e ligações de **D'**

Exemplo: Considere o esquema $\mathbf{S_1}$ mostrado na figura 4.62. Conforme discutido no padrão **P8**, temos que Professor.(Alocação.*empregado*)$^{-1}$.*departamento* $\equiv$ Empregado.*pertence*.

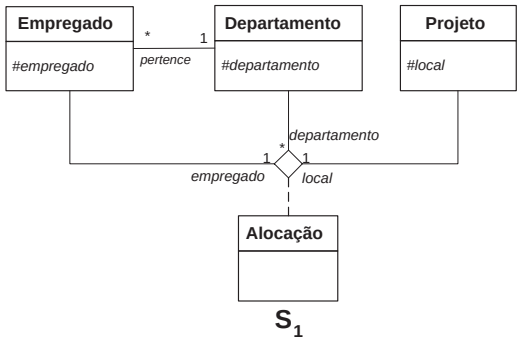

Figura 4.62: Exemplo - DE de Equivalência entre uma Classe de Associação e uma Associação

Nesse caso, propomos reestruturar o esquema $\mathbf{S_1}$ como sugerido no esquema $\mathbf{S_2}$ da figura 4.63. No esquema integrado $\mathbf{S_2}$, definimos a classe de associação **Alocação'** entre as classes participantes da associação *pertence* (**Empregado** e **Departamento**) e a classe **Projeto** passou a estar associada diretamente com a nova classe de associação **Alocação'**. Os atributos e ligações da classe de associação **Alocação'** correspondem aos atributos e ligações da classe de associação **Alocação**.

O mapeamento entre o esquema transformado $\mathbf{S_2}$ e o esquema original $\mathbf{S_1}$ é formalmente especificado pelas assertivas de correspondência abaixo:

(i) Alocação $\equiv$ Alocação'

(ii) Empregado.*pertence* $\equiv$ Empregado.(Alocação'.*emp*)$^{-1}$.*depto*

(iii) Alocação.*empregado* $\equiv$ Alocação'.*emp*

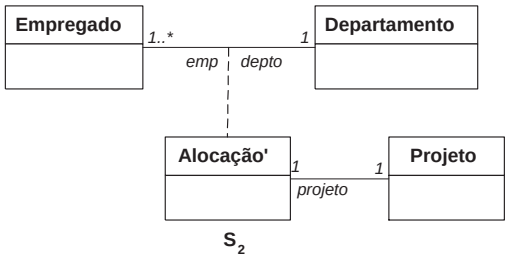

Figura 4.63: Exemplo - Representando DE de Equivalência entre uma Classe de Associação e uma Associação

(iv) `Alocação`.*departamento* $\equiv$ `Alocação'`.*depto*

(v) `Alocação`.*projeto* $\equiv$ `Alocação'`.*projeto*

Usos Conhecidos: Relacionamentos semânticos entre componentes de um esquema conceitual é abordado em [SNL86].

Padrões Relacionados: No padrão **P8** mostramos como identificar, a partir da análise do esquema conceitual e da semântica do mundo real, as restrições de dependência existencial entre classes de associação e associações.

P20. Eliminando Dependências Existenciais de Subconjunto entre uma Classe de Associação e uma Associação

Contexto: No padrão **P8**, identificamos restrições de dependência existencial de subconjunto entre uma classe de associação e uma associação definida entre as classes participantes da classe de associação.

Suponha o esquema **S** mostrado na figura 4.64.

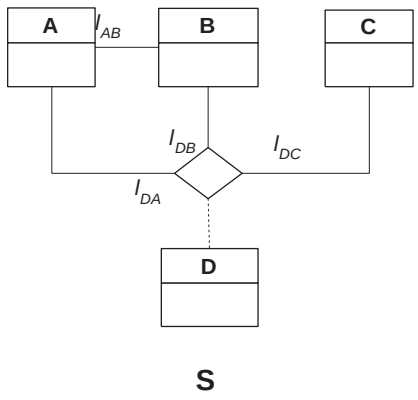

Figura 4.64: DE de Subconjunto entre uma Classe de Associação e uma Associação

Existe uma restrição de dependência existencial de subconjunto entre a classe de associação **D** e a associação l_{AB} dada por $\mathbf{A}.(\mathrm{D}.l_{DA})^{-1}.l_{DB} \subset \mathbf{A}.l_{AB}$.

Problema: Reestruturar o esquema conceitual de modo a capturar a restrição de dependência existencial de subconjunto identificada entre a classe de associação **D** e a associação l_{AB} definida entre as classes **A** e **B** participantes da classe de associação **D**.

Solução: O esquema **S** deve ser transformado no esquema **S'** como mostrado na figura 4.65.

Os atributos e ligações do esquema **S'** devem ser distribuídos como se segue:

- `At(D'') = At(D)`
- `Lg(D'') = Lg(D)` - l_{DC}

Exemplo: Considere o esquema $\mathbf{S_1}$ da figura 4.66.

De acordo com o padrão **P8**, temos que existe uma restrição de dependência existencial de subconjunto entre a classe de associação **Matrícula** e a associação *ensina* dada por `Professor`$.(\texttt{Matrícula}.professor)^{-1}.disciplina \subset$ `Professor`$.ensina$.

