Mas se, em vez de 1, fosse uma multiplicidade maior como "*"? Nesse caso, a leitura seria: "para cada valor do qualificador corresponde um *conjunto* de instâncias da classe qualificada".

Por exemplo, livros podem ser classificados por gênero. Como vários livros podem pertencer ao mesmo gênero, esse atributo não constitui um identificador (*oid*). Mas é possível definir uma *partição* do conjunto dos livros a partir do gênero, como na Figura 7.25.

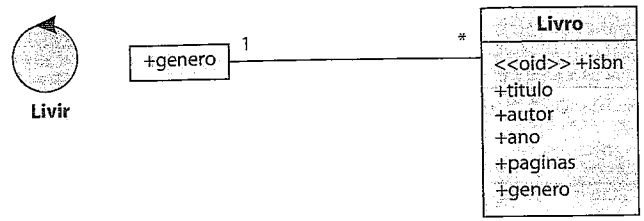

Figura 7.25: Uma partição.

A Figura 7.25 estabelece que, para cada valor possível de gênero (que poderia ser uma enumeração, por exemplo), corresponde um conjunto de livros (com zero ou mais elementos). O papel do lado esquerdo com multiplicidade 1 estabelece que todo livro participa da associação (é obrigatória) e, portanto, todo livro tem um gênero.

Essa restrição também faz admitir que o gênero poderia ser um atributo da classe Livro. Quando o qualificador é um atributo da classe, é chamado de *qualificador interno* (como nas Figuras 7.24 e 7.25); quando o qualificador não é atributo da classe qualificada, é chamado de *qualificador externo*, como na Figura 7.26.

7.4.5.7. Relação

Agora, se um livro puder ser classificado em mais de um gênero – por exemplo, o *Guia do Mochileiro das Galáxias* (Adams, 2004), que pode ser classificado no gênero "Humor", mas também no gênero "Ficção Científica" – nesse caso, bastaria trocar a multiplicidade de papel do lado esquerdo da Figura 7.25 para * ou 1..*, resultando em uma relação como na Figura 7.26.

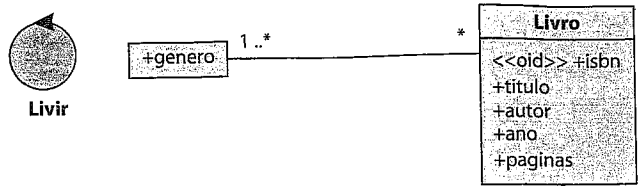

Figura 7.26: Uma relação.

A relação da Figura 7.26 diz que um livro tem um ou mais gêneros, e um gênero tem um conjunto de livros associado a ele. Nesse caso, o qualificador tem de ser, necessariamente, externo.

Possivelmente, a real necessidade das associações qualificadas nesses diagramas surgirá de forma mais enfática no capítulo seguinte, quando será necessário escrever expressões para referenciar objetos. Será mais fácil acessar um elemento indexado por uma associação qualificada do que obter todo o conjunto de instâncias referenciadas pela associação e procurar o elemento em questão ali dentro a partir de uma chave de busca.

7.4.6. Agregação e Composição

Algumas associações podem ser consideradas mais fortes do que outras no sentido de que elas definem um objeto que é composto por outros. Uma casa, por exemplo, é composta por seus cômodos.

Se existir exclusividade nessa associação, ou seja, se um item não pode ser parte de nenhum outro conceito, então a agregação é considerada forte e representada por um losango preto, como na Figura 7.27. Esse tipo de associação é também chamado de *composição*.

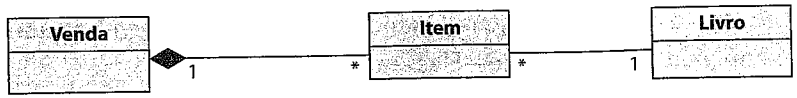

Figura 7.27: Composição.

A composição indica exclusividade na agregação. Quando essa exclusividade não é exigida, pode-se usar um losango branco, como na Figura 7.28, para indicar uma agregação compartilhada.

Por exemplo, pode-se querer saber que pessoas visualizaram os detalhes de um determinado livro e saber também quantas vezes cada pessoa visualizou esses detalhes. Pode não ser relevante saber quem visualizou primeiro, mas apenas quem visualizou mais vezes. Esse é um caso típico para utilização de um multiconjunto. Cada vez que uma pessoa visualiza um livro, uma nova associação é adicionada ao multiconjunto. A modelagem é mostrada na Figura 7.22.

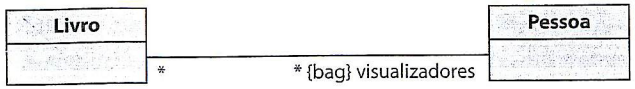

Figura 7.22: Um multiconjunto.

7.4.5.4. Lista

A lista (*sequence*) combina as propriedades de permitir a repetição de elementos e considerar a ordem entre eles. Um elemento pode aparecer mais de uma vez na lista.

Pode-se imaginar, por exemplo, que pessoas que compraram uma determinada quantidade de livros se habilitem para receber um brinde, mas apenas uma quantidade limitada de brindes é entregue por dia. Assim, uma lista de pessoas pode ser definida e os brindes vão sendo distribuídos para os primeiros da lista à medida que se tornam disponíveis. Se uma pessoa fizer duas ou mais compras na quantidade exigida para fazer jus ao brinde, pode entrar novamente na lista. Esse caso é mostrado na Figura 7.23.

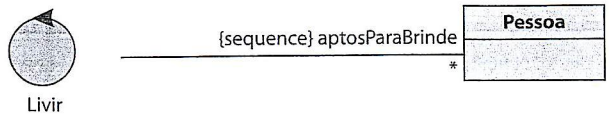

Figura 7.23: Uma lista.

A lista tem dois casos especiais importantes. Quando os primeiros elementos a serem retirados da lista são os mais antigos, como no caso da Figura 7.23, ela é uma *fila*. Nesse caso, pode-se usar a restrição {queue}.

Quando os primeiros elementos a serem retirados são os mais recentes, ela se comporta como uma *pilha*, que pode ser definida com a restrição {stack}.

Uma lista genérica pode ter inserções e retiradas em qualquer posição. Pilhas e filas só podem ter inserções e retiradas em um único local, embora qualquer de seus elementos possa ser visualizado.

7.4.5.5. Mapeamento

Quando um conceito tem um atributo identificador, pode-se criar um mapeamento do identificador para o conceito, de forma que seja muito mais fácil identificar e localizar instâncias específicas do conceito. Por exemplo, o ISBN de um livro é um identificador para o livro. Então, o cadastro de livros (associação a partir da controladora) pode ser qualificado pelo ISBN e, dessa forma, em vez de a controladora ter acesso a um mero conjunto de livros, ela terá acesso a um mapeamento que permite identificar um livro diretamente a partir de seu ISBN. A Figura 7.24 mostra a modelagem desse mapeamento.

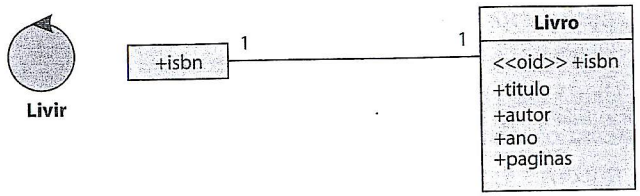

Figura 7.24: Um mapeamento.

O pequeno retângulo do lado esquerdo da associação representa um *qualificador* para o papel do lado oposto. Ou seja, a leitura que se deve fazer dessa associação é a seguinte: "para cada ISBN, há uma instância de livro". O fato de o papel do lado direito estar marcado com multiplicidade 1 significa que a cada ISBN corresponde um e exatamente um livro, ou seja, não há dois livros com o mesmo ISBN. Isso tem de ser necessariamente verdade, pois o ISBN é um identificador do livro e, portanto, não admite repetições. Na prática, continua sendo uma associação de um para muitos, mas, do lado *muitos*, cada elemento é identificado unicamente por um qualificador.

O fato de o papel do lado esquerdo também estar marcado com multiplicidade 1 significa que toda e qualquer instância de livro tem um ISBN e participa desse mapeamento, ou seja, não pode estar de fora.

7.4.5.6. Partição

No caso da Figura 7.24, a multiplicidade 1 do lado direito define um mapeamento, ou seja, para cada valor do qualificador (isbn) corresponde exatamente uma instância da classe qualificada (Livro).

Na modelagem conceitual, usualmente faz-se referência apenas a atributos e associações. Assim, se o contexto é Venda, então self.total representa o atributo total de uma instância de Venda e self.itens representa um conjunto de instâncias de Item associadas à venda pelo papel itens.

Quando a notação "." é usada sobre uma coleção, ela denota a coleção das propriedades de todos os elementos da coleção original. Assim, no contexto de Venda, a expressão "self.itens.titulo" referencia o conjunto de títulos (*strings*) dos itens de uma venda. Já a expressão self.itens.livro, que aparece na definição da associação derivada da Figura 7.19, representa o conjunto das instâncias de Livro associados às instâncias de Item associados a uma instância de Venda.

7.4.5. Coleções

Coleções de objetos não devem ser representadas como conceitos no modelo conceitual, mas como associações. De fato, uma associação com multiplicidade * representa um conjunto de objetos da classe referenciada. Assim, no exemplo da Figura 7.17, "frota" é um papel que associa um conjunto de automóveis a um dono. A modelagem da Figura 7.20 é, portanto, inadequada e desnecessária.

Figura 7.20: Uma coleção inadequadamente representada como conceito.

Assim, a forma correta de representar um conjunto ou coleção de objetos é através de um papel de associação, como na Figura 7.17 e não como um conceito (Figura 7.20).

Associações podem representar mais do que conjuntos; podem representar *tipos abstratos de dados*. Na verdade, podem também representar tipos concretos, mas estes não são importantes na atividade de análise, sendo utilizados apenas na atividade de projeto.

Para lembrar: tipos *abstratos* de dados são definidos apenas pelo seu comportamento. É o caso, por exemplo, do conjunto (*set*), onde elementos não se repetem e onde não há ordem definida; do multiconjunto (*bag*), onde

os elementos podem se repetir, mas não há ordem; da lista (*sequence*), onde elementos podem se repetir e há uma ordem entre eles; e também do conjunto ordenado (*ordered set*), fila (*queue*), pilha (*stack*) etc.

Tipos *concretos* de dados são as implementações físicas dos tipos abstratos. Por exemplo, uma lista pode ser implementada de diversas formas: como um *array*, como *lista encadeada*, como uma *árvore binária* etc. No caso dos tipos concretos, além do comportamento, há uma estrutura física que os define.

7.4.5.1. Conjuntos

Um papel de associação, na falta de mais detalhes, representa um *conjunto*, ou seja, elementos não se repetem e não há nenhuma ordem definida entre eles. Na Figura 7.17, frota é, dessa forma, um *conjunto* de automóveis de uma pessoa.

Se um mesmo automóvel for adicionado a essa associação para a mesma pessoa, o efeito é nulo, pois ele já pertence ao conjunto.

7.4.5.2. Conjunto Ordenado

Supondo que um livro que ainda não está em estoque tenha reservas de pessoas interessadas em comprá-lo assim que chegar, não se pode garantir que a quantidade de exemplares recebidos vá atender a todos os pedidos. Então, a forma mais justa de atender a esses pedidos seria atender prioritariamente as reservas mais antigas. Para isso elas devem estar ordenadas.

Por outro lado, uma mesma pessoa não pode reservar o mesmo livro mais de uma vez. Então, ainda se trata de um conjunto, sem repetições, mas os elementos se apresentam em ordem: primeiro, segundo, terceiro etc. Para definir esse tipo de estrutura de dados no modelo conceitual, basta adicionar a restrição {ordered} ao papel, como na Figura 7.21.

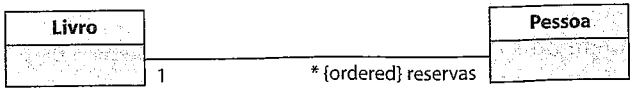

Figura 7.21: Um conjunto ordenado de reservas.

7.4.5.3. Multiconjunto

Há situações em que a ordem dos elementos não importa, mas um mesmo elemento pode aparecer mais de uma vez na coleção. Essa estrutura denomina-se *multiconjunto*, ou, mais simplesmente, em inglês, *bag*.

b) associações unidirecionais podem ser implementadas de forma mais eficiente que as bidirecionais.

Assim, essa decisão, que não afeta significativamente a modelagem conceitual, pode ser tomada com melhor conhecimento de causa (a direção das mensagens trocadas entre objetos) na atividade de projeto, como será visto no Capítulo 9.

7.4.4. Associação Derivada

Assim como algumas vezes pode ser interessante definir atributos derivados, que são calculados a partir de outros valores, pode ser também interessante ter associações derivadas, ou seja, associações que, em vez de serem representadas fisicamente, são calculadas a partir de outras informações que se tenha. Por exemplo, suponha que uma venda, em vez de se associar diretamente aos livros, se associe a um conjunto de itens, e estes, por sua vez, representem uma quantidade e um título de livro específico (Figura 7.18).

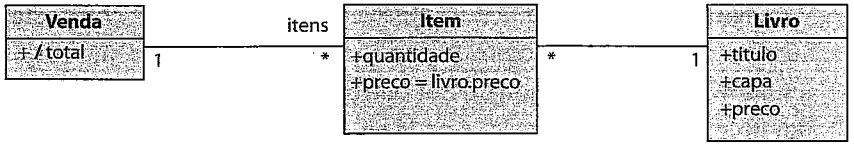

Figura 7.18: Uma venda e seus itens.

Esse tipo de modelagem é necessário quando os itens de venda não são representados individualmente, mas como quantidades de algum produto. Além disso, o livro enquanto produto pode ter seu preço atualizado, mas o item que foi vendido terá sempre o mesmo preço. Daí a necessidade de representar também o preço do item como um atributo com valor inicial igual ao preço do livro.

Porém, a partir da venda, não é mais possível acessar diretamente o conjunto de livros. Seria necessário tomar o conjunto de itens e, para cada item, verificar qual é o livro associado. Criar uma nova associação entre Venda e Livro não seria correto porque estaria representando informação que já existe no modelo (mesmo que de forma indireta). Além disso, uma nova associação entre Venda e Livro poderia associar *qualquer* venda com *qualquer* livro, não apenas aqueles que já estão presentes nos itens da venda, o que permitiria a representação de informações inconsistentes.

A solução de modelagem, nesse caso, quando for relevante ter acesso a uma associação que pode ser derivada de informações que já existem, é o uso de uma associação derivada, como representado na Figura 7.19.

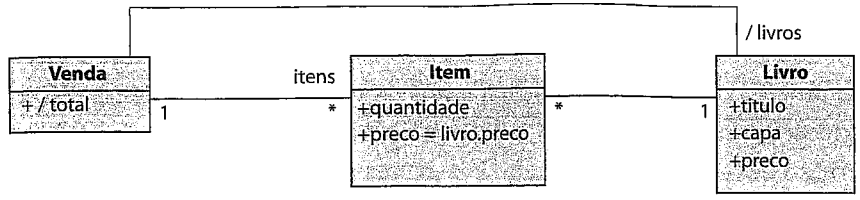

Figura 7.19: Uma associação derivada.

Uma associação derivada só tem papel e multiplicidade em uma direção (no caso, de Venda para Livro). Na outra direção ela é indefinida. Ao contrário de associações comuns que podem ser criadas e destruídas, as derivadas só podem ser consultadas (assim como os atributos derivados, elas são *read-only*). A forma de implementar uma associação derivada varia e otimizações podem ser feitas para que ela não precise ser recalculada a cada vez que for acessada.

Uma associação derivada pode ser definida em OCL. O exemplo da Figura 7.19 poderia ser escrito assim:

```
Context Venda::livros derive: self.itens.livro
```

Em relação a essa expressão OCL, pode-se observar que:

a) o contexto é a própria associação derivada a partir da classe Venda conforme definido no diagrama;
b) usa-se "derive" como no caso de atributos derivados. O que define se é um atributo ou associação derivada é o contexto e o tipo de informação, já que atributos são alfanuméricos e associações definem conjuntos de objetos;
c) "self" denota uma instância do contexto da expressão OCL. No caso, qualquer instância de Venda;
d) "." é uma notação que permite referenciar uma propriedade de um objeto.

Propriedades que podem ser referenciadas pela notação "." são:

a) atributos;
b) associações;
c) métodos.

7.4.2. Multiplicidade de Papéis

Na modelagem conceitual, é fundamental que se saiba a quantidade de elementos que uma associação permite em cada um de seus papéis. Por exemplo, na associação entre Pessoa e Automovel da Figura 7.11, quantos automóveis uma pessoa pode dirigir? Quantos motoristas um automóvel pode ter?

A resposta sempre dependerá de um estudo sobre a natureza do problema e sobre o real significado da associação, especialmente se ela representa o presente ou o histórico. Quer dizer, se a associação da Figura 7.11 representa o presente, pode-se admitir que um automóvel tenha um único motorista. Mas, se a associação representa o histórico, pode-se admitir que o automóvel tenha uma série de motoristas, que o dirigiram em momentos diferentes.

Assim, é fundamental que o analista decida claramente o que a associação significa antes de anotar a multiplicidade de seus papéis.

Há, basicamente, duas decisões a tomar sobre a multiplicidade de um papel de associação:

a) se o papel é obrigatório ou não. Por exemplo, uma pessoa é obrigada a ter pelo menos um automóvel? Um automóvel deve obrigatoriamente ter um dono?;
b) se a quantidade de instâncias que podem ser associadas através do papel tem um limite conceitual definido. Por exemplo, existe um número máximo ou mínimo de automóveis que uma pessoa pode possuir?

Deve-se tomar cuidado com algumas armadilhas conceituais. Em relação ao primeiro tópico, por exemplo, espera-se que a toda venda corresponda um pagamento. Mas isso não torna a associação obrigatória, pois a venda pode existir sem pagamento. Um dia ela possivelmente será paga, mas pode existir sem o pagamento por algum tempo. Então, esse papel *não* é obrigatório para a venda.

Outro caso refere-se ao limite máximo. Claro que o número máximo de automóveis que uma pessoa pode possuir é o número de automóveis que existe no planeta. Mas, à medida que outros automóveis venham a ser construídos, esse magnata poderá possuí-los também. Ou seja, embora exista um limite físico, não há um limite lógico para a posse. Então, o papel deve ser considerado virtualmente sem limite superior.

A multiplicidade de papel é representada por uma expressão numérica, onde:

a) "*" representa o infinito;
b) a vírgula (,) significa "e";
c) ponto-ponto (..) significa "até".

A seguir são apresentados alguns exemplos usuais e seu significado:

a) 1 exatamente um
b) 0..1 zero ou um
c) * de zero a infinito
d) 1..* de um a infinito
e) 2..5 de dois a cinco
f) 2,5 dois ou cinco
g) 2,5..8 dois ou de cinco a oito

Os valores de multiplicidade que incluem o zero são opcionais. Já os demais representam papéis obrigatórios.

A Figura 7.17 mostra que uma pessoa pode ter uma frota composta de um número qualquer de automóveis (opcional) e que pode ser motorista de um automóvel (também opcional). Por outro lado, mostra que um automóvel tem um único dono (obrigatório) e pode ter ou não um motorista (opcional).

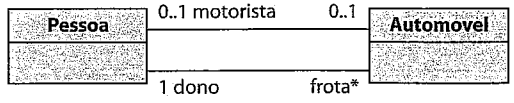

Figura 7.17: Associações com multiplicidade de papel.

7.4.3. Direção das Associações

Uma associação, no modelo conceitual, deve ser *não direcional*, isto é, a origem e o destino da associação não devem ser estabelecidos. Isso se deve ao fato de que, na atividade de análise, basta saber que dois conceitos estão associados.

