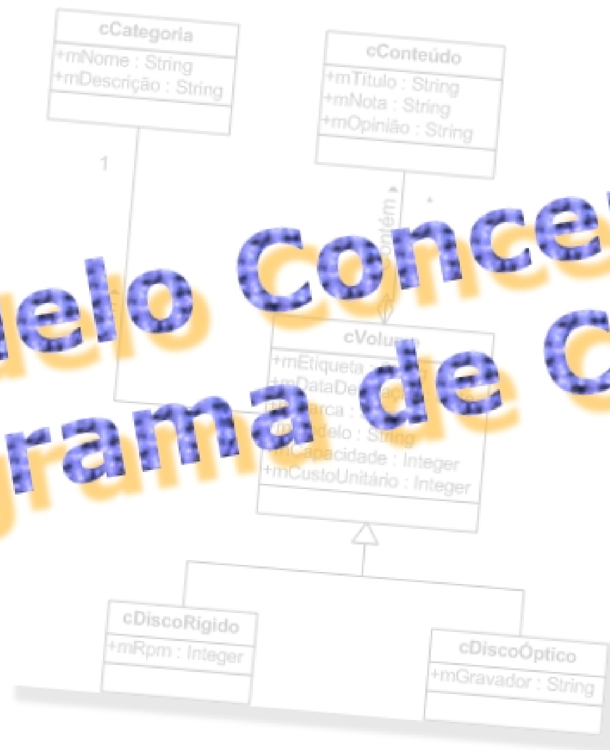
Modelo Conceptual Diagrama de Classes



© Artur Marques, 2005

Índice

Introdução 3

Modelação Orientada ao Objecto 3

Classes 4

Relações de Associação 5

Relações – Agregações 7

Relações – Generalizações 8

Relações de Dependência 8

Bibliografia 9

© Artur Marques, 2005

Introdução

Uma base de dados deve representar, em algum Sistema de Gestão de Bases de Dados (SGBD), informação do «mundo real». Para representar o «mundo real», há que observá-lo e modelá-lo de alguma maneira. Dois **modelos semânticos** muito utilizados são o Entidade/Relacionamento (E/R) e a orientação a objectos (OO), aos quais correspondem técnicas para a representação de **modelos conceptuais**. Por exemplo, em E/R tem-se os diagramas E/R [Date, 1990], e em OO tem-se a Unified Modeling Language (UML).

Este artigo pretende ser uma introdução à abordagem OO e à utilização de UML enquanto ferramenta para a produção de diagramas de classes de objectos.

Modelação Orientada ao Objecto

A orientação a objectos surge na área de linguagens de programação e é depois adoptada para actividades de análise e design/desenho [Rumbaugh et al., 1991].

Antes da orientação a objectos, as linguagens de programação convidavam claramente a pensar nos problemas em dois momentos separados: o momento da identificação e construção de estruturas de dados e o momento dos algoritmos, implementados em procedimentos e/ou funções. Esta abordagem, como tudo, tem méritos e problemas; um problema é que no «mundo real» forma e função *não são* separáveis: por exemplo, o ser humano exibe as funções que exibe (como o caminhar bípede), apenas porque a sua estrutura (como o seu esqueleto) o permite; mas também é verdade que a própria estrutura resulta da função desejada (no caso humano, depois de uma longa evolução).

A abordagem OO diz-se assim «**mais natural**».

A mudança fundamental foi a criação de uma nova construção para tipos de dados: a **classe**. A classe permite unificar estrutura e função; isto é, construir uma classe é modelar uma entidade em termos de atributos e de comportamentos, sem separá-los.
Variáveis de uma classe (manifestações concretas da classe) dizem-se **objectos** ou **instâncias** da classe. O conjunto dos valores dos atributos de um objecto, em certo momento, dizem-se o **estado** do objecto.

Uma abordagem mais natural facilita a comunicação de conceitos mais abstractos (por exemplo o conceito de herança), pelo que a aplicação da teoria dos objectos à modelação conceptual enriquece a sua expressividade.