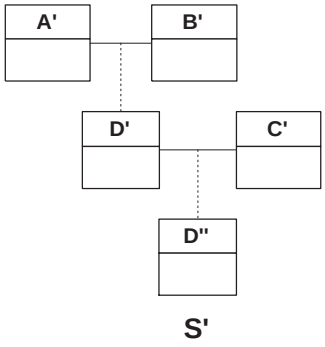

Figura 4.65: Removendo DE de Subconjunto entre uma Classe de Associação e uma Associação

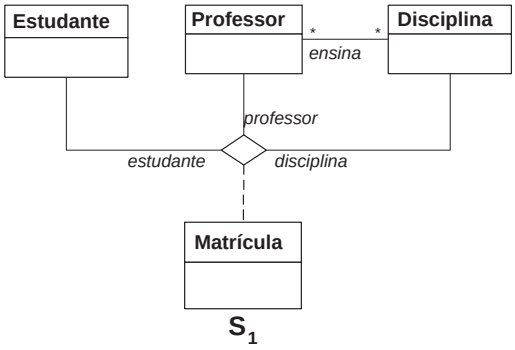

Figura 4.66: Exemplo - DE de Subconjunto entre uma Classe de Associação e uma Associação

De forma a capturar a restrição de dependência existencial de subconjunto entre a classe de associação **Matrícula** e a associação *ensina*, sugerimos reestruturar o esquema $\mathbf{S_1}$ como mostrado no esquema integrado $\mathbf{S_2}$ da figura 4.67. No esquema $\mathbf{S_2}$, inicialmente associamos as classes **Disciplina** e **Professor**, dando origem à classe de associação **Oferta**. Em seguida, associamos a classe **Estudante** à nova classe de associação **Oferta**, obtendo a classe de associação **Matrícula'**. As ligações da classe de associação **Oferta** correspondem às ligações da classe de associação **Matrícula** cujas classes participam da associação *ensina*. Os atributos da classe de associação **Matrícula'** correspondem aos atributos da classe de associação **Matrícula**. As ligações da classe de associação **Matrícula'** correspondem às ligações da classe de associação **Matrícula** que não possuem ligações semanticamente equivalentes em **Oferta**.

O mapeamento entre o esquema transformado $\mathbf{S_2}$ e o esquema original $\mathbf{S_1}$ é formalmente especificado pelas assertivas de correspondência abaixo:

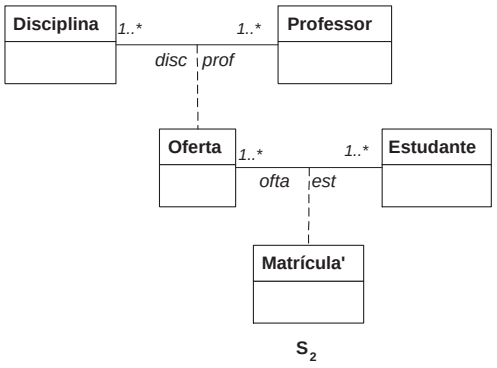

Figura 4.67: Exemplo - Representando DE de Subconjunto entre uma Classe de Associação e uma Associação

(i) `Matrícula` $\equiv$ `Matrícula'`

(ii) `Matrícula`$.\mathit{estudante} \equiv$ `Matrícula'`$.\mathit{est}$

(iii) `Matrícula`$.\mathit{disciplina} \equiv$ `Matrícula'`$.\mathit{ofta}.\mathit{disc}$

(iv) `Matrícula`$.\mathit{professor} \equiv$ `Matrícula'`$.\mathit{ofta}.\mathit{prof}$

(v) `Professor`$.\mathit{ensina} \equiv$ `Professor.(Oferta`$.\mathit{professor})^{-1}.\mathit{disciplina}$

Usos Conhecidos: Relacionamentos semânticos entre componentes de um esquema conceitual é abordado em [SNL86].

Padrões Relacionados: No padrão **P8** mostramos como identificar, a partir da análise do esquema conceitual e da semântica do mundo real, as restrições de dependência existencial entre classes de associação e associações.

P21. Adicionando Associações Ocultas

Contexto: No padrão **P3**, mostramos como identificar no esquema conceitual associações ocultas representadas por restrições de dependência existencial entre classes.

Suponha o esquema **S** da figura 4.68.

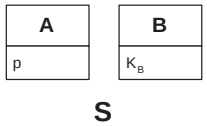

Figura 4.68: Associações Ocultas

Sejam **A** e **B** classes, `At(A)` o conjunto de atributos da classe **A**, `At(B)` o conjunto de atributos da classe **B** e $\mathbf{K_B}$ chave da classe **B**. Existe uma restrição de dependência existencial entre as classes **A** e **B** dada por $\mathbf{A[p]} \subset \mathbf{B[K_B]}$.

Problema: Reestruturar o esquema conceitual para capturar a restrição de dependência existencial identificada entre as classes **A** e **B**.

Solução: O esquema **S** deve ser transformado no esquema **S'** com mostrado na figura 4.69.

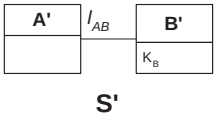

Figura 4.69: Associações Ocultas

Os atributos e ligações das classes **A'** e **B'** são definidos como segue:

- `At(A') = At(A) -` $\{\mathbf{p}\}$
- `At(B') = At(B)`
- `Lg(A') = Lg(A) +` l_{AB}
- `Lg(B') = Lg(B)`

O mapeamento entre o esquema original **S** e o esquema transformado **S'** é formalmente especificado pelas assertivas de correspondência abaixo:

(i) `A` $\equiv$ `A'`

(ii) `B` $\equiv$ `B'`

(iii) `A.p` $\equiv$ `A'`$.l_{AB}.K_B$

Exemplo: Suponha o esquema $\mathbf{S_1}$ mostrado na figura 4.70. De acordo com o padrão **P3**, temos que existe uma restrição de dependência existencial entre as classes **Livro** e **Autor** dada por `Livro[autor]` $\subset$ `Autor[nome]`.