Há pouca praticidade em definir prematuramente (antes da atividade de projeto) a direção de uma associação. Associações *unidirecionais* teriam como vantagem apenas o seguinte:

a) não é necessário definir o nome de papel na origem de uma associação unidirecional;

associação com automóveis: aqueles dos quais é dona, correspondendo à sua frota, e aquele do qual é motorista, correspondendo simplesmente ao seu automovel.

Voltando à discussão sobre as associações não serem operações, observa-se que uma pessoa pode comprar um automóvel. Isso é uma operação porque transforma as associações: uma associação de posse deixa de existir e outra é criada em seu lugar. Seria incorreto representar a operação como uma associação, como na Figura 7.13.

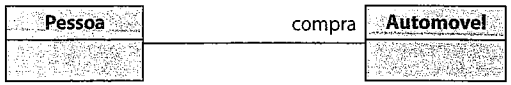

Figura 7.13: Uma operação incorretamente rotulando um papel de associação.

Segundo a definição da UML, associações também podem ter nomes, que são colocados no meio da linha que liga duas classes e não nas extremidades. Porém, tais nomes, além de serem difíceis de atribuir (analistas iniciantes preencherão seus diagramas com associações chamadas "possui" ou sinônimos), não têm qualquer resultado prático. Mais vale trabalhar com bons nomes de papel do que com nomes de associações. Então, é prático simplesmente ignorar que associações tenham nomes e viver feliz com os nomes de papel, muito mais úteis para definir possibilidades de navegação entre conceitos (Capítulos 8 e 9) e para nomear elementos de programação (Capítulo 12).

Algumas vezes, pode ser interessante guardar informações sobre operações ou transações que foram efetuadas. Assim, pode ser interessante para o sistema guardar as informações sobre a transação de compra, como data, valor pago etc. Nesse caso, a transação é tratada no modelo conceitual como um conceito complexo (Figura 7.14) e representa estaticamente a memória da operação que um dia foi efetuada.

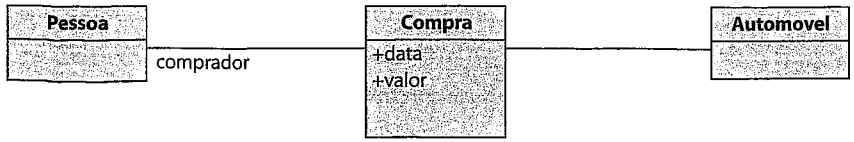

Figura 7.14: Transação representada como conceito.

Porém, como se pode notar, a transação é representada por um conceito, enquanto as associações continuam representando ligações estáticas entre conceitos e não operações ou transformações.

7.4.1. Como Encontrar Associações

Se a informação correspondente aos conceitos e atributos pode ser facilmente encontrada no texto dos casos de uso, o mesmo não ocorre com as associações.

Mas, então, como encontrar as associações entre os conceitos complexos? Há duas regras:

a) conceitos dependentes (como Compra) precisam necessariamente estar ligados aos conceitos que os complementam (como Comprador e Item);
b) informações associativas só podem ser representadas através de associações.

Informações associativas podem até aparecer nos casos de uso como conceitos. Mas quando se afirma que uma pessoa é dona ou motorista de um automóvel, essa informação só pode ser colocada na forma de uma associação. Analistas viciados em modelos de dados relacionais poderão fazer uma modelagem como a da Figura 7.15, mas está incorreta, pois atributos devem ser alfanuméricos e nunca conceitos complexos (para isso existem associações).

Figura 7.15: Um atributo representando *indevidamente* uma associação.

Também é incorreto utilizar chaves estrangeiras (Date, 1982), como na Figura 7.16. Quando dois conceitos complexos se relacionam, o elemento de modelagem é a associação, e fim de conversa!

Figura 7.16: Um atributo como chave estrangeira representando *indevidamente* uma associação.

Uma regra geral de *coesão* a ser usada é que um conceito só deve conter seus próprios atributos e nunca os atributos de outro conceito. Ora, o CPF do dono é um atributo de uma pessoa e não do automóvel. Por isso é inadequado representá-lo como na Figura 7.16. Outro motivo para que se tenham associações visíveis é prático: é muito mais fácil perceber quais objetos efetivamente têm referências para com outros. Mais adiante (Capítulo 9) será visto como essas linhas de *visibilidade* são importantes para que se possa projetar um código efetivamente orientado a objetos de qualidade.

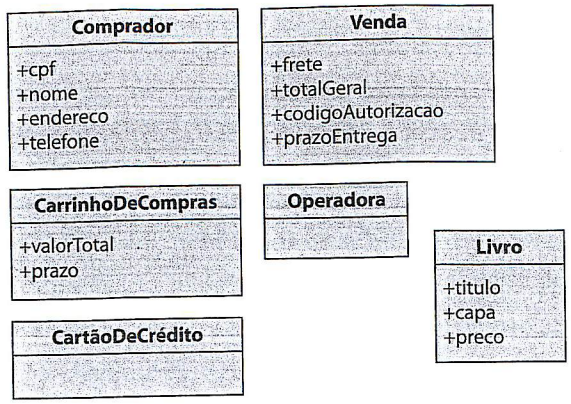

Figura 7.9: Modelo conceitual preliminar (ainda sem associações) criado a partir dos conceitos e atributos identificados no caso de uso da Figura 7.8.

Observa-se que, nos casos em que a informação passada no caso de uso foi detalhada (Comprador, Venda, Livro), os atributos foram devidamente identificados, mas, no caso em que a informação não foi devidamente detalhada no caso de uso (CartaoDeCredito Operadora), não foram identificados atributos. Isso demonstra a necessidade de fazer constar, no caso de uso, as informações que são efetivamente passadas dos atores ao sistema e vice-versa.

7.4. Associações

Quando dois ou mais conceitos complexos se relacionam entre si, diz-se que existe uma *associação* entre eles. Por exemplo, uma venda está associada aos livros que foram vendidos e também ao comprador a quem os livros foram vendidos. Um pagamento, quando existir, precisa estar obrigatoriamente associado a uma venda. Saindo um pouco do exemplo do sistema Livir, uma pessoa pode estar associada a um automóvel que seja de sua propriedade.

Os textos dos casos de uso mencionam associações com pouca frequência. Como os casos de uso descrevem ações de interação entre usuários e o sistema, então a sua descrição acaba mencionando principalmente as *operações*. Cabe aqui, portanto, definir claramente a diferença:

a) *associação* é uma relação estática que pode existir entre dois conceitos complexos, complementando a informação que se tem sobre eles em um determinado instante ou referenciando informação associativa nova;

b) *operação* é o ato de transformar a informação, fazendo-a passar de um estado para outro, mudando, por exemplo, a configuração das associações, destruindo e/ou criando novas associações ou objetos, ou modificando o valor dos atributos.

Assim, o texto dos casos de uso está frequentemente repleto de operações, mas não de associações. Quanto se têm, por exemplo, as classes Pessoa e Automovel, como na Figura 7.10, a associação estática que existe entre elas pode indicar a posse de uma pela outra.

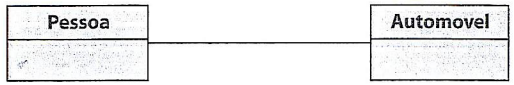

Figura 7.10: Representação de uma associação estática entre dois conceitos.

Porém, existem diferentes formas de associação entre pessoas e automóveis que não meramente a posse. Uma pessoa pode ser *passageira* de um automóvel ou *motorista* dele, ou ainda a associação pode simplesmente representar que uma determinada pessoa *gosta* de um determinado automóvel. Para eliminar tais ambiguidades, é conveniente, em muitos casos, utilizar um *nome de papel* para as associações, o qual pode ser colocado em um ou ambos os lados e deve ser lido como se fosse uma função. Na Figura 7.11, por exemplo, uma pessoa está no papel ou função de motorista de um automóvel.

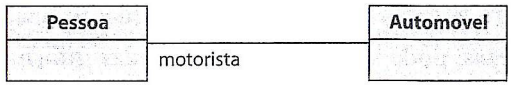

Figura 7.11: Uma associação com nome de papel.

Diferentes papéis podem ser representados através de associações diferentes, como na Figura 7.12.

Figura 7.12: Múltiplas associações entre conceitos.

Na falta de um nome de papel explícito, o próprio nome da classe associada deve ser considerado como nome de papel. No caso da Figura 7.12, um automóvel explicitamente tem dois tipos de associação com pessoas: dono e motorista. Do ponto de vista inverso, porém, uma pessoa tem dois tipos de

dem indicar conceitos, pois o verbo pode exprimir um ato que corresponde a um substantivo, como por exemplo, "pagar", que corresponde ao substantivo "pagamento"; "comprar", que corresponde ao substantivo "compra" etc.

Via de regra, entretanto, o analista deve ter em mente os objetivos do sistema enquanto procura descobrir os elementos do modelo conceitual. Não é interessante representar, no modelo conceitual, a informação que é irrelevante para o sistema. Assim, nem todos os substantivos e verbos deverão ser considerados no modelo. O analista tem a responsabilidade de compreender quais as verdadeiras necessidades de informação e filtrar as irrelevâncias.

O processo de identificação dos conceitos e atributos, então, consiste em:

a) identificar no texto dos casos de uso as palavras ou sintagmas que correspondem a conceitos sobre os quais se tem interesse em manter informação no sistema;
b) agrupar as palavras ou expressões que são sinônimos, como, por exemplo, "compra" e "aquisição", "comprador" e "cliente" etc.;
c) identificar quais dos itens considerados correspondem a conceitos complexos e quais são meros atributos. Os atributos são aqueles elementos que podem ser considerados alfanuméricos, usualmente: nomes, números em geral, códigos, datas, valores em moeda, valores booleanos (verdadeiro ou falso) etc.

Aplicando essa técnica ao caso de uso da Figura 5.10, são encontrados os principais elementos de informação a serem trabalhados. Na Figura 7.8, os elementos identificados como conceitos ou atributos são grifados no texto.

Caso de Uso: Comprar livros

1. [IN] O comprador informa sua identificação.
2. [OUT] O sistema informa os livros disponíveis para venda (título, capa e preço) e o conteúdo atual do carrinho de compras.
3. [IN] O comprador seleciona os livros que deseja comprar.
4. O comprador decide se finaliza a compra ou se guarda o carrinho:
   - 4.1 Variante: Finalizar a compra.
   - 4.2 Variante: Guardar carrinho.

Variante 4.1: Finalizar a compra

4.1.1. [OUT] O sistema informa o valor total dos livros e apresenta as opções de endereço cadastradas.
4.1.2. [IN] O comprador seleciona um endereço para entrega.
4.1.3. [OUT] O sistema informa o valor do frete e total geral, bem como a lista de cartões de crédito já cadastrados para pagamento.
4.1.4. [IN] O comprador seleciona um cartão de crédito.
4.1.5. [OUT] O sistema envia os dados do cartão e valor da venda para a operadora.
4.1.6. [IN] A operadora informa o código de autorização.
4.1.7. [OUT] O sistema informa o prazo de entrega.

Variante 4.2: Guardar carrinho

4.2.1 [OUT] O sistema informa o prazo (dias) em que o carrinho será mantido.

Exceção 1a: Comprador não cadastrado

1a.1 [IN] O comprador informa seu CPF, nome, endereço e telefone.

Retorna ao passo 1.

Exceção 4.1.2a: Endereço consta como inválido

4.1.2a.1 [IN] O comprador atualiza o endereço.

Avança para o passo 4.1.2.

Exceção 4.1.6a: A operadora não autoriza a venda

4.1.6a.1 [OUT] O sistema apresenta outras opções de cartão ao comprador.
4.1.6a.2 [IN] O comprador seleciona outro cartão.

Retorna ao passo 4.1.5.

Figura 7.8: Um caso de uso com possíveis conceitos e atributos grifados.

O resultado dessa análise pode ser imediatamente transportado para uma diagrama de classes UML (Figura 7.9).

pode ter vários identificadores. Assim, o conceito de identificador assemelha-se mais ao conceito de *chave candidata* de banco de dados (Date, 1982).

7.2.2. Classe Controladora de Sistema

Usualmente, espera-se que um modelo conceitual corresponda a um grafo conexo, ou seja, que todos os conceitos se conectem de alguma forma. A não existência dessa situação pode indicar ao analista que algumas associações importantes ainda não foram descobertas.

Será muito útil, desde a fase de modelagem conceitual, adicionar ao diagrama uma classe que representa o sistema como um todo. Essa classe corresponderá à *controladora de sistema* ou *controladora-fachada* (*façade controller*), já mencionada no Capítulo 6.

Sendo uma classe de controle, admite-se que a controladora de sistema tenha apenas uma única instância (o sistema); ela pode ser uma classe estereotipada com <<control>> ou ainda desenhada de acordo com o padrão de Jacobson, como Livir na Figura 7.23.

7.2.3. Conceitos Dependentes e Independentes

Uma classificação interessante e útil para conceitos é verificar se são dependentes ou independentes.

Um conceito é *dependente* se precisa estar associado a outros conceitos para fazer sentido, ou seja, para representar uma informação minimamente compreensível. Não faria sentido ter um conceito de venda que não estivesse associado a um comprador e um conjunto de livros.

Um conceito é *independente* se pode ser compreendido sem estar associado a outros. Por exemplo, o conceito Pessoa pode ser compreendido a partir de seus atributos, sem necessidade de estar associado a outros conceitos. Uma pessoa não precisa ter comprado nada para ser uma pessoa. Essas associações com compras são opcionais para ela. Então, o conceito Pessoa é, nesse sentido, independente.

Um paralelo pode ser encontrado na linguística, onde os *verbos transitivos* precisam de complemento (correspondendo aos conceitos dependentes) e os *intransitivos* não precisam de complemento (correspondendo aos conceitos independentes). Assim, pode-se dizer "João dormiu" sem acrescentar

nenhum complemento ao verbo, e a frase faz sentido. Mas não se pode dizer "João fez" sem que haja um complemento, mesmo que por elipse. Para a frase ter sentido, João tem de ter feito *alguma coisa*.

Mas, para que serve essa distinção? É interessante notar que apenas os conceitos independentes se prestam a ser cadastros, ou seja, eles são operáveis através de um caso de uso *CRUD*. Pode-se ter, então, cadastros de pessoas, de livros, de fornecedores etc., mas, usualmente, os conceitos dependentes não se prestam a esse tipo de operação. Compras, vales-presente, pagamentos não podem simplesmente ser *cadastrados* na forma *CRUD*, mas são operados nos casos de uso que correspondem a processos de negócio, porque possivelmente comportam interações e exceções que fogem ao padrão.

Mais adiante, será visto que o acesso à informação no sistema (modelo dinâmico) inicia sempre na controladora-fachada. Ou seja, qualquer caminho de acesso à informação parte da controladora-fachada. Assim, conceitos diretamente ligados à controladora-fachada são aqueles cujas instâncias podem ser acessadas diretamente. Livros e compradores são cadastros (conceitos independentes) e, portanto, podem ser acessados diretamente.

Como consequência disso, apenas os conceitos independentes estarão inicialmente associados à classe controladora de sistema. Os conceitos dependentes não terão esse tipo de associação a não ser que ela represente algum tipo de informação adicional. Por exemplo, uma associação entre reservas (conceito dependente) poderia estar ligada à controladora-fachada através de uma associação ordenada para indicar a ordem de prioridade entre as reservas, conforme será visto mais adiante.

7.3. Como Encontrar Conceitos e Atributos

O processo de descoberta dos elementos do modelo conceitual pode variar. Porém, uma forma bastante útil é olhar para o texto dos casos de uso expandidos ou os diagramas de sequência de sistema. A partir desses artefatos, pode-se descobrir todos os elementos textuais que eventualmente referenciam informação a ser guardada e/ou processada.

Usualmente, esses elementos textuais são compostos por substantivos, como "pessoa", "compra", "pagamento" etc., ou por expressões que denotam substantivos (conhecidas em linguística como *sintagmas nominais*), como "autorização de venda". Além disso, no texto, algumas vezes alguns verbos po-

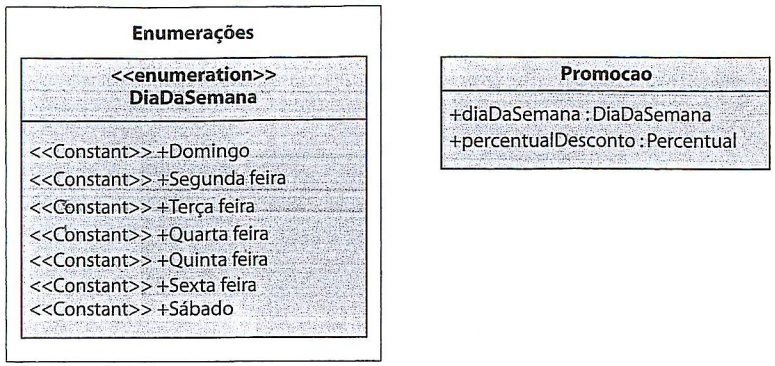

Figura 7.6: Um pacote com uma enumeração e seu uso em uma classe.

Na Figura 7.6, o atributo diaDaSemana, na classe Promocao pode assumir um dentre os sete valores da enumeração DiaDaSemana.

Em hipótese alguma enumerações podem ter associações com outros elementos ou atributos (os valores que aparecem dentro da declaração da enumeração são constantes e não atributos). Se isso acontecer, não se trata mais de uma enumeração, mas de um conceito complexo. Nesse caso, não se deve usar o estereótipo <<enumeration>> nem se pode usar o nome da enumeração como se fosse um tipo, como na Figura 7.6. Quando se trata de um conceito complexo, sua relação com outros conceitos tem de ser feita através de associações, como será mostrado mais adiante.

7.1.5. Tipos Primitivos

O analista pode e deve definir *tipos primitivos* sempre que se deparar com atributos que tenham regras de formação, como no caso do CPF. Tipos primitivos podem ser classes estereotipadas com <<primitive>>, como na Figura 7.50.

Alguns tipos primitivos como Date, Money etc. já são definidos em OCL e na maioria das linguagens de programação. Mas outros tipos podem ser criados e suas regras de formação podem ser definidas pelo analista. É o caso de CPF, CEP, NumeroPrimo e quaisquer outros tipos alfanuméricos cuja regra de formação possa ser analisada sintaticamente, ou seja, avaliando expressões em que não constem dados gerenciados pelo sistema. Por exemplo, CPF pode ser um tipo primitivo, mas CPFDeComprador não, pois para saber se o CPF

pertence a um comprador seria necessário avaliar os dados de compradores gerenciados pelo sistema.

7.2. Conceitos

É impossível falar de atributos sem falar nos conceitos, já que são fortemente ligados. Conceitos são usualmente agrupamentos coesos de atributos sobre uma mesma entidade de informação.

Conceitos são mais do que valores alfanuméricos. São também mais do que meramente um amontoado de atributos, pois eles trazem consigo um significado e podem estar associados uns com os outros.

7.2.1. Identificadores

Um *identificador* é um atributo que permite que uma instância de um conceito seja diferenciada de outras. Uma vez que um atributo tenha sido estereotipado como identificador (<<oid>> do inglês *Object IDentifier*), não é possível que existam duas instâncias do mesmo conceito com o mesmo valor para esse atributo. Um exemplo de identificador é o número de CPF para os brasileiros (Figura 7.7) porque não existem duas pessoas com o mesmo CPF (pelo menos não oficialmente).

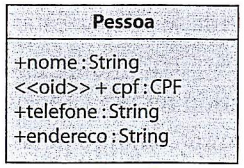

Figura 7.7: Um conceito com identificador.

Alguns dialetos usam o estereótipo <<PK>> para identificadores, derivado do inglês *Primary Key* (chave primária), um conceito de banco de dados que representa um atributo ou campo que não pode ser repetido em duas entidades.

Entretanto, a semelhança entre OID e PK não é perfeita porque um registro em um banco de dados relacional pode ter apenas uma chave primária (algumas vezes composta por mais de um atributo), enquanto um conceito

Uma venda, por exemplo, pode ser criada com um valor total que inicialmente (antes que haja algum livro na venda) será zero. Isso pode ser definido no próprio diagrama de classes do modelo conceitual, como mostrado na Figura 7.4.

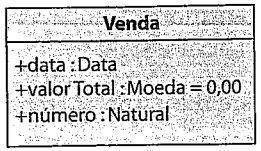

Figura 7.4: Um atributo com valor inicial.

Pode-se usar a linguagem OCL também para definir o valor inicial de um atributo de forma textual. Para isso, é necessário primeiro declarar o contexto da expressão (no caso, o atributo valorTotal, na classe Venda, representado por Venda::valorTotal) e usando a expressão init para indicar que se trata de um valor inicial de atributo. A expressão OCL seria escrita assim:

```
Context Venda::valorTotal:Moeda init: 0,00
```

Pode-se omitir o tipo Moeda, pois a OCL não obriga a declaração de tipos nas suas expressões:

```
Context Venda::valorTotal init: 0,00
```

É possível também que um atributo tenha um valor inicial calculado de forma mais complexa. Mais adiante serão apresentados exemplos de expressões complexas com OCL que podem ser usadas para inicializar atributos com valores como somatórios, quantidade de elementos associados, maior elemento associado etc.

7.1.3. Atributos Derivados

Atributos derivados são valores alfanuméricos (novamente não se admitem objetos nem estruturas de dados como conjuntos e listas) que não são definidos senão através de um cálculo. Ao contrário dos valores iniciais, que são atribuídos na criação do objeto e depois podem ser mudados à vontade, os atributos derivados não admitem qualquer mudança diretamente neles. Em outras palavras, são atributos *read-only*.

Um atributo derivado deve ser definido por uma expressão. No diagrama, representa-se o atributo derivado com uma barra (/) antes do nome do

atributo seguida da expressão que o define. Na Figura 7.5, define-se que o lucro bruto de um produto é a diferença entre seu preço de venda e seu preço de compra.

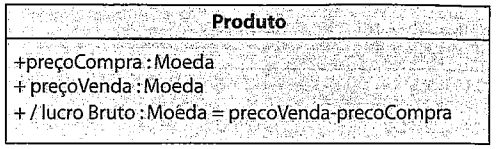

Figura 7.5: Um atributo derivado.

Em OCL, o mesmo atributo derivado poderia ser definido usando a expressão derive:

```
Context Produto::lucroBruto:Moeda
 derive:
   precoVenda - precoCompra
```

Nessa classe, apenas os atributos precoCompra e precoVenda podem ser diretamente alterados por um *setter*. O atributo lucroBruto pode apenas ser consultado. Ele é o resultado do cálculo conforme definido.

Mecanismos de otimização da fase de implementação podem definir se atributos derivados como lucroBruto serão recalculados a cada vez que forem acessados ou se serão mantidos em algum armazenamento oculto e recalculados apenas quando um de seus componentes for mudado. Por exemplo, o lucroBruto poderia ser recalculado sempre que precoCompra ou precoVenda executarem a operação *set* que altera seus valores.

7.1.4. Enumerações

Enumerações são um meio-termo entre o conceito e o atributo. Elas são basicamente *strings* e se comportam como tal, mas há um conjunto predefinido de *strings* válidas que constitui a enumeração. Por exemplo, o dia da semana só pode assumir um valor dentre sete possíveis: "domingo", "segunda-feira", "terça-feira" etc. Assim, um atributo cujo valor pode ser um dia da semana poderá ser tipado com uma enumeração.

A enumeração pode aparecer nos diagramas UML como uma classe estereotipada. Sugere-se que não sejam colocadas no modelo conceitual, mas em um pacote específico para conter enumerações, como mostrado na Figura 7.6.

c) *associações*, que consistem em um tipo de informação que liga diferentes conceitos entre si. Porém, a associação é mais do que uma mera ligação: ela própria *é* um tipo de informação. No sistema Livir, devem existir associações entre uma venda e seus itens e entre a venda e seu comprador.

Nas próximas seções, esses elementos serão detalhados. É praticamente impossível falar de um sem mencionar os outros, pois os três se inter-relacionam fortemente.

7.1. Atributos

Atributos são, no modelo conceitual, os tipos escalares, textos e outros formatos derivados populares, como data, moeda, intervalo etc.

Não devem ser consideradas como atributos as estruturas de dados com múltiplos valores como listas, conjuntos, árvores etc. Essas estruturas devem ser modeladas como associações, conforme será visto mais adiante.

Conceitos complexos (classes) também não devem ser modelados como atributos. Por exemplo, um livro não pode ser atributo de uma venda. Pode existir, se for o caso, uma *associação* entre um livro e uma venda, pois ambos são conceitos complexos.

Atributos são sempre representados no seio de uma classe, conforme a Figura 7.2, em que a classe Comprador tem os atributos nome, cpf, endereco e telefone.

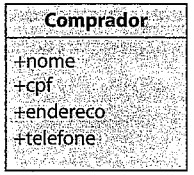

Figura 7.2: Atributos representados dentro de uma classe.

7.1.1. Tipagem

Atributos podem ser tipados, embora o modelo conceitual não exija isso explicitamente. A Figura 7.3 mostra uma versão da mesma classe da Figura 7.2 com atributos tipados.

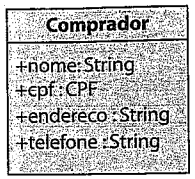

Figura 7.3: Atributos tipados.

O significado dos tipos é o mesmo correntemente atribuído pelas linguagens de programação. Observa-se, na Figura 7.3, a existência de um tipo clássico (String) e um tipo primitivo definido (CPF). Quando um atributo é definido por regras de formação, como é o caso do CPF, convém que se defina um tipo primitivo especialmente para ele, como foi feito na Figura 7.3.

O atributo telefone também poderia ser definido por um tipo especial, já que admite uma formatação específica.

Por outro lado, seria possível argumentar que um telefone é um número. Isso é verdade. Mas dificilmente alguém faria operações matemáticas com números de telefones, a não ser nos cálculos de improbabilidade da "Coração de Ouro" (Adams, 2005), mas isso já é ficção. No mundo real, embora um telefone seja composto apenas por números, ele se comporta mais como uma *string*. É mais comum extrair uma *substring* (código DDD, por exemplo) do que somar um telefone com outro.

Já o endereço é um caso à parte. Trata-se de um atributo ou um conceito complexo? Afinal, um endereço é composto por logradouro, CEP, cidade etc. Esse caso, como muitos outros, define-se pela necessidade de informação do sistema. Se endereços são usados apenas para gerar etiquetas de envelopes, então eles se comportam como atributos e podem ser representados por uma simples *string*. Porém, se endereços são usados para calcular distâncias entre diferentes pontos ou para agrupar compradores por localidade ou proximidade, então eles se comportam como conceitos complexos e devem ser modelados como uma classe com atributos e associações próprias.

7.1.2. Valores Iniciais

Um atributo pode ser declarado com um valor inicial, ou seja, sempre que uma instância do conceito (classe) for criada, aquele atributo receberá o valor inicial definido, que posteriormente poderá ser mudado, se for o caso.

delo dinâmico da atividade de projeto vai ter de mostrar claramente como as transformações ocorrem através das colaborações entre objetos.

O modelo conceitual deve descrever a informação que o sistema vai gerenciar. Trata-se de um artefato do domínio do problema e não do domínio da solução. Portanto, o modelo conceitual não deve ser confundido com a arquitetura do software (representada no DCP, ou diagrama de classes de projeto do Capítulo 9) porque esta, embora inicialmente derivada do modelo conceitual, pertence ao domínio da solução e, portanto, serve a um objetivo diferente.

O modelo conceitual também não deve ser confundido com o modelo de dados (Capítulo 11), pois o modelo de dados enfatiza a representação e a organização dos dados armazenados, enquanto o modelo conceitual visa a representar a compreensão da informação e não a sua representação física. Assim, um modelo de dados relacional é apenas uma possível representação física de um modelo conceitual mais essencial.

O analista deve lembrar que a atividade de análise de domínio considera apenas o mundo exterior ao sistema. Por isso, não faz sentido falar em modelo de dados nessa fase porque os dados estarão representados no interior do sistema. Uma maneira interessante de compreender o modelo conceitual é imaginar que os elementos descritos nele correspondem a informações que inicialmente existem apenas na mente do usuário, como na Figura 7.1, e não em um sistema físico de armazenamento.

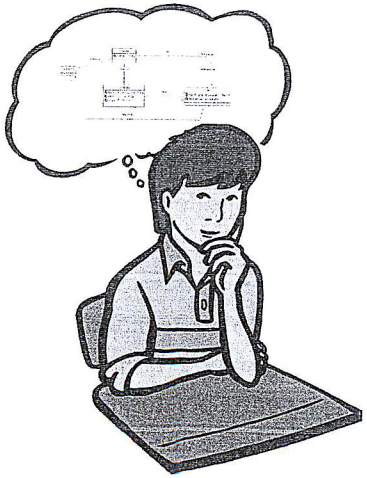

Figura 7.1: O modelo conceitual é uma representação da visão que o usuário tem das informações gerenciadas pelo sistema.

O usuário, através das operações e consultas de sistema (Capítulo 6), passa informações ao sistema e recupera informações do sistema. O sistema nem sequer precisa ser considerado como um sistema *computacional* nesse momento. Ou seja, essas informações existem independentemente de um computador para armazená-las.

O objetivo da análise é estudar o problema. Mas o sistema computacional seria uma solução para o problema e, portanto, objeto da atividade de projeto. O sistema-solução poderia também não ser computacional. Seria possível analisar todo um sistema e propor uma solução manual para implementá-lo, na qual os dados são armazenados em fichas de papel e as operações são efetuadas por funcionários da empresa usando lápis, borracha e grampeador.

Assim como os casos de uso essenciais, o modelo conceitual deve ser independente da solução física que virá a ser adotada e deve conter apenas elementos referentes ao domínio do problema em questão, ficando relegados à atividade de projeto os elementos da solução, ou seja, todos os conceitos que se referem à tecnologia, como interfaces, formas de armazenamento (banco de dados), segurança de acesso, comunicação etc.

O modelo conceitual representa somente o aspecto estático da informação. Portanto, não podem existir no modelo conceitual referências a operações ou aspectos dinâmicos dos sistemas. Então, embora o modelo conceitual seja representado pelo diagrama de classes da UML, o analista não deve ainda adicionar métodos a essas classes (o Capítulo 9 vai mostrar como fazer isso de forma adequada na atividade de projeto).

Quando se trabalha modelagem conceitual com o diagrama de classes da UML, existem precisamente três tipos de elementos para modelar a informação:

a) *atributos*, que são informações alfanuméricas simples, como números, textos, datas etc. Exemplos de atributos no sistema Livir são: nome do comprador, data do pagamento, título do livro e valor da venda. Observa-se que um atributo sempre está ligado a um elemento mais complexo: o conceito;

b) *conceitos*, que são a representação da informação complexa que agrega atributos e que não pode ser descrita meramente por tipos alfanuméricos. Exemplos de conceitos no sistema Livir são: livro, comprador, venda e pagamento;

Mas DTOs também podem ser implementados através de classes simples, sem associações, com apenas atributos e seus respectivos *geters* e *setters*. Neste caso, é necessário fazer a cópia dos atributos do objeto original para o DTO. DTOs ainda podem ser representados por estruturas de dados como o registro (RECORD) de Pascal ou estruturas XML (*eXtensible Markup Language*).

Capítulo

7

Modelagem Conceitual

A *análise de domínio* está relacionada à descoberta das informações que são gerenciadas no sistema, ou seja, à representação e transformação da informação. Ela ocorre em pelo menos duas fases do processo unificado. Na fase de concepção, pode-se fazer um modelo conceitual preliminar. Na fase de elaboração, esse modelo é refinado e complementado.

No sistema de informações de uma livraria virtual, as informações descobertas na análise de domínio possivelmente seriam relativas aos compradores, livros, editoras, autores, pagamentos, vendas etc.

As informações têm dois aspectos analisados: o *estático* (também denominado *estrutural*, que será estudado neste capítulo) e o *funcional* (estudado no Capítulo 8). O aspecto estático pode ser representado no *modelo conceitual*, e o aspecto funcional pode ser representado através dos *contratos de operações e consultas de sistema*.

Na atividade de análise não existe *modelo dinâmico*, visto que a análise considera apenas a descrição da visão *externa* do sistema. O modelo dinâmico, consistindo nas colaborações entre objetos, é reservado à atividade de projeto (Capítulo 9), pois apenas nessa fase é que se vai tratar dos aspectos internos do sistema. Assim, o modelo funcional da análise apenas especifica o que entra e o que sai do sistema, sem indicar como as transformações ocorrem. Já o mo-