Algumas **vantagens** da modelação OO [Coad e Yourdon, 1991]:

- Permite compreender melhor os domínios do problema.
- Contribui para a consistência interna dos resultados da análise, pela unificação de atributos e serviços.
- Permite melhor interacção entre analista(s) e especialista(s) do domínio do problema.
- O mecanismo de herança permite representar explicitamente atributos e serviços comuns entre objectos diferentes.
- Facilita a reutilização dos resultados da análise.
- Facilita a integração das fases de análise (o que deverá ser construído) e de desenho (como construir?).

© Artur Marques, 2005

Tipicamente, identificam-se como **pilares** da abordagem OO:

- A **Abstracção**, pois facilita a representação concisa de objectos complexos. O nível de abstracção diz-se apropriado quando se captura tudo o que é relevante para o domínio do problema e se ignoram os outros detalhes.
- O **Encapsulamento**, no sentido em que utilizadores (clientes) de um objecto (servidor) só precisam de conhecer a **interface** dos serviços a que acedem e não a sua implementação, de forma a que se a classe mudar essa implementação, nenhuma modificação será necessária nos objectos clientes, que poderão assim comunicar exactamente com as mesmas **mensagens**. O encapsulamento também está relacionado com questões de acesso: há atributos e serviços que estão acessíveis a certas classes (e suas instâncias), mas não a outras.
- A **Herança**, que está relacionada com situações de generalização e de especialização. A ideia é que classes que exibam os mesmos atributos e serviços, com eventuais pequenas diferenças e/ou pequenos acrescentos, possam derivar de uma classe geral comum e então especializar as suas diferenças.

A abordagem OO pode ser utilizada para descrever o **modelo de classes** de um sistema. Esta estrutura estática pode ser escrita em muitas notações, sendo aqui feita uma introdução à notação UML.
UML é um standard, gerido pelo Object Management Group (OMG), para a visualização, especificação, construção e documentação de sistemas de software. Numa primeira aproximação, UML parece um produto de software, que tem evoluído ao longo de diversas versões (1.0, 1.1, 1.2, 1.3, 1.4, 2.0…) mas, tal como uma linguagem, este standard tem vários dialectos, próprios para domínios distintos, como modelação de negócios, modelação de dados e desenvolvimento em tempo real.

Classes

Uma classe descreve as propriedades e os comportamentos de certo tipo de objectos. A partir da classe, o sistema criará objectos concretos (ditos instâncias da classe), com tais propriedades e tais comportamentos.

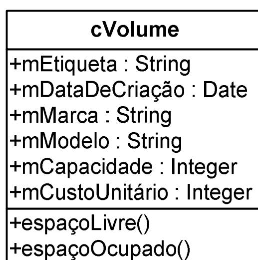


Figura 1

O exemplo da Figura 1 ilustra uma classe com nome **cVolume**, e diversos atributos (**mEtiqueta**, **mDataDeCriação**, … , **mCustoUnitário**), todos com tipo de dados assinalado (String, Date, … , Integer).
A classe presta dois serviços/comportamentos/operações: **espaçoLivre** e **espaçoOcupado**.

O grafismo exacto (como cantos arredondados ou em ângulo recto…) e a quantidade de detalhes são variáveis. Por exemplo, poderiam não estar indicados os tipos de dados dos atributos.
Nesta representação exemplificativa os atributos e as operações também têm indicação do seu nível de acesso, sendo todos prefixados por «+», por serem todos «públicos», o que significa que estão todos permanentemente acessíveis a qualquer classe.
O nível de acesso «privado» tem o prefixo «-» e corresponde às situações em que o acesso só é permitido dentro da própria classe.
O prefixo «#» usa-se quando os membros são «protegidos»; isto é, acessíveis apenas na sua própria classe e nas classes herdeiras.

O nível de detalhe a usar na representação depende dos objectivos do modelo de classes, o que está em conformidade com a flexibilidade da UML.

© Artur Marques, 2005

Neste exemplo também se procurou seguir uma nomenclatura; isto é, um estilo consistente para os nomes dos identificadores, como começar os membros atributos por «m» e seguir a «notação do camelo» (começar a escrever em minúsculas e depois capitalizar a primeira letra de cada nova palavra que componha o identificador). Seguir uma nomenclatura não é mandatório, mas ajuda a criar uma disciplina que depois facilita a leitura.

O essencial na representação de uma classe é
- utilizar identificadores claros, no singular;
- identificar as três regiões: nome da classe, zona de atributos e zona de operações.

Relações de Associação

Uma associação representa uma relação estrutural entre classes, ou seja diz que há um relacionamento entre objectos das classes.
Normalmente, as associações são bidireccionais, o que significa que ambos os objectos «se conhecem». Pode tornar-se explícita a bidireccionalidade ou pode forçar-se a representação num só sentido, através de setas.

Uma associação tem um **nome**, podendo optar-se por nomes diferentes, consoante o sentido da leitura.
Para indicação de quantos objectos são ligados, está disponível uma notação para a **multiplicidade**; exemplos: **0..*** ou simplesmente * significa «de zero a muitos», **1..*** significa «de 1 a muitos», **5..11** significa «de 5 a 11», **1** significa «exactamente 1».
De notar que um (1) é a multiplicidade por defeito; isto é, aquela que vinga, não estando indicada outra explicitamente.

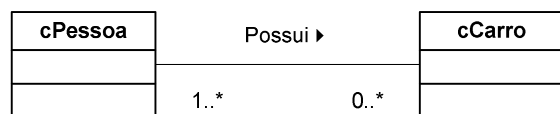


Figura 2

No exemplo da Figura 2, tem-se a associação **Possui**, entre as classes **cPessoa** e **cCarro**. A sua multiplicidade é de 1 ou mais, para zero ou mais.
Pretende-se a semântica de que uma pessoa pode possuir nenhum, um ou diversos carros; e que um carro pode ser possuido por uma ou mais pessoas.

É uma associação bidireccional implícita. A setinha que se vê à direita do nome **Possui**, pretende somente sugerir o sentido da leitura (**Pessoa Possui Carro**). Poderia também estar um nome no sentido contrário (**é possuido**), dizendo que um carro é posse de pessoa(s).
Estas setas de leitura não devem ser confundidas com setas de direcção da associação que, se explicitadas, se desenham na própria linha que representa a ligação entre as classes.

Para lá do **nome** e da **multiplicidade**, a linha que representa a associação pode ter, em cada extremidade, a indicação do **papel** de cada participante na associação.
No caso do exemplo da Figura 2, do lado da classe cPessoa poderia escrever-se «**proprietária**» e do lado de cCarro «**propriedade**».

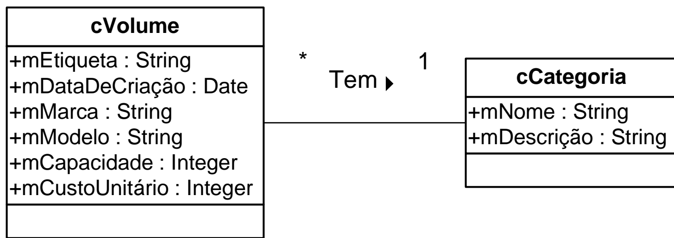


Figura 3

Neste outro exemplo da Figura 3, indica-se que qualquer objecto da classe **cVolume** (por exemplo um DVD-ROM) corresponde a um e a um só objecto da classe **cCategoria** (documentário, filme...); isto é, um volume **Tem** uma e uma só categoria.
Entretanto, uma categoria pode ser a de muitos volumes.

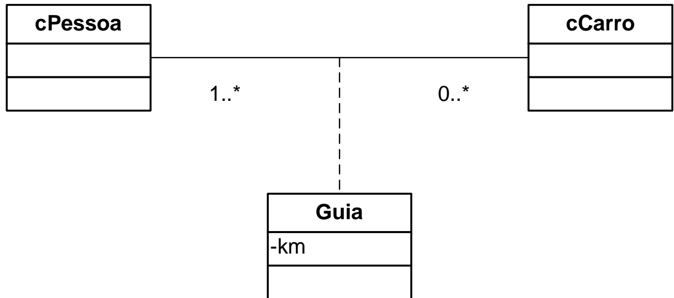


Figura 4

Quando uma associação é complexa ao ponto de ser possível pensar em membros próprios, a sua representação deve fazer-se por uma **«classe associativa»**.

O exemplo da Figura 4 pretende ilustrar uma situação dessas: as pessoas guiam carros, mas saber quantos quilómetros percorrem num carro específico, não é atributo nem de **cPessoa**, nem de **cCarro**, mas antes da relação associativa entre essas classes (**Guia**).

Para indicar que, numa associação, os objectos de alguma das classes intervenientes estão ordenados por algum critério, na extremidade da linha que corresponde à classe ordenada, escreve-se **{ordered}**, não havendo todavia «ordem por defeito».

Uma outra situação interessante é a que permite expressar que os objectos de uma classe se relacionam com uma OU outra classe, mas não com ambas em simultâneo. Veja-se o exemplo da Figura 5.

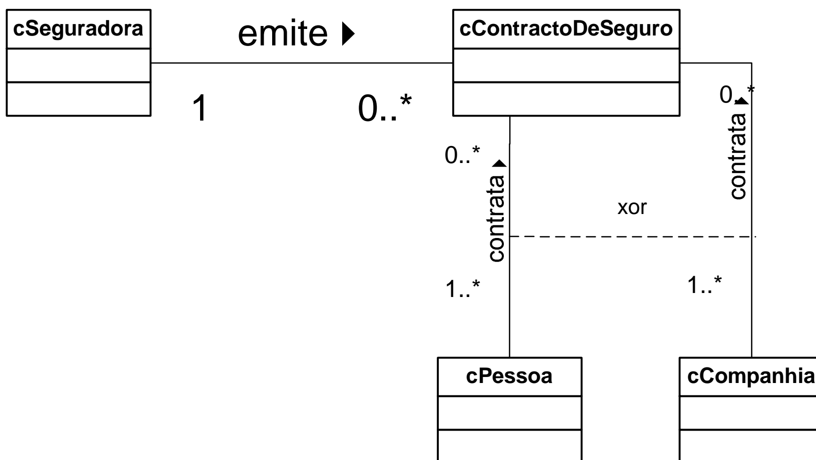


Figura 5

Neste exemplo pretende capturar-se que uma seguradora emite de zero a muitos contractos, contratados por pessoas e por companhias, mas sujeitos à restrição (constraint) de que certo contracto em particular (uma instância da classe **cContractoDeSeguro**) *ou* estará relacionada com uma instância de **cPessoa** *ou* com uma instância de **cCompanhia**, em exclusivo (operação lógica de ou-exclusivo / xor) e nunca com ambas em simultâneo!

© Artur Marques, 2005

Relações – Agregações

Nas associações sem agregação, as classes estão ao mesmo nível conceptual; nas agregações, existe um relacionamento «todo-parte»; isto é, uma das classes possui ou é composta por objectos da outra... ou seja, instâncias da classe agregadora são um «todo» de partes da classe agregada.

Veja-se o exemplo da Figura 6.

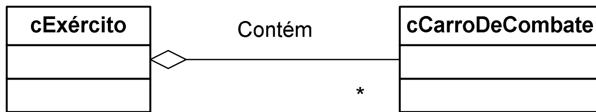


Figura 6

Esta é uma situação de **agregação normal**: carros de combate podem ser acrescentados e removidos, que não deixará de existir Exército – esta característica é importante para distinguir entre agregações simples/normais e composições.

As partes (carros de combate) estão na composição do «todo» (Exército). O «diamante» aberto representa a agregação das partes.

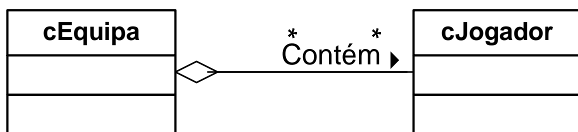


Figura 7

Uma **agregação partilhada** é aquela que se verifica quando as partes puderem fazer parte de qualquer «todo».

Por exemplo, imagine-se um desporto de equipas em que os jogadores podem jogar para qualquer equipa (as partes podem estar em qualquer todo). A representação de uma **agregação partilhada**, ilustra-se na Figura 7.

Uma última forma de agregação, é a **agregação composta** ou **composição**. Aqui a relação de pertença entre o todo e as partes é muito mais forte do que na agregação simples, como se as partes vivem-se dentro do todo. Ou seja, para lá da semântica de forte pertença, existe uma semântica de tempo de vida das partes limitado ao tempo de vida do todo, que assim é responsável pela criação e destruição daquilo que agrega.

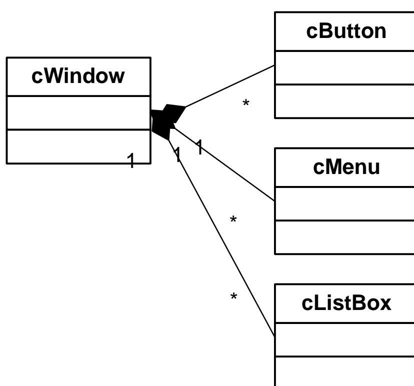


Figura 8

A composição representa-se como a agregação, mas com um «diamante fechado».

No exemplo da Figura 8, transmite-se que objectos da classe **cWindow** são uma **composição** de objectos das classes **cButton**, **cMenu** e **cListBox**, pelo que estes últimos só «vivem» enquanto a instância de **cWindow** correspondente viver.

© Artur Marques, 2005

Relações – Generalizações

As generalizações correspondem a situações em que se identificam elementos **gerais** e elementos deles derivados, mais **específicos**.
Os elementos mais específicos, **herdam** os atributos e os comportamentos dos elementos gerais, embora diferindo de alguma forma, por exemplo tendo novos atributos ou novos comportamentos, ou especializando algum comportamento herdado.

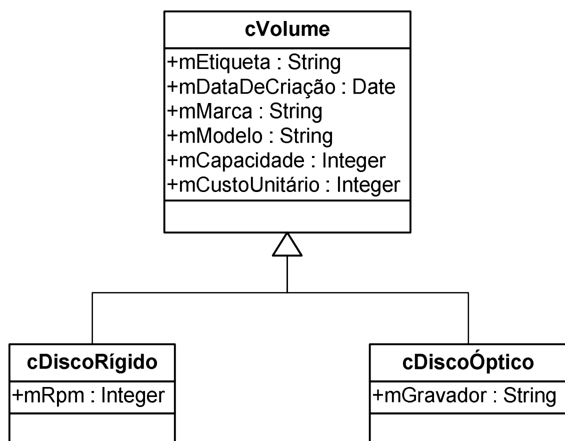


Figura 9

A Figura 9 corresponde a um exemplo de **Generalização**, em que as classes **cDiscoRígido** e **cDiscoÓptico** são casos específicos da (super)classe **cVolume**; ou seja, as suas instâncias não deixam de *ser* volumes – tendo pois todos os atributos e comportamentos da «classe mãe» – mas com diferenças próprias (neste caso apenas atributos) que justificaram a especialização.

As generalizações correspondem a relações do tipo «ser»:

- um disco óptico é um volume;
- um coelho é um mamífero;
- um mamífero é um ser vivo...

Relações de Dependência

A UML disponibiliza ainda relações de dependência para indicar que um elemento requer outro elemento, para algum propósito.

Numa dependência identificam-se um elemento «cliente» e um elemento «fornecedor»; o cliente é quem depende do fornecedor, pelo que uma alteração na especificação do segundo, pode comprometer o primeiro.

A representação da dependência faz-se por uma seta tracejada, do cliente para o fornecedor.

As dependências mais habituais são as de **«utilização»** e as de **«permissão»**.

Por exemplo, uma classe A para ser *utilizada* pode necessitar da presença de outra classe B. Em linguagens de programação, esta situação corresponde à necessidade de fazer a importação/inclusão da classe B.

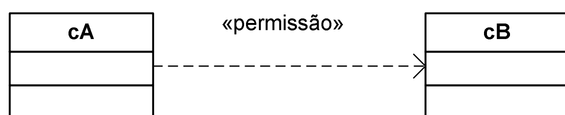


Figura 10

No exemplo da Figura 10, temos uma relação de dependência entre a classe cliente **cA** e a classe fornecedora **cB**.
É uma dependência representada pelo estereótipo de **«permissão»**.
Assim, objectos de cA necessitam de uma «permissão», para algum propósito, que só é concedida por objectos de cB.

© Artur Marques, 2005

Bibliografia

Carlos Costa – *Modelo Conceptual: Diagrama de Classes*. 2004.

Hans-Erik Eriksson, Magnus Penker, Brian Lyons, David Fado – *UML2 Toolkit*. Wiley Publishing, Inc., 2004.

Edward Yourdon, Peter Coad – *Object Oriented Analysis*. Yourdon Press Computing Series, 1991.

© Artur Marques, 2005