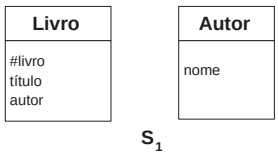

Figura 4.70: Exemplo - Identificando Associações Ocultas

De forma a capturar a restrição de dependência existencial identificada entre as classes **Livro** e **Autor**, propomos reestruturar o esquema $\mathbf{S_1}$ como mostrado no esquema $\mathbf{S_2}$ da figura 4.71. No esquema integrado $\mathbf{S_2}$, criamos duas novas classes **Livro'** e **Autor'** e definimos a associação *autor* entre elas. Os atributos da classe **Livro'** correspondem aos atributos da classe **Livro** que não estão semanticamente relacionados com a classe **Autor**.

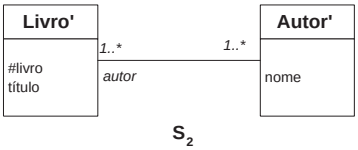

Figura 4.71: Exemplo - Adicionando Associações Ocultas

O relacionamento entre o esquema transformado $\mathbf{S_2}$ e o esquema original $\mathbf{S_1}$ é formalmente especificado pelas assertivas de correspondência abaixo:

(i) `Livro` $\equiv$ `Livro'`

(ii) `Autor` $\equiv$ `Autor'`

(iii) `Livro.autor` $\equiv$ `Livro'`$.autor.$`nome`

(iv) `Livro.#livro` $\equiv$ `Livro'.#livro`

(v) `Livro.título` $\equiv$ `Livro'.título`

Usos Conhecidos: Uma discussão relacionada a esse padrão pode ser encontrada em [Lós98] no enfoque do modelo ER.

Padrões Relacionados: No padrão **P3**, discutimos algumas regras que poderão auxiliar na identificação das restrições de dependência existencial entre classes.

C4. Padrões de Refinamento

Os padrões reunidos nessa categoria foram classificados de acordo com os diferentes tipos de situações de modelagem identificadas que indicavam a necessidade da reestruturação dos esquemas conceituais.

P22. Associações Mandatórias x Associações Opcionais

Contexto: Suponha o esquema $\mathbf{S_1}$ mostrado na figura 4.72. A multiplicidade mínima definida para a ligação *trabalha* é 1, significando que uma instância da classe **Empregado** *deve* estar associada a no mínimo uma instância da classe **Empresa**. Essas associações são denominadas associações *mandatórias*.

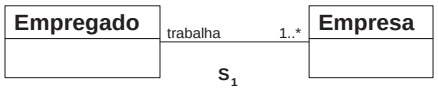

Figura 4.72: Exemplo - Associação Mandatória

As associações *opcionais*, por sua vez, modelam situações em que a multiplicidade mínima definida para a associação é 0. Por exemplo, no esquema $\mathbf{S_2}$ mostrado na figura 4.73, a multiplicidade mínima definida para a associação entre as classes **Aluno** e **Livro** é 0, logo, identificamos a existência de uma associação opcional. Isso significa que uma instância da classe **Aluno** *pode* estar associada a zero ou mais instâncias da classe **Livro**.

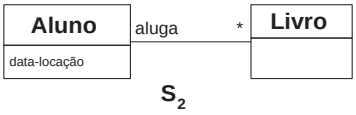

Figura 4.73: Exemplo - Associação Opcional

O uso de associações opcionais pode levar a ambigüidades e inconsistências no esquema conceitual, pois uma única representação é usada para descrever duas situações distintas: instâncias que participam da associação e instâncias que não participam da associação.

Suponha o esquema **S** da figura 4.74.

Sejam **A** e **B** classes, `At(A)` o conjunto de atributos da classe **A**, `At(B)` o conjunto de atributos da classe **B**. Identificamos uma associação opcional entre as classes **A** e **B**.

Problema: Eliminar do esquema conceitual a ambigüidade causada pela associação opcional existente entre as classes **A** e **B**.

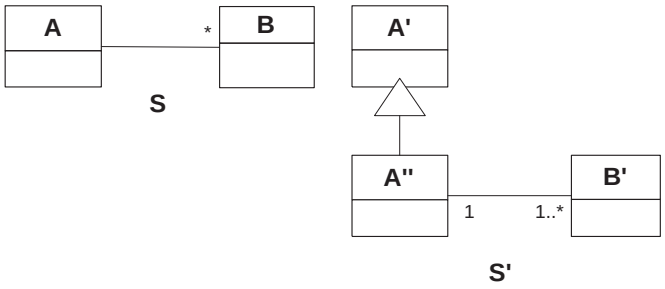

Figura 4.74: Associação Mandatória x Associação Opcional

Solução: O esquema **S** deve ser transformado no esquema **S'** como mostrado na figura 4.74. Os atributos e ligações no novo esquema **S'** são distribuídos como segue:

- `At(A'')` = atributos da classe **A** específicos para o relacionamento com a classe **B**

- `At(A') = At(A) - At(A'')`, onde - (diferença de conjuntos) especifica que os atributos de **A'** são aqueles atributos de **A** que não possuem atributos semanticamente equivalentes em **A"**

- `At(B') = At(B)`

- `Lg(A'') = Lg(A)`

- `Lg(B') = Lg(B)`

O mapeamento entre o esquema transformado **S'** e o esquema original **S** é formalmente especificado pelas assertivas de correspondência abaixo:

(i) `A` $\equiv$ `A'`

(ii) `B` $\equiv$ `B'`

(iii) os atributos da classe **A** específicos para o relacionamento com a classe **B** são mapeados diretamente para a classe **A"**

(iv) os demais atributos de **A** são mapeados diretamente em atributos de **A'**

(v) todos os atributos de **B** são mapeados diretamente em atributos de **B'**

Exemplo: Suponha o esquema $\mathbf{S_2}$ mostrado na figura 4.73. O valor da multiplicidade mínima definida para a ligação *aluga* é 0. Isso significa que um aluno *pode* alugar um livro. De acordo com a semântica do mundo real, sabemos que quando um aluno aluga um livro, este livro torna-se indisponível. Além disso, o aluno passa a ser responsável pela devolução do livro, ou seja, o estado das classes **Aluno** e **Livro** muda quando suas instâncias são associadas.

Em algumas situações, a representação mostrada no esquema da figura 4.73, além de representar duas situações distintas: instâncias que participam da associação com instâncias que não participam, também não consegue capturar a mudança de estado das classes **Aluno** e **Livro**.

Para solucionar os problemas identificados, o esquema $\mathbf{S_2}$ da figura 4.73 é reestruturado como mostrado no esquema $\mathbf{S_3}$ da figura 4.75. Nesta nova representação, criamos a classe **Locador**, cujas instâncias correspondem às instâncias da classe **Aluno** que participam de associações com instâncias da classe **Livro** e a classe **Livro-Alugado**, cujas instâncias correspondem às instâncias da classe **Livro** que participam de associações com instâncias da classe **Aluno** e definimos uma associação mandatória entre as classes criadas.

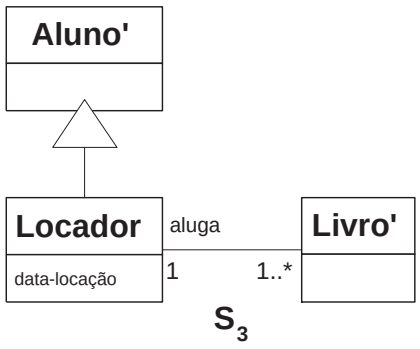

Figura 4.75: Exemplo - Associação Mandatória x Associação Opcional

O mapeamento entre o esquema transformado $\mathbf{S_2}$ e o esquema original $\mathbf{S_1}$ é formalmente especificado pelas assertivas de correspondência abaixo:

(i) `Livro` $\equiv$ `Livro'`

(ii) `Aluno` $\equiv$ `Aluno'`

(iii) `Aluno.data-locação` $\equiv$ `Locador.data-locação`

(iv) os demais atributos de **Aluno** são mapeados diretamente em atributos de **Aluno'**

(v) todos os atributos de **Livro** são mapeados diretamente em atributos de **Livro'**

Usos Conhecidos: [WSW99] discute o uso das associações opcionais no contexto da Ontologia.

P23. Classe de Associação x Classe

Contexto: Suponha o esquema **S** mostrado na figura 4.76.

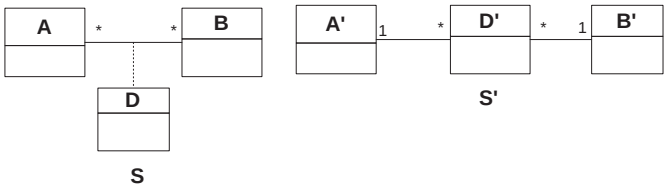

Figura 4.76: Associação m:n com Atributo

Seja **D** uma classe de associação e **A** e **B** classes participantes da classe de associação **D**. A semântica associada ao esquema **S** exige que uma determinada instância **a** de **A** esteja associada a uma mesma instância **b** de **B** mais de uma vez em **D**.

Problema: Mostrar que a classe de associação binária **D** deve ser modelada como uma classe.

Solução: O esquema **S** deve ser transformado no esquema **S'** como sugerido na figura 4.76. Os atributos e ligações da nova classe **D'** são definidos como segue:

- At(D') = At(D)

- Lg(D') = Lg(D)

O mapeamento entre o esquema transformado **S'** e o esquema original **S** é formalmente especificado pelas assertivas de correspondência abaixo:

(i) A $\equiv$ A'

(ii) B $\equiv$ B'

(iii) D $\equiv$ D'

Exemplo: Considere novamente o esquema $\mathbf{S_1}$ mostrado na figura **??**. Suponha que um estudante tenha sido reprovado em uma determinada disciplina e necessite cursar essa disciplina novamente. Isso implica que a classe de associação **Oferta** deveria conter mais de uma instância relacionando um estudante **e** com uma disciplina **d**. Como iremos

armazenar essa informação, já que, por definição, uma classe de associação não permite duplicação de instâncias relacionando os mesmos objetos?

Para solucionar o problema mencionado anteriormente, o esquema $\mathbf{S_1}$ da figura **??** deve ser reestruturado como mostrado no esquema $\mathbf{S_2}$ da figura 4.77. No esquema $\mathbf{S_2}$, a classe de associação **Oferta** foi promovida à classe **Oferta** e foi associada a cada uma das classes participantes da associação original através de uma ligação monovalorada. O esquema $\mathbf{S_2}$ permite que um estudante esteja associado a uma mesma disciplina mais de uma vez.

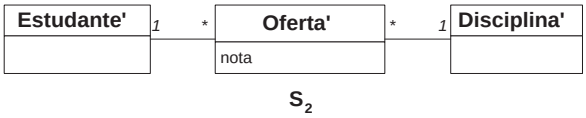

Figura 4.77: Exemplo - Classe x Classe de associação

O mapeamento entre o esquema transformado $\mathbf{S_2}$ e o esquema original $\mathbf{S_1}$ é formalmente especificado pelas assertivas de correspondência abaixo:

(i) `Oferta` $\equiv$ `Oferta'`

(ii) `Estudante` $\equiv$ `Estudante'`

(iii) `Disciplina` $\equiv$ `Disciplina'`

Usos Conhecidos: Uma discussão relacionada a esse padrão pode ser encontrada em [Fow97b].

P24. Adicionando Classes Descritivas

Contexto: A regra proposta nesse padrão é bastante útil no domínio das aplicações onde a descrição de um objeto é distinta do objeto em si. Uma classe descritiva é uma especificação ou descrição de outra classe. Em um esquema conceitual, dizemos que a classe descritiva **DescriçãoX** descreve a classe **X**.

Suponha o esquema **S** mostrado na figura 4.78. Seja **A** uma classe e $\mathbf{a_1}, \mathbf{a_2}, ...\mathbf{a_n}$ atributos descritivos da classe **A**.

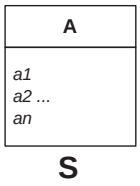

Figura 4.78: Classe Descritiva

Problema: Adicionar uma classe descritiva ao esquema conceitual **S**.

Solução: Quando a remoção das informações que elas descrevem resulta em perda da informação que precisa ser mantida, ou quando sua adição ocasiona uma redução da informação duplicada ou redundante. Nesse caso, o esquema **S** deve ser transformado no esquema **S'** como mostrado na figura 4.79.

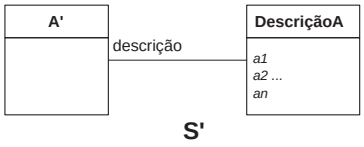

Figura 4.79: Adicionando Classe Descritiva

Os atributos e ligações no novo esquema **S'** são distribuídos como segue:

- `At(A') = At(A) -` $\{a_1, a_2, ...a_n\}$
- `At(DescriçãoA) =` $\{a_1, a_2, ...a_n\}$
- `Lg(A') = Lg(A) +` *descrição*

O relacionamento entre o esquema transformado **S'** e o esquema original **S** é especificado formalmente pelas assertivas de correspondência abaixo:

(i) `A` $\equiv$ `A'`

(ii) `A.`a_i `= A'.`*descrição*`.`a_i`,` $1 \leq i \leq n$

(iii) os demais atributos de **A** são mapeados diretamente em atributos de **A'**

Exemplo: Suponha o esquema $\mathbf{S_1}$ mostrado na figura 4.80. No esquema $\mathbf{S_1}$, verificamos que a classe **Livro** possui os atributos **#livro**, **descrição** e **preço**. Nessa representação, cada instância da classe **Livro** representa um livro físico, logo, sempre que um livro físico é vendido, sua instância correspondente na classe **Livro** é removida.

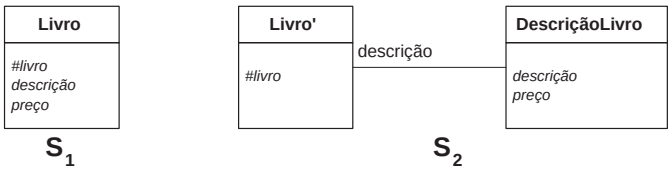

Figura 4.80: Exemplo - Classe Descritiva

Observe que o esquema $\mathbf{S_1}$, se implementado como descrito, possuirá dados duplicados, pois as informações relacionadas à descrição e ao preço serão repetidas para todas as instâncias do mesmo livro. Além disso, caso todos os volumes físicos de um determinado livro sejam vendidos, todas as suas instâncias serão removidas da classe **Livro**, implicando na perda de todas as suas informações descritivas.

A situação descrita ilustra a necessidade de reestruturarmos o esquema $\mathbf{S_1}$ conforme mostrado no esquema $\mathbf{S_2}$ da figura 4.80. No esquema $\mathbf{S_2}$, adicionamos uma classe descritiva, chamada **DescriçãoLivro**, para armazenar todas as informações descritivas dos livros. Uma instância da classe **DescriçãoLivro** não representa um livro físico e sim uma descrição ou informação sobre os livros. Finalmente, definimos a ligação *descrição* entre as classes **Livro** e **DescriçãoLivro**. Vale a pena ressaltar que mesmo se todos os livros físicos forem vendidos e as instâncias correspondentes forem removidas, as informações da classe **DescriçãoLivro** permanecem inalteradas, solucionando assim, o problema da perda de informação. Além disso, o uso dessa nova representação elimina o problema da duplicação dos dados detectado no esquema $\mathbf{S_1}$.

O mapeamento entre o esquema transformado $\mathbf{S_2}$ e o esquema original $\mathbf{S_1}$ é formalmente especificado pelas assertivas de correspondência abaixo:

(i) `Livro` $\equiv$ `Livro'`

(ii) `Livro.#livro` $\equiv$ `Livro'.#livro`

(iii) `Livro.descrição` $\equiv$ `Livro'.`*descrição*`.descrição`

(iv) `Livro.preço` $\equiv$ `Livro'.`*descrição*`.preço`

Usos Conhecidos: Uma discussão relacionada a esse padrão pode ser encontrada em [Lar97].

P25. Modelando Quantidades

Contexto: Como mostrado no esquema $\mathbf{S_1}$ da figura 4.81, a maneira mais comum e mais simples de representar atributos que armazenam quantidades é utilizando um valor numérico. Um problema em utilizar essa representação está no fato de um número não ser apropriado para representar toda a semântica do mundo real. O que significa dizer que o peso de uma pessoa é 60 ou que sua altura é 160? Para o número fazer sentido, nós precisamos das unidades. A regra descrita nesse padrão propõe uma representação alternativa e ao mesmo tempo mais eficiente para essas situações.

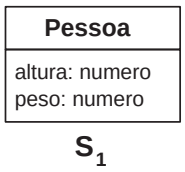

Figura 4.81: Exemplo - Modelagem Tradicional das Quantidades

Suponha o esquema **S** mostrado na figura 4.82. Seja **A** uma classe e $\mathbf{a_1}, \mathbf{a_2}, ...\mathbf{a_n}$ atributos numéricos de **A**.

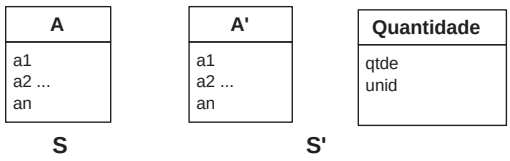

Figura 4.82: Modelagem das Quantidades

Problema: Propor uma modelagem mais eficiente para os atributos $\mathbf{a_1}, \mathbf{a_2}, ...\mathbf{a_n}$.

Solução: O esquema **S** deve ser transformado no esquema **S'** como mostrado na figura 4.82. No esquema **S'**, definimos o tipo **Quantidade** que combina valores numéricos com unidades. O novo tipo **Quantidade** consegue enriquecer a semântica capturada pelo esquema. Os atributos e ligações de **A'** são definidos como segue:

- `At(A') = At(A)`

- `Lg(A') = Lg(A)`

O mapeamento entre o esquema transformado **S'** e o esquema original **S** é formalmente especificado pelas assertivas de correspondência abaixo:

(i) `A` $\equiv$ `A'`

(ii) todos os atributos de **A** são mapeados diretamente em atributos de **A'**

Exemplo: Suponha novamente o esquema $\mathbf{S_1}$ mostrado na figura 4.81. Os atributos **altura** e **peso** estão representados como valores numéricos. Nesse caso, o mais apropriado é reestruturar o esquema $\mathbf{S_1}$ como sugerido na figura 4.83. No esquema $\mathbf{S_2}$, a quantidade é modelada como uma classe distinta denominada **Quantidade**. Nessa nova representação, estamos combinando números e unidades. Por exemplo, 60 quilos ou 160 centímetros. Além disso, a classe **Quantidade** torna-se um novo tipo que poderá ser utilizado como um tipo embutido da linguagem.

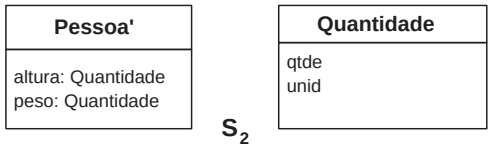

Figura 4.83: Exemplo - Modelando Quantidades Usando o Tipo Quantidade

Valores monetários também podem ser representados como quantidades, usando a moeda como unidade. Por exemplo, utilizando o esquema $\mathbf{S_3}$ da figura 4.84, o valor R$80,00 será representado como uma quantidade de 80 e unidade de Real. Nesses casos, utilizamos o termo **Dinheiro** em vez de **Quantidade**. Com essa nova representação você poderá manusear facilmente diferentes moedas.

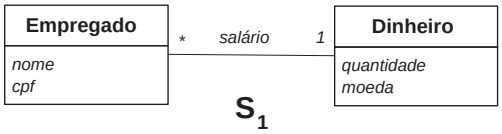

Figura 4.84: Exemplo - Modelando Quantidades usando o tipo Dinheiro

O mapeamento entre o esquema transformado $\mathbf{S_2}$ e o esquema original $\mathbf{S_1}$ é formalmente especificado pelas assertivas de correspondência abaixo:

(i) `Pessoa` $\equiv$ `Pessoa'`

(ii) todos os atributos de **Pessoa** são mapeados diretamente em atributos de **Pessoa'**

Alguns projetistas afirmam que a quantidade deve ser modelada como uma associação, pois não é um tipo embutido. Outros recomendam representá-la como um atributo, pois pode se tornar um tipo fundamental e amplamente utilizado. A forma como você representa essa situação não importa muito durante a modelagem conceitual. O importante é que você visualize tipos como **Quantidade** ou **Dinheiro** no seu esquema conceitual. Em geral, quando vários atributos possuem um comportamento em comum que está presente em várias classes, combine esses atributos em um novo tipo fundamental.

Usos Conhecidos: Uma discussão relacionada a esse padrão pode ser encontrada em [Lar97] e [Fow97a].

P26. Modelando Históricos

Contexto: Suponha o esquema **S** mostrado na figura 4.85.

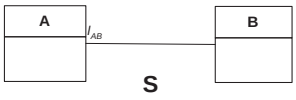

Figura 4.85: Modelando Históricos

Sejam **A** e **B** classes e l_{AB} uma ligação definida entre as classes **A** e **B**. Desejamos armazenar o histórico da ligação l_{AB}.

Problema: Representar no esquema conceitual o histórico da associação l_{AB}.

Solução: O esquema **S** deve ser transformado no esquema **S'** como mostrado na figura 4.86. No esquema **S'**, a ligação l_{AB} foi transformada na classe de associação **D** e a nova classe **Período** foi adicionada.

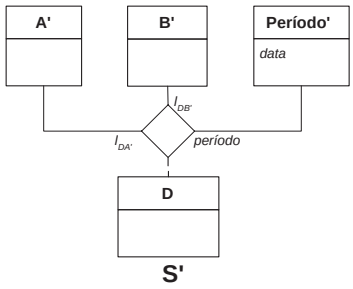

Figura 4.86: Modelando Históricos

O mapeamento entre o esquema transformado **S'** e o esquema original **S** é formalmente especificado pelas assertivas de correspondência abaixo:

(i) $\mathtt{A} \equiv \mathtt{A'}$

(ii) $\mathtt{B} \equiv \mathtt{B'}$

(iii) todos os atributos de **A** são mapeados diretamente em atributos de **A'**

(iv) todos os atributos de **B** são mapeados diretamente em atributos de **B'**

(v) $\mathtt{A}.l_{AB} \equiv \mathtt{A'}.(\mathtt{D}.l_{DA'})^{-1}.l_{DB'}$

Exemplo: Suponha que você queira armazenar o salário de um empregado. Como sugerido no padrão **P25**, o salário de um empregado pode ser modelado como mostrado no esquema $\mathbf{S_1}$ da figura 4.87. Contudo, à medida que o tempo passa, o valor desse salário pode variar. Como poderíamos responder à pergunta "*Qual era o valor do salário da empregada Maria em 2 de janeiro de 1997*", se não estamos armazenando o histórico do salário dos empregados?

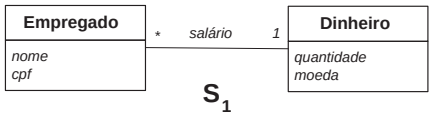

Figura 4.87: Exemplo - Modelando o Salário de um Empregado

Para solucionar o problema descrito, o esquema $\mathbf{S_1}$ deve ser reestruturado como mostrado no esquema $\mathbf{S_2}$ da figura 4.88. No esquema $\mathbf{S_2}$, definimos a classe de associação **Salário** que é constituída das classes **Empregado**, **Dinheiro** e **Período**. Essa nova representação, além de armazenar os valores dos salários dos empregados em um determinado instante, também permite armazenar o histórico dos salários desse empregado no decorrer do tempo.

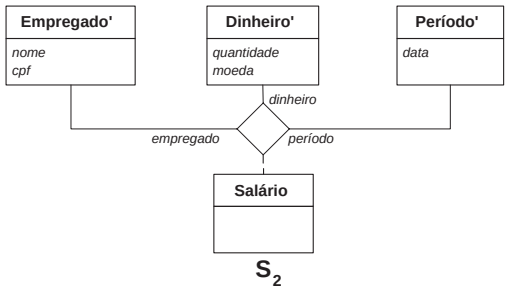

Figura 4.88: Exemplo - Modelando o Histórico das Associações

O mapeamento entre o esquema transformado $\mathbf{S_2}$ e o esquema original $\mathbf{S_1}$ é formalmente especificado pelas assertivas de correspondência abaixo:

(i) `Empregado` $\equiv$ `Empregado'`

(ii) `Dinheiro` $\equiv$ `Dinheiro'`

(iii) `Empregado.nome` $\equiv$ `Empregado'.nome`

(iv) `Empregado.cpf` $\equiv$ `Empregado'.cpf`

(v) `Dinheiro.quantidade` $\equiv$ `Dinheiro'.quantidade`

(vi) `Dinheiro.moeda` $\equiv$ `Dinheiro'.moeda`

(vii) `Empregado.`$salário$ $\equiv$ `Empregado'.(Salário.`$empregado$`)`$^{-1}$`.`$dinheiro$

Usos Conhecidos: Uma discussão relacionada a esse padrão pode ser encontrada em [Fow97a].

Padrões Relacionados: No padrão **P25** discutimos uma forma de representar valores numéricos no esquema conceitual.

Capítulo 5

Conclusões

A modelagem conceitual é considerada uma fase crucial do processo de projeto de um banco de dados, pois é durante essa fase que o projetista deve compreender, estruturar e representar a semântica do mundo real através de um modelo de dados de alto nível, de modo que esses aspectos possam ser incorporados a um sistema de informação e reflitam as necessidades dos usuários e aplicações de forma natural e direta. Erros introduzidos em um esquema conceitual podem se estender para as próximas fases do projeto do banco de dados, tornando-se cada vez mais caros e difíceis de serem removidos. Construir um bom esquema requer bastante cuidado e um entendimento detalhado do domínio da aplicação a ser modelada para que esta possa ser representada de forma clara e precisa.

Neste trabalho abordamos o problema do processo de integração de visões durante o projeto conceitual de um banco de dados. Segundo [SNL86], dois motivos justificam a necessidade de integração de visões durante o projeto de um banco de dados: (i) *a estrutura de um banco de dados para grandes aplicações é bastante complexa de ser modelada por um único projetista em uma única visão;* (ii) *em geral, os grupos de usuários trabalham de forma independente nas organizações e possuem seus próprios requisitos dos dados, que podem conflitar com os interesses de outros grupos.*

A principal contribuição deste trabalho foi o desenvolvimento de um catálogo de padrões de modelagem conceitual. Esses padrões de modelagem foram então utilizados para auxiliar os projetistas nas atividades envolvidas no processo de integrar as visões de diversos grupos de usuários com o uso do diagrama de classes da UML. A idéia associada a esse catálogo de padrões é propor uma série de soluções individuais para problemas de integração de visões que, em conjunto, mostram como construir o esquema conceitual global. Com esse objetivo, foram criados 26 padrões, agrupados em três categorias principais e classificados de acordo com o problema de integração que eles tratam.

A motivação para o desenvolvimento desse estudo foi encontrada na ineficácia existente nas atuais práticas de modelagem conceitual, quando aplicadas a situações específicas do mundo real. A UML foi utilizada por se tratar de uma linguagem de modelagem unificada, além de ter mostrado todos os sinais de se tornar a linguagem de modelagem padrão para especificar, visualizar, documentar e construir aplicações orientadas a objeto. Nosso estudo tomou como ponto de partida alguns trabalhos realizados para a modelagem conceitual dos dados envolvendo o uso do modelo de dados entidade e relacionamento (modelo ER). Observamos que, de várias formas, o diagrama de classes da UML assemelha-se ao diagrama ER. Na verdade, grande parte das técnicas propostas para a integração de visões com a UML tiveram origem na modelagem ER, apesar do diagrama de classes ser bem mais abrangente que o diagrama ER.

Uma outra importante contribuição deste trabalho foi a formalização do significado dos construtores do modelo de objetos da UML para que estes pudessem ser usados adequadamente na especificação dos padrões de modelagem propostos, além de mostrar implicações práticas desta formalização para as atividades de modelagem conceitual. Além disso, nós estendemos o modelo de objetos da UML para incluir as seguintes restrições de integridade: restrições de chave, dependência funcional, dependência existencial e assertivas de correspondência.

Como trabalhos futuros pretendemos incorporar o catálogo de padrões a um tutorial para ensino do processo de integração de visões utilizando o diagrama de classes da UML. Esse tutorial possibilitará, através de exemplos, questionamentos e textos explicativos, o aprendizado de forma didática de importantes técnicas de modelagem conceitual, auxiliando os projetistas de banco de dados no desenvolvimento de projetos conceituais consistentes, mesmo em situações não facilmente capturadas por um esquema conceitual.

Bibliografia

[A+77] Christopher Alexander et al. *A Pattern Language*. Oxford University Press, 1977.

[AFS81] S. Al-Fedaghi and P. Scheuermann. Mapping considerations in the design of schemas for the relational model. *IEEE Trans. Software Eng.*, 1981.

[BJR97] Grady Booch, Ivar Jacobson, and James Rumbaugh. The UML specification document. Technical report, Rational Software Corporation, 1997. Disponível em www.rational.com.

[BL84] C. Batini and M. Lenserini. A methodology for data schema integration in the entity relationship model. *IEEE Trans. Software Eng.*, 1984.

[Boo94] Grady Booch. *Object-Oriented Analysis and Design with Applications*. Addison-Wesley, second edition, 1994.

[Bro84] M. L. Brodie. *On Conceptual Modeling*, chapter On the Development of Data Models. Springer Verlg, 1984.

[Che76] Peter Chen. The Entity-Relationship Model - toward a unified view of data. *ACM Transactions Database System*, Março 1976.

[Cop92] J. O. Coplien. *Advanced C++ Programming Styles and Idioms*. Addison-Wesley, 1992.

[Cop98] James O. Coplien. Software design patterns: Common questions and answers, 1998. Software Production Research Department.

[CV83] M. Casanova and Vânia Maria Ponte Vidal. Towards a sound view integration methodology. In *2nd ACM SIGACT/SIGMOD Conference on Principles of Database Systems*, 1983.

[DSB97] Deb Dey, Veda Catherine Storey, and Terence M. Barron. Improving database design through the analysis of relatioships. Draft Only, Maio 1997.

[ELN87] R. ElMasri, J. Larson, and S.B. Navathe. Integration algorithms for federated databases and logical database design. Technical report, Honeywell Corporate Research Center, 1987.

[EN00] Ramez Elmasri and Shamkant B. Navathe. *Fundamentals of Database Systems.* Addison-Wesley, 3rd edition, 2000.

[Fow97a] Martin Fowler. *Analysis Patterns. Reusable Object Models.* Object Technology Series. Addison-Wesley, 1997.

[Fow97b] Martin Fowler. *UML Distilled.* Object Technology Series. Addison-Wesley, 1997.

[Fur98] José Davi Furlan. *Modelagem de Objetos Através da UML. Análise e Desenho Orientados a Objeto.* Makron Books, 1998.

[G+99] Franca Garzotto et al. Modeling-by-patterns of web aplications. In *ER* -, 1999.

[J+92] Ivar Jacobson et al. *Object-Oriented Software Engineering: A Use Case Driven Approach.* Addison-Wesley, 1992.

[Kah79] B. Kahn. *A Structured Logical Database Design Methodology.* PhD thesis, University of Michigan, 1979.

[Lar97] Craig Larman. *Applying UML and Patterns - An Introduction to Object-Oriented Analysis and Design.* Prentice Hall, 1997.

[Lin85] T. W. Ling. A normal form for entity-relationship diagrams. In *International Conference on the Entity-Relationship Approach*, 1985.

[Lós98] Bernadete Farias Lóscio. Atualização de (múltiplas) bases de dados através de mediadores. Master's thesis, Universidade Federal do Ceará, 1998.

[MIR93] R.J. Miller, Y. E. Ioannidis, and R. Ramakrishman. The use of information capacity in schema integration and translation. In *19th International Conference on Very Large Databases*, 1993.

[Mul99] Robert J. Muller. *Database Design for Smarties. Using UML for Data Modeling.* Morgan Kaufmann, 1999.

[NG82] S.B. Navathe and S.G. Gadgil. A methodology for view integration in logical database design. In *8th International Conference on Very Large Data Bases*, 1982.

[Nor01] Danielle Noronha. Projeto de banco de dados objeto-relacional a partir do diagrama de classes da uml. Master's thesis, Universidade Federal do Ceará, 2001.

[NS78] S. B. Navathe and S. G. Schkolnick. View representation in logical database design. In *ACM-SIGMOD International Conference*, 1978.

[R+92] James Rumbaugh et al. *Object-Oriented Modeling and Design*. Prentice Hall, 1992.

[Rat97] Rational Software Corporation, Santa Clara, CA. *UML Semantics*, version 1.1 edition, 1997. Disponível em www.rational.com.

[Sch87] Michael Schrefl. A comparative analysis of view integration methodologies. In *GI-Fachtagung EMISA*, 1987.

[SG88] Veda Catherine Storey and Robert C. Goldstein. A methodology for creating user views in database design. *ACM Transactions Database Systems*, 13(3), 1988.

[SNL86] R. Shamkant Navathe, Elmasri and J. Larson. Integrating user views in database design. *Data Engineering*, 1986.

[SS77] J. M. Smith and D. C. P. Smith. Database abstractions: Aggregation and generalization. In *ACM Transactions on Database Systems*, 1977.

[Sto88] Veda Catherine Storey. *View Creation - An Expert System for Database Design*. ICIT Press, 1988.

[Sto91] Veda C. Storey. Relational database design based on the entity-relationship model. *Data e Knowledge Engineering*, 1991.

[TF82] T. Teorey and J. Fry. *Design of Database Structures*. Prentice-Hall, 1982.

[Vid94] Vânia Maria Ponte Vidal. *Preservando a Semântica de Atualizações na Integração de Visões*. PhD thesis, Universidade Federal do Rio de Janeiro, 1994.

[WE79] G. Wiederhod and R. ElMasri. A structural model for database systems, 1979. Computer Science Department.

[WSW99] Yair Wand, Veda Catherine Storey, and Ron Weber. An ontological analisys of the relationship construct in conceptual modeling. Fevereiro 1999.

[Y+82] S. B. Yao et al. View modeling and integration using the functional data model. *IEEE Trans Soft Eng*, 8(6), 1982.