

PROF. RAFAEL VARGAS MESQUITA

TÉCNICO EM INFORMÁTICA

Análise e Projeto de Sistemas

**CACHOEIRO DE ITAPEMIRIM
IFES
2012**

Governo Federal
Ministro da Educação
Fernando Haddad

Instituto Federal do Espírito Santo - Ifes
Reitor
Denio Rebello Arantes

Pró-Reitora de Ensino
Cristiane Tenan Schlittler dos Santos

Diretor-geral do Campus Cachoeiro de Itapemirim
Jorge Mário de Moura Zuany

Diretora do CEAD – Centro de Educação a Distância
Yvina Pavan Baldo

Coordenadores da UAB – Universidade Aberta do Brasil
Marize Lyra Silva Passos
José Mario Costa Junior

Curso de Licenciatura em Informática

Coordenação de Curso
Alexandre Fraga de Araujo

Designer Instrucional
Edmundo Rodrigues Junior
Marcelo Albuquerque Schuster

Professor Formador/Autor
Rafael Vargas Mesquita

M582a Mesquita, Rafael Vargas
 Análise e projeto de sistemas / Rafael Vargas
 Mesquita. Cachoeiro de Itapemirim: Ifes, 2011.
 103 p. : il.

ISBN 978-85-62934-49-0

1. Software - desenvolvimento. 2. Linguagem de
modelagem unificada (UML). 3. Gerenciamento de
projetos – computação. 4. Simulação (Computadores)
I. Instituto Federal de Educação, Ciência e Tecnologia do
Espírito Santo. II. Título.

CDD: 005.1

DIREITOS RESERVADOS

Instituto Federal do Espírito Santo - Ifes

Av. Rio Branco, 50 – Santa Lúcia – Vitória – ES – CEP 29.056-255 – Telefone: (27) 3227-5564

Créditos de autoria da editoração

Capa: Juliana Cristina da Silva

Projeto gráfico: Juliana Cristina e Nelson Torres

Iconografia: Nelson Torres

Editoração eletrônica: Gráfica Editora Fátima

Revisão de texto:

Esther Ortlieb Faria de Almeida

COPYRIGHT – É proibida a reprodução, mesmo que parcial, por qualquer meio, sem autorização escrita dos autores e do detentor dos direitos autorais.

Olá, Aluno(a)!

É um prazer tê-lo(a) conosco.

O Ifes – Instituto Federal do Espírito Santo – oferece a você, em parceria com as Prefeituras e com o Governo Federal, o Curso de Graduação em Licenciatura em Informática, na modalidade a distância. Apesar de este curso ser ofertado a distância, esperamos que haja proximidade entre nós, pois, hoje, graças aos recursos da tecnologia da informação (e-mails, chat, videoconferência, etc.), podemos manter uma comunicação efetiva.

É importante que você conheça toda a equipe envolvida neste curso: coordenadores, professores especialistas, tutores a distância e tutores presenciais, porque, quando precisar de algum tipo de ajuda, saberá a quem recorrer.

Na EaD – Educação a Distância, você é o grande responsável pelo sucesso da aprendizagem. Por isso, é necessário que se organize para os estudos e para a realização de todas as atividades, nos prazos estabelecidos, conforme orientação dos Professores Formadores e Tutores.

Fique atento às orientações de estudo que se encontram no Manual do Aluno!

A EaD, pela sua característica de amplitude e pelo uso de tecnologias modernas, representa uma nova forma de aprender, respeitando, sempre, o seu tempo.

Desejamos-lhe sucesso e dedicação!

Equipe do Ifes

ICONOGRAFIA

Veja, abaixo, alguns símbolos utilizados neste material para guiá-lo em seus estudos.

Fala Professor



Fala do Professor.

Conceitos



Conceitos importantes. Fique atento!

Atividades



Atividades que devem ser elaboradas por você, após a leitura dos textos.

Indicações



Indicação de leituras complementares, referentes ao conteúdo estudado.

Atenção



Destaque de algo importante, referente ao conteúdo apresentado. Atenção!

Reflexão



Reflexão/questionamento sobre algo importante referente ao conteúdo apresentado.

Anotações



Espaço reservado para as anotações que você julgar necessárias.

APRESENTAÇÃO

Olá!

Meu nome é Rafael Vargas Mesquita, responsável pela disciplina de Análise e Projeto de Sistemas. Atuo como professor do IFES há cinco anos. Sou graduado em Ciência da Computação (2001) e Mestre em Gestão da Qualidade de Software (2007), ambos pela Universidade Federal de Lavras (UFLA). Atualmente, estou cursando Doutorado pela Universidade Estadual Norte Fluminense (UENF) e minhas áreas de interesse são: Engenharia de Software, Qualidade de Software, Melhoria de Processo de Software, Medição de Software, Análise e Projeto de Sistemas Orientados a Objetos.

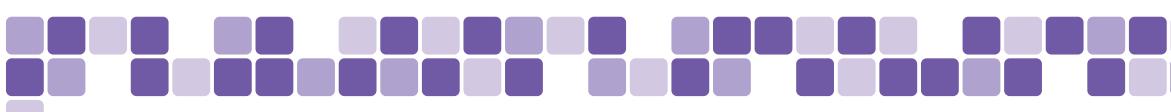
Nesta disciplina serão discutidos assuntos relacionados à disciplina de Análise e Projeto de Sistemas. Muitos dos conceitos aqui discutidos já foram, e serão, vistos com diferentes níveis de profundidade nas disciplinas de Engenharia de Software e Gestão de Projetos, respectivamente.

O objetivo deste material é auxiliá-lo no estudo da disciplina de Análise e Projeto de Sistemas, por meio de dicas e sugestões que destacam os pontos mais importantes a serem estudados. Neste material daremos um enfoque mais prático à disciplina, citando exemplos sempre que possível, e dando sugestões de leituras complementares.

Em geral, para ser bem sucedido neste curso é importante que você, além dos estudos regulares, faça as atividades sugeridas e estude os exercícios resolvidos evitando, dessa forma, o acúmulo de conteúdo. A participação no ambiente e, consequentemente, a interação com os colegas e tutores é extremamente importante para o sucesso na disciplina e no curso como um todo. Procure realizar com afinco todas as atividades sugeridas.

Desejo, sinceramente, que este material venha lhe agregar valores e que lhe ajude a construir conhecimentos sobre os assuntos estudados.

Prof. Rafael Vargas Mesquita dos Santos



Sumário

Cap. 1 - VISÃO GERAL 11

- 1.1 ORGANIZAÇÃO DOS CAPÍTