UNIVERSIDADE DO VALE DO ITAJAÍ
CENTRO DE CIÊNCIAS TECNOLÓGICAS DA TERRA E DO MAR
CURSO DE CIÊNCIA DA COMPUTAÇÃO

INTRODUÇÃO A UML

por

Marcelo Magnani
Carlos Henrique Bughi, MSc.

Itajaí (SC), novembro de 2009

SUMÁRIO

LISTA DE FIGURAS

1 ESCOPO

Este documento apresenta uma visão geral sobre a UML, focando principalmente no diagrama de classes e suas dependências.

1.1 UML - UNIFIED MODELING LANGUAGE

A UML (*Unified Modeling Language*) é uma junção de três notações principais e uma série de técnicas de modelagem retiradas de metodologias diversificadas existentes desde a década de 80. Durante esse tempo, tornou-se o padrão para modelagem de sistemas orientados a objetos em quase 70% dos centros de tecnologia da informação (PENDER, 2004).

Segundo seus criadores, Booch, Rumbaugh e Jacobson (2005), a UML é uma linguagem voltada à visualização, especificação, construção e documentação de um sistema utilizando um vocabulário e um conjunto de regras que permitem realizar a representação conceitual e física de um software.

Essa linguagem permite que desenvolvedores de sistemas especifiquem, visualizem e documentem os modelos de modo que admita escalabilidade, segurança e execução robusta, elevando o nível de abstração, facilitando o entendimento e identificando padrões de comportamento, viabilizando a criação de projetos modulares, resultando em componentes e bibliotecas de componentes que diminuem o tempo de desenvolvimento do sistema (LIMA, 2005).

Segundo Santos (2004), os principais diagramas da UML são:

- Diagramas de classes: representam as classes, seus conteúdos e relacionamentos com as demais classes do sistema;
- Diagramas de colaboração: demonstra a interação e comunicação entre os objetos dando ênfase aos seus relacionamentos;
- Diagramas de objetos: representam os objetos e seus relacionamentos em um determinado ponto no tempo;

- Diagramas de casos de uso: representam os casos de usos, atores e seus relacionamentos para descrever os requisitos do sistema;
- Diagramas de seqüência: demonstra a interação e comunicação entre os objetos no sistema dando ênfase à seqüência cronológica em que as comunicações ocorrem;
- Diagramas de estados: mostram os estados pelos quais um objeto ou uma interação passa durante seu tempo de vida;
- Diagrama de atividades: representam os fluxos das atividades do sistema;
- Diagramas de componentes: representam a organização e dependência entre um conjunto de componentes;
- Diagrama de implantação: mostra a configuração dos nós de processamento em tempo de execução e os componentes, processos e objetos que neles vivem;

A ferramenta Webcase contempla os diagramas de casos de uso e diagrama de atividades. Em seguida são apresentados os principais conceitos, seguindo as definições dos criadores da UML (Booch, Rumbaugh e Jacobson, 2005) e do autor do livro UML A Bíblia (Tom Pender, 2004), sobre os diagramas de classes e diagramas de seqüência, foco do projeto proposto.

1.1.1 Diagrama de classe

Pender (2004) afirma que o diagrama de classes se encontra no centro do processo de modelagem de objetos, sendo a principal fonte para geração de códigos e modelos, tanto se tratando de engenharia direta (*i.e.,* modelo para código) como de engenharia reversa (*i.e.,* código para modelo).

Um diagrama de classes mostra a estrutura estática do modelo, onde seus elementos são representados por classes, explicitando sua estrutura interna (*i.e.,* atributos e métodos) e seus relacionamentos com demais classes (LIMA, 2005).

Cada classe definida no diagrama deve possuir um nome único, preciso e conciso que representa o tipo de objeto representado por ela. É bastante comum

sistemas possuírem vários diagramas de classes, agrupados por assuntos em comum para se tornarem facilmente legíveis e compreensíveis (PENDER, 2004).

A Figura 1 ilustra como o diagrama de classes utiliza retângulos e linhas para representar os recursos do sistema e seus relacionamentos.

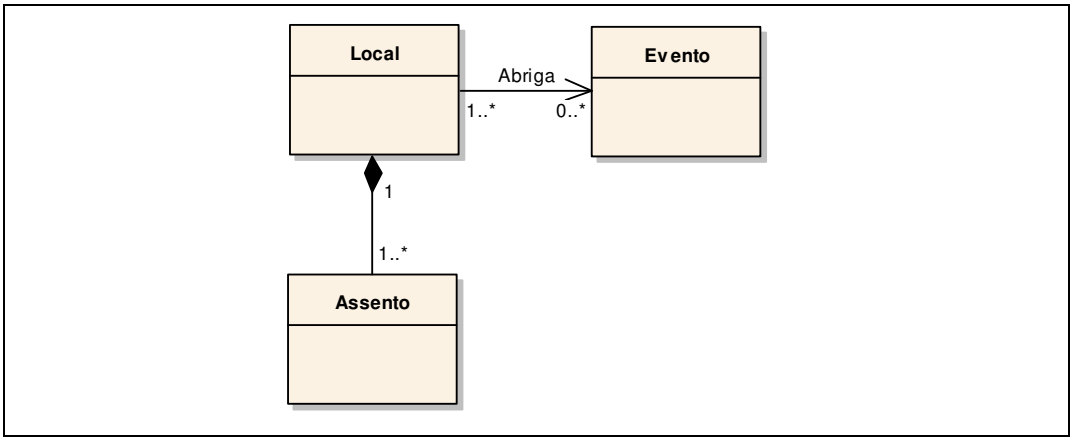

Figura 1. Diagrama de Classes com os relacionamentos entre as classes.
Fonte: Pender (2004).

Segundo Pender (2004), para compreender as características e funcionamento do diagrama de classes, é necessário que a diferença entre o conceito de objetos e classes esteja bem definida. Segundo conceito da *Object Managment Group* (OMG) (2005), um objeto é uma instância de uma classe que pode invocar operações e alterar valores de atributos definidos nela, enquanto a classe é uma definição dos recursos (operações) e características (atributos) que um objeto pode utilizar.

Na seqüência são explicados os componentes do diagrama de classes, com o objetivo de identificar suas características e funcionalidades.

1.1.1.1 Atributos

Os atributos de uma classe são responsáveis por armazenar as informações que um objeto possui. A seção de atributos situa-se abaixo do retângulo correspondente ao nome da classe, conforme ilustrado na Figura 2 (MILES e HAMILTON, 2006).

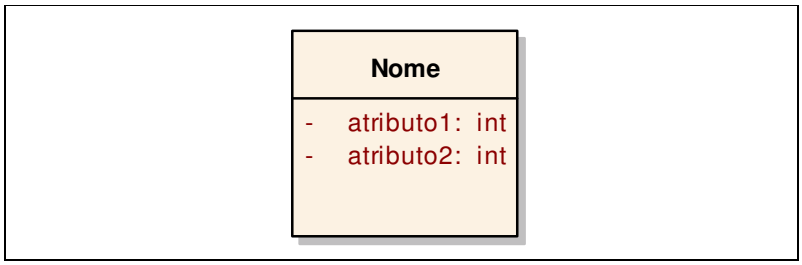

Figura 2. Exemplo do compartimento de atributos de uma classe.

Os valores que os atributos possuem representam as informações do objeto. Por exemplo, conforme ilustrado na Figura 3, em uma classe Aeronave os atributos podem ser: nome, capacidade e velocidade máxima. Os valores dos atributos variam para cada objeto do tipo Aeronave.

Também é possível que uma classe possua um atributo que referencie outra classe. Utilizando o exemplo da Figura 3, uma classe Vôo pode ter como atributos: horário da decolagem, horário do pouso, aeroporto de decolagem, aeroporto de pouso e aeronave. O atributo "aeronave" é um objeto do tipo (classe) Aeronave.

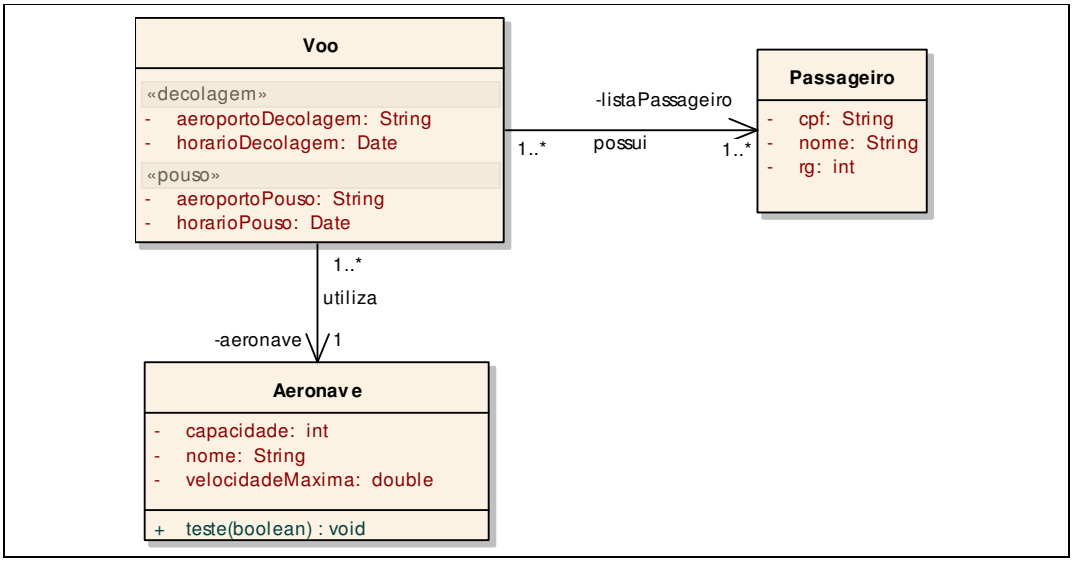

Figura 3. Diagrama de Classes com os atributos das classes Aeronave, Vôo e Passageiro.

Além disso, também é possível que uma classe possua uma coleção de objetos, definido por meio da relação de multiplicidade entre as classes (em programação, usualmente são utilizados vetores para esse tipo de caso). Ainda utilizando o exemplo ilustrado na Figura 3 para representar essa possibilidade, pode-se criar uma classe

denominada Passageiro, que possui como atributos: nome, CPF e RG. Adicionando um atributo na classe Vôo com o nome lista de passageiros. O atributo lista de passageiros é uma lista de objetos do tipo (classe) Passageiro.

Segundo Pender (2004), para garantir a integridade dos atributos e do sistema onde eles são utilizados, a UML permite utilizar algumas propriedades para cada atributo através da seguinte sintaxe: [visibilidade] [/] nome [: tipo] [multiplicidade] [= valor padrão] [string de propriedade]. A Figura 4 exemplifica a visualização dos atributos em uma classe.

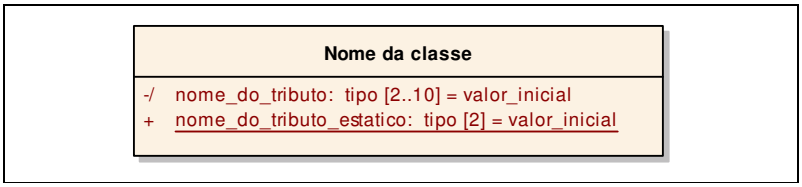

Figura 4. Propriedade dos atributos.

A seguir são apresentados os conceitos de cada uma das propriedades dos atributos.

1.1.1.2 Visibilidade

O termo "visibilidade" é utilizado para referenciar o escopo de acesso permitido para atributos e operações em uma classe, podendo ser classificado como privada (*private*), pacote (*package*), pública (*public*) e protegida (*protected*).

Quadro 1. Símbolos de visibilidade

Nível de visibilidade	**Símbolo**
Private (Privada)	-
Package (Pacote)	~
Public (Pública)	+
Protected (Protegida)	#

Fonte: Adaptado de Pender (2004).

O Quadro 1 contém os símbolos sugeridos pela UML para designar os níveis de visibilidade em um diagrama. Na seqüência são apresentadas as definições para cada tipo de visibilidade.

- Privada (-): limita o acesso aos atributos e operações para objetos da mesma classe. Por exemplo, os atributos e operações privadas da Classe A não podem ser acessíveis à Classe B ou qualquer outra classe do sistema, mesmo em relações de especialização.
- Pacote (~): permite o acesso de atributos e funções pelos objetos de classes situadas no mesmo pacote. Por exemplo, a Classe A e Classe B pertencem ao Pacote AB, enquanto a Classe C e Classe D pertencem ao Pacote CD. Os atributos e operações declarados com visibilidade de pacote na Classe A poderão ser acessados pela Classe B, porém não pela Classe C e Classe D.
- Pública (+): permite o acesso de atributos e operações por todas as classes do sistema.
- Protegida (#): limita o acesso de atributos e operações pelas subclasses (classes que herdam a outra classe). Por exemplo, a Classe B herda a Classe A. Os atributos protegidos da Classe A podem ser acessados somente pela Classe A e Classe B.

1.1.1.3 Valor derivado

O valor derivado é representado com uma barra (/) antes do nome do atributo (*i.e.,* a ausência da barra indica que é um valor básico). Um atributo derivado representa que é necessário algum tipo de tratamento para obtenção do seu valor final por regra.

Pender (2004) afirma que, dependendo da freqüência e custo para obter o valor do atributo, pode-se optar por armazenar o valor obtido ou não. Em casos onde é preciso cálculos complexos e demorados para chegar ao valor esperado opta-se por armazenar o mesmo, caso contrário, o valor é calculado sempre quando for necessário exibi-lo ou utilizá-lo. Por exemplo, em uma classe com os atributos distância, tempo e velocidade, o atributo velocidade é um valor derivado, pois o mesmo é obtido através da divisão da distância pelo tempo.

1.1.1.4 Nome

O nome do atributo é utilizado para identificar o mesmo, logo é obrigatório e deve ser único em cada classe. Ele deve expressar claramente o tipo de informação armazenada a fim de facilitar a programação e manutenção do código.

1.1.1.5 Tipo de dado

O tipo do dado identifica a espécie da informação que será armazenada no atributo. Pode ser uma referência a um tipo de dado primitivo (*e.g.,* inteiro, caractere ou palavra), referência a um tipo específico da linguagem (*e.g., float, array, long*) ou uma referência a outra classe do sistema.

1.1.1.6 Multiplicidade

A multiplicidade especifica a quantidade de valores que podem estar associados a um elemento do modelo, podendo ser um intervalo de valores, um valor específico, um intervalo sem limite ou um conjunto de valores discretos. O conceito de multiplicidade, assim como de visibilidade, é utilizado em diversos elementos da UML, principalmente em atributos e relacionamentos (LARMAN, 1999).

Há duas formas de representar a multiplicidade nos diagramas, exemplificadas na Figura 5. No caso onde a multiplicidade é representada na forma de texto (*e.g.,* atributos) o valor deve ficar entre colchetes ([]), já quando se refere a relacionamento de elementos do diagramas, o valor não possui colchetes.

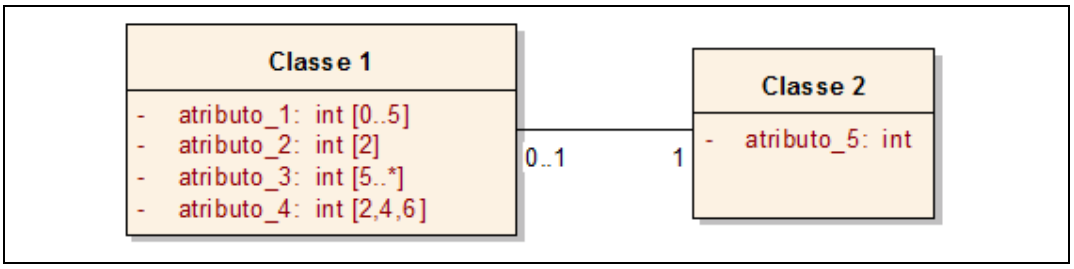

Figura 5. Diagrama de classes com multiplicidade definida nos atributos e no relacionamento entre as classes.

- Intervalo de valores: Um intervalo de valores indica o valor mínimo e máximo (separados por dois pontos) da multiplicidade do elemento. Por exemplo, [0..5] ou 0..5 representa de zero a cinco, [1..10] ou 1..10 representa de um a dez.
- Valor específico: Indica que o valor máximo e mínimo da multiplicidade são iguais. Por exemplo, a multiplicidade dois pode ser representada na forma [2..2] ou [2].
- Intervalo sem limite: Quando o valor da multiplicidade não possui restrição é utilizado um asterisco (* ou [*]) isolado, significando que pode variar de

zero até infinito. Já nos casos onde somente o valor mínimo é especificado, utiliza-se asterisco na posição do valor máximo, por exemplo [5..*], indicando que o valor mínimo da multiplicidade é igual a cinco e não há limite para o valor máximo.

- Conjunto de valores discretos: Quando há uma lista de possíveis valores específicos, os mesmos são representados separados por vírgula. Por exemplo [2, 4, 6] indica que a quantidade de associações do elemento pode ser dois, quatro ou seis.

1.1.1.7 Valor padrão

O valor padrão representa o valor inicial que o atributo irá conter na criação do objeto e é representado pelo sinal de igual (=) seguido do valor. Essa propriedade é utilizada para facilitar o uso do sistema e proteger a integridade do mesmo contra omissão de valores.

1.1.1.8 String de propriedade

A string de propriedade é utilizada para representar alguma definição que não existe um local designado para ela. É expressa através de chaves ({}) e geralmente é utilizada em casos onde há uma regra para garantir a integridade do atributo ou para algum comentário pertinente ao mesmo.

1.1.1.9 Designação de nível de classe *versus* nível de instância

Segundo Pender (2004), a maioria dos atributos de uma classe pertence a um objeto específico, sendo designados como atributos de nível de instância e são acessíveis somente ao objeto que o possui. Já os atributos acessíveis sem a necessidade de haver uma instância da classe (geralmente nomeados de atributos estáticos pelas linguagens de programação) devem ser designados como atributos de nível de classe e são identificados no diagrama de classe por serem sublinhados.

Um exemplo prático de utilização de atributos estáticos é no controle de venda de ingressos numerados seqüencialmente. A classe denominada Ingresso possui um atributo estático contendo o número da última venda e garante que não haja duplicidade, pois a cada venda efetivada esse valor é incrementado.

1.1.1.10 Operações

O compartimento com as operações da classe encontra-se abaixo do retângulo dos atributos conforme exibido na Figura 6. As operações de uma classe definem os comportamentos e serviços que o objeto pode oferecer (GUEDES, 1999).

É comum utilizar estereótipos para agrupar um conjunto de operações, separando-as de acordo com suas funcionalidades ou critérios que facilitem a manutenção da classe, conforme ilustrado na Figura 6.

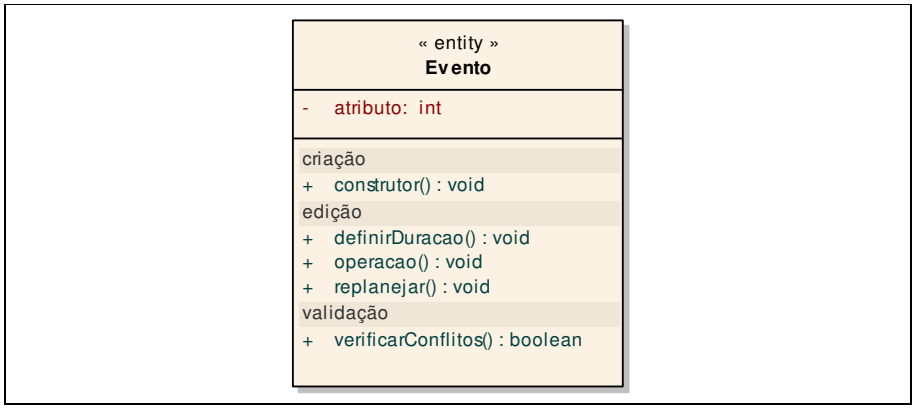

Figura 6. Exemplo do compartimento de operações de uma classe com estereótipos criação, edição e validação.

As propriedades de cada operação são apresentadas na seqüência e seguem a seguinte notação: [visibilidade] [nome] ([lista de parâmetros]): [tipo de dado do retorno] [{string de propriedade}].

1.1.1.11 Visibilidade

Assim como nos atributos, a visibilidade pode ser pública, privada, pacote e protegida. Os detalhes dos tipos de visibilidade são explicados no item “Visibilidade” da seção 1.1.1.1.

1.1.1.12 Nome

O nome da operação identifica a funcionalidade específica da mesma, por isso deve ser o mais significativo e expressivo possível.

Algumas linguagens permitem que haja mais de uma operação com o mesmo nome, porém a assinatura (combinação do nome, lista de parâmetros e retorno da função) deve ser única. Essa prática é denominada sobrecarga de operação.

1.1.1.13 Lista de parâmetros

Segundo Booch, Rumbaugh e Jacobson (2005), a lista de parâmetros é uma lista ordenada dos valores de entrada de uma operação. Cada valor de entrada deve ser definido do mesmo modo utilizado para definir atributos, separando-os por vírgulas e entre parênteses, por exemplo, nome_da_operação (valor_1: Inteiro, valor_2: Inteiro).

A passagem de parâmetros não é obrigatória e opta-se por passar ou não esses valores dependendo da funcionalidade da operação (mesmo quando não é necessária a passagem de parâmetros, deve colocar parênteses vazios). Uma operação que soma dois valores de entrada recebidos por parâmetros deverá ter a seguinte sintaxe: soma (valor_1: Inteiro, valor_2: Inteiro).

1.1.1.14 Tipo de dado do retorno

O tipo de dado do retorno representa o tipo de informação que a operação irá retornar (diferentemente dos parâmetros, o retorno especifica somente o tipo, não sendo necessário especificar o nome). Por exemplo, no caso de uma operação que some dois valores e retorne o resultado da operação, o retorno será do tipo "inteiro" e a sintaxe da operação será: soma (valor_1: Inteiro, valor_2: Inteiro):Inteiro.

Tecnicamente, a UML 2.0 permite retornar mais de um tipo de valor, porém na grande maioria das linguagens de programação só é possível retornar um valor, fazendo-se necessário utilizar vetores ou objetos para atender tal necessidade.

1.1.1.15 String de propriedade

Como nos atributos, a string de propriedade permite acrescentar alguma informação adicional que não se encaixe em nenhum dos outros elementos predefinidos. Geralmente é utilizada para expressar a descrição da implementação da operação.

1.1.1.16 Designação de nível de classe *versu*s nível de instância

Da mesma forma que os atributos, as operações também possuem designação de nível, onde operações de nível de instância só são acessíveis a um objeto em específico (instância da classe), enquanto as operações de nível de classe são acessíveis pelo sistema sem a necessidade de instanciar um objeto.

Na maioria das linguagens, as operações de nível de classe são denominadas como operações estáticas e são representadas no diagrama de classe sendo sublinhadas.

1.1.1.17 Estereótipo de classes

Segundo Pender (2004), a UML permite definir o tipo de funcionalidade de uma classe utilizando estereótipos, tornando possível, por exemplo, distinguir as classes pertencentes ao contexto do sistema (elementos da entidade) das classes de controle do software (elementos de controle) e das classes de interface (telas e interface de comunicação). Em uma aplicação desenvolvida para uma escola, as classes Aluno, Professor e Nota são elementos da entidade, as classes responsáveis por gravar dados no banco de dados são elementos de controle, e as classes que interagem com os usuários (telas e formulários) são elementos da interface.

Para prover essa distinção, a UML modela um estereótipo delimitando o nome com divisas (aspas francesas) como no caso de <<entity>> (endidade), <<control>> (controle) ou <<boundary>> (interface).

Os estereótipos não fazem parte do nome da classe e não geram nenhum código para a mesma. A finalidade dos estereótipos é prover coerência no tratamento de elementos comuns nos diagramas UML.

Pender (2004) afirma que, toda classe com o estereótipo <<entity>> é conhecida por desempenhar um papel especifico dentro do projeto do sistema, possui conhecimento apenas sobre si mesma e seus relacionamentos imediatos, e tem a responsabilidade de assegurar sua integridade, independente de onde esteja sendo utilizada. As classes do estereótipo <<control>> não possui nenhuma informação sobre si mesma, somente tem conhecimento dos seus recursos que precisa manipular ou dirigir, e tem a responsabilidade de controlar o uso e o comportamento dos mesmos e outros elementos de software a sua disposição. Já as classes do estereótipo <<boundary>> são responsáveis pela interface do sistema, podendo ser telas, ou qualquer outro meio que o sistema se comunique com usuários ou outros sistemas.

Outra forma de representar o estereótipo de uma classe é utilizando ícones que são representações alternativas do mesmo, como mostra a Figura 7.

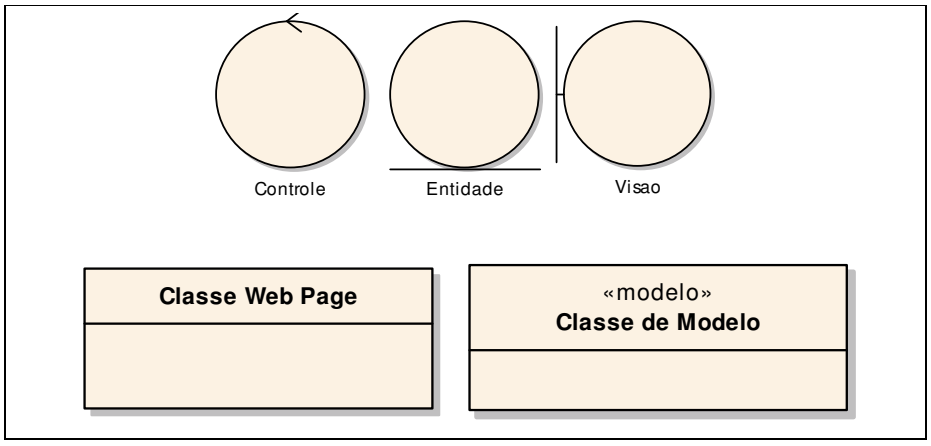

Figura 7. Exemplo de ícones para representação dos estereótipos.

Além dos estereótipos citados (*i.e.,* entidade, controle e interface), pode-se especificar outros tipos de acordo a funcionalidade da classe.

1.1.2 Relacionamentos

Segundo Booch, Rumbaugh e Jacobson (2005), para coordenar as interações entre as classes é necessário que elas possuam ligações, estejam cientes uma da outra e conheçam as regras de como devem interagir.

A UML define três tipos de relacionamentos que são utilizados para descrever a interação entre os recursos (classes). São eles: associação, generalização e dependência.

1.1.2.1 Associação

As associações são as regras que controlam o relacionamento entre duas classes. Por exemplo, uma associação entre uma classe denominada Local e uma classe denominada Evento significa que uma instância de Local tem uma associação com uma instância de Evento (PENDER, 2004).

Assim como nas classes, atributos e operações, os nomes das associações são importantes, pois expressam ao leitor a intenção do relacionamento.

Semelhante à multiplicidade dos atributos, apresentada na seção 1.1.1.1, a multiplicidade das associações referem-se ao número válido de objetos que podem estar relacionados. A diferença significativa entre a multiplicidade de atributos e associações,

é que nas associações não são utilizados colchetes ([]) como delimitadores para os valores possíveis, conforme ilustrado na Figura 8.

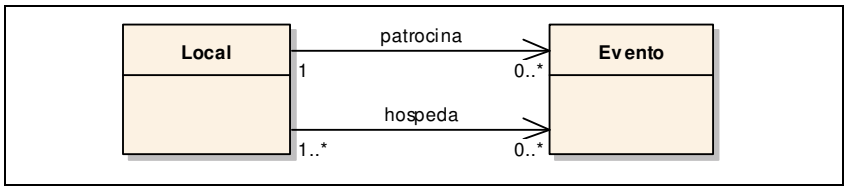

Figura 8. Exemplo da utilização de multiplicidade.
Fonte: Adaptado de Pender (2004).

Existem ocasiões onde os objetos participantes de um relacionamento pertencem à mesma classe, necessitando criar uma associação reflexiva para representar tal caso. Por exemplo, um funcionário pode possuir nenhum ou vários funcionários subordinados a ele. Nesse cenário, um objeto da classe Funcionário possui uma ligação com outros objetos também da classe Funcionário, sendo assim, há uma ligação reflexiva nessa classe, conforme ilustrado na Figura 9.

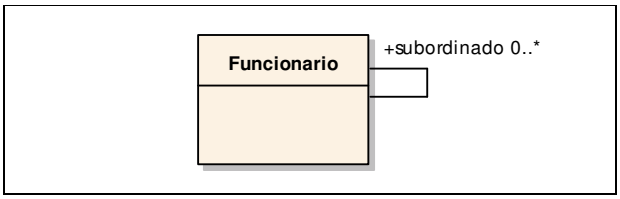

Figura 9. Associação reflexiva.

Nos casos onde, além do relacionamento entre as classes, há necessidade de armazenar informações referentes a esse relacionamento, são utilizadas as chamadas classes de associação. Essas classes possuem atributos com informações referentes à ligação de duas classes. No exemplo ilustrado na Figura 10, a classe Local se relaciona com várias classes de Evento, e a classe de associação LocalEvento é utilizada para armazenar a data de inicio e término de um evento em um determinado local.

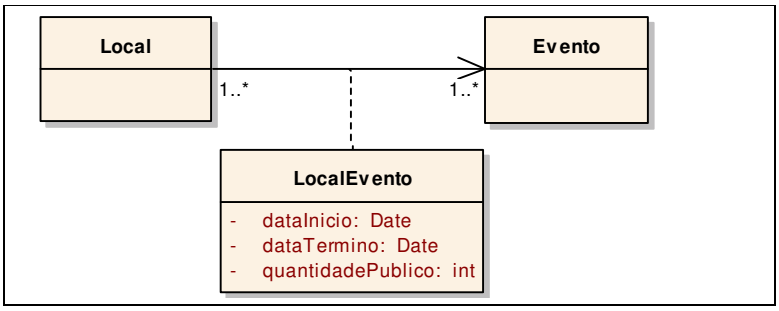

Figura 10. Exemplo de classe de associação.
Fonte: Adaptado de Pender (2004).

Em uma associação simples, cada classe é independente da outra, apenas há a necessidade que ambas se comuniquem. Em casos mais complexos, que exigem a definição de relacionamentos hierárquicos, é utilizado um tipo especifico de associação denominado agregação.

Na agregação de objetos é preciso haver um ponto de controle, um chefe, um único objeto que represente a interface do conjunto e assuma a responsabilidade por coordenar o comportamento da agregação, ou seja, um conjunto de objetos é coordenado por um objeto único que possui o controle sobre os demais (PENDER, 2004).

A atribuição do controle centralizada em um único objeto pode estar em muitos níveis dentro da hierarquia da agregação. Por exemplo, um motor poderia ser o controlador de suas partes, e o carro ser o controlador do motor e de outras partes do carro.

Para representar uma associação de agregação, é utilizada uma linha entre as duas classes com um losango próximo a classe controladora da agregação, conforme demonstrado na Figura 11.

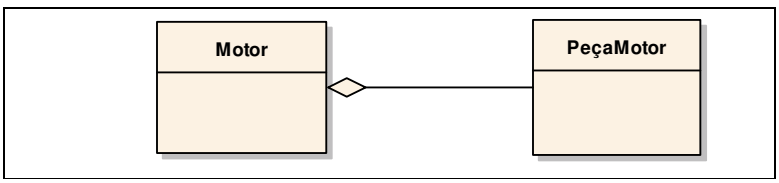

Figura 11. Exemplo de agregação entre classes.
Fonte: Adaptado de Pender (2004).

Já nos casos onde os objetos membros da agregação não existem sem o objeto controlador (agregado) é utilizada a composição, que seria um subconjunto da agregação. Na composição o tempo de vida do objeto membro depende do tempo de vida do objeto agregado.

A composição é considerada como uma agregação "forte", por exemplo, uma peça de um evento precisa estar associada a um evento (sem o evento a peça não existe). Para representar esse tipo de agregação, é utilizada uma linha entre as duas classes com um losango sólido próximo a classe controladora da composição, conforme Figura 12 (PENDER, 2004).

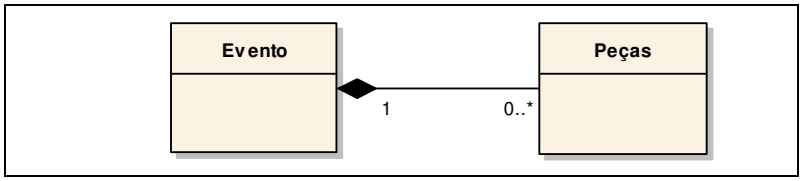

Figura 12. Exemplo de composição entre classes.
Fonte: Pender (2004).

1.1.2.2 Generalização

A generalização é utilizada para organizar as propriedades de um conjunto de objetos que compartilham a mesma finalidade. Nesse tipo de relacionamento, que também é conhecido como herança, o elemento mais específico herda as características do elemento mais geral, porém contem informações adicionais.

Termos como "espécie de" e "tipo de" normalmente são utilizados para descrever o relacionamento de generalização entre as classes. Por exemplo, um musical é um tipo de peça, uma peça é um tipo de evento.

Para representar esse tipo de relacionamento, é utilizada uma linha entre os dois objetos com um triângulo próximo ao objeto mais geral, conforme exemplo da Figura 13.

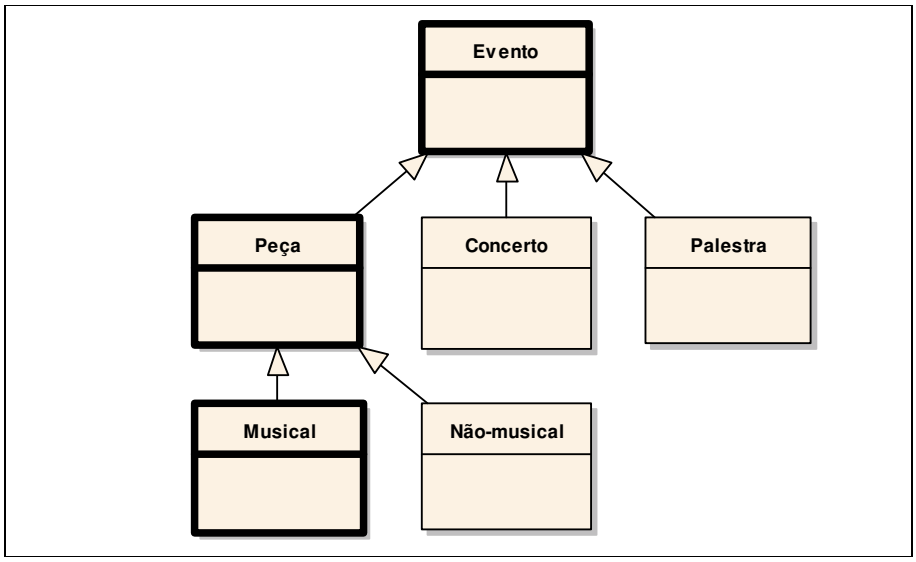

Figura 13. Exemplo de generalização de classes.

Fonte: Pender (2004).

Em palavras, o diagrama acima poderia expressar: “Um musical é um tipo de peça, e uma peça é um tipo de evento”.

Uma classe é considerada abstrata quando não possui as informações ou comportamentos necessários para criar um objeto. Devido a isso, as classes abstratas só podem ser utilizadas quando assumirem papel de superclasses. Nesses casos a fonte itálica é utilizada no nome das classes para representar que a classe é abstrata.

O exemplo Figura 14 ilustra um caso de utilização de classes abstratas. A classe Pessoa é uma classe abstrata, pois sempre terá uma subclasse do tipo Física ou Jurídica.

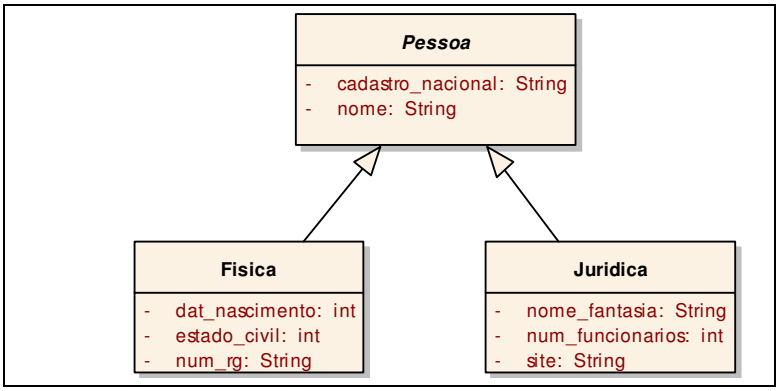

Figura 14. Exemplo de classe abstrata.

1.1.2.3 Dependência

Pender (2004), afirma que o relacionamento de dependência é utilizado nos casos onde uma classe necessita acessar recursos específicos de outra classe, e em caso de mudança na classe que fornece os recursos é necessário que haja mudança também na classe que utiliza os recursos.

A dependência representa um relacionamento cliente-fornecedor, por exemplo, no caso de da classe denominada Marketing acessar recursos da classe denominada Vendas, caso haja uma mudança na classe Vendas, a classe Marketing não poderá mais realizar o acesso dos recursos sem antes passar por uma mudança também.

Para representar esse tipo de relacionamento é utilizada uma linha tracejada entre as duas classes com uma seta na direção cliente-fornecedor, conforme demonstrado na Figura 15.

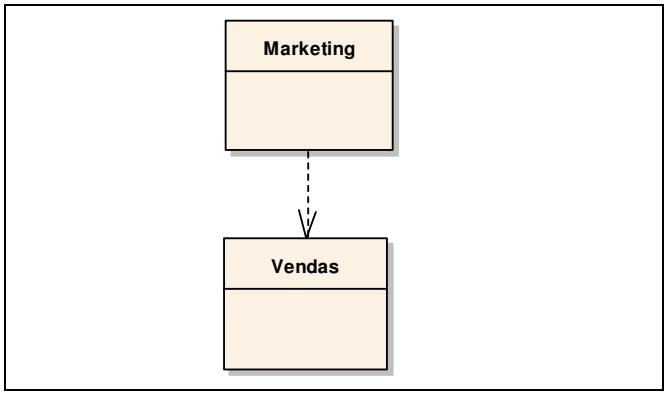

Figura 15. Exemplo de dependência entre classes.
Fonte: Pender (2004).

1.1.2.4 Interface

Como citado no item 1.1.1.10 referente à operações, a assinatura de uma operação consiste na combinação do seu nome, lista de parâmetros e valor de retorno. O conjunto de assinaturas das operações define uma interface (BOOCH; RUMBAUGH; JACOBSON, 2005).

Uma interface não é uma classe real, e sim uma decodificação, sendo assim possui algumas restrições que as classes não tem, como não possuírem atributos e suas operações não serem implementadas (somente as assinaturas das operações).

Para representar as interfaces, é utilizada uma linha pontilhada entre duas classes com um triângulo junto à interface, conforme ilustrado na Figura 16. Por exemplo, a interface denominada Produto contém as operações comprar(), estocar(), vender() e darPreco(). As classes Camisa e Perfume "realizam" ou "implementam" a interface Produto.

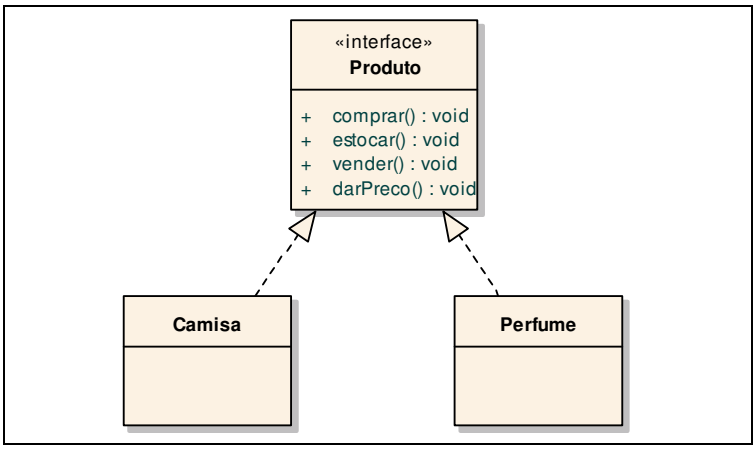

Figura 16. Exemplo de interface.
Fonte: Pender (2004).

1.1.2.5 Navegabilidade

Considerando uma associação simples entre duas classes, pode-se considerar que é possível navegar entre elas, passando de uma classe para outra quando a associação é bidirecional. Porém há casos onde somente uma classe deve se capaz de navegar até outra classe, caracterizando uma navegação com somente uma direção, ou seja, direcional (PENDER, 2004).

As associações direcionais são modeladas com uma seta na extremidade da associação, se a seta estiver presente, a classe para onde a seta aponta é navegável. Caso não haja seta, considera-se que a associação é bidirecional.

O exemplo da Figura 17, indica que a classe Usuário pode ter várias senhas e através da classe Usuário é possível navegar até a classe Senha, porém o contrário não é permitido através dessa associação.

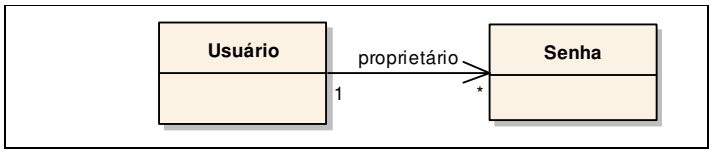

Figura 17. Navegabilidade entre classes.

Fonte: Booch, Rumbaugh e Jacobson (2005).

2 REFERÊNCIAS BIBLIOGRÁFICAS

ALENCAR, A. J.; SCHMITZ, E. A. **Análise de risco em gerência de projetos**. Rio de Janeiro: Brasport, 1999. Disponível em: < http://books.google.com/books?id=3ByM-bYzTNoC&pg=PP1&dq=Análise+de+Risco+em+Gerência+de+Projetos&hl=pt-BR>. Acessado em: 06 mai. 2009

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **UML Guia do Usuário**. Rio de Janeiro: Campus, 2005.

CORRÊA, Fábio R.; BORGES, Robson M., Manuel. **Análise orientada a objetos com UML**. São Leopoldo. s. d.. Disponível em: <http://www.inf.unisinos.br/~barbosa/paradigmas/consipa_1/artigos/uml.pdf>. Acesso em: 14 mar. 2009. Módulo do curso Analistas e projetistas OO oferecido pelo Instituto de Informática da UNISINOS.

ESPÍNDOLA, Evandro Camarini. **A importância da modelagem de objetos no desenvolvimento de sistemas**. 27 abr. 2007. Disponível em: <http://www.linhadecodigo.com.br/Artigo.aspx?id=1293/>. Acesso em: 14 mar. 2009.

FEDELI, A. *et al*. **Orientação a objeto com prototipação**. São Paulo: Cengage, 2002. Disponível em: < http://books.google.com/books?id=oJ8I6wMWtw4C&printsec=frontcover&hl=pt-BR>. Acesso em: 20 abr. 2009

GUEDES, G. T. A. **UML - Uma abordagem prática**. São Paulo: Novatec, 1999. Disponível em: <http://books.google.com/books?id=1UGl0iZIhBUC&printsec=frontcover&dq=UML+-+Uma+Abordagem+Prática&hl=pt-BR>. Acessado em: 03 abr. 2009

LARMAN, C. **Utilizando UML e padrões:** uma introdução à análise e ao projeto orientados a objetos e ao desenvolvimento iterativo. Porto Alegre: Bookman, 1999.

LIMA, A. S. **UML 2.0:** do requisito à solução. São Paulo: Érica, 2005.

MEDEIROS, E. **Desenvolvendo Software com UML**. São Paulo: Pearson Makron Books, 2006.

MILES, R.; HAMILTON, K. **Learning UML 2.0**. Sebastopol, USA: O'Reilly, 2006. Disponível em: <http://books.google.com/books?id=hy3VIZg2qRQC&printsec=frontcover&dq=Learning+UML+2.0&hl=pt-BR>. Acesso em: 17 mar. 2009

OBJECT MANAGEMENT GROUP. **Unified Modeling Language: Superstructure**, Needham, 2005. Disponível em: <http://www.omg.org/spec/UML/2.0/Infrastructure/PDF/>. Acesso em: 25 mai. 2009

PENDER, T. **UML a Bíblia**. Rio de Janeiro: Campus, 2004.

PICHILIANI, M.C., HIRATA C.M. **Uma ferramenta de software livre para apoiar a construção colaborativa de diagramas UML**. VIII Workshop sobre software livre. Porto Alegre, 2006.

REZENDE, Denis Alcides. Introdução à Engenharia de Software. **Engenharia de software e sistemas de informação**. São Paulo: Brasport, 2006. Disponível em: <http://books.google.com/books?hl=pt-BR&lr=&id=rtBvl_L-1mcC&oi=fnd&pg=PA170&dq=+documentar+software&ots=9xdpZK2v_p&sig=WiFmrdqpHfxZINkrvhyIc746nL4#PPA5,M1>. Acesso em: 12 mar. 2009

SPARX SYSTEM. **Sparx System**. 2009. Disponível em: <http://www.sparxsystems.com.au/>. Acesso em: 10 jun. 2009

YANG, H. **Software evolution with UML and XML**. Hershey, USA: IGI Publishing, 2005. Disponível em: <http://books.google.com/books?id=r73UvaPNbFgC&pg=PP1&dq=Software+evolution+with+UML+and+XML&hl=pt-BR>. Acesso em: 15 mai. 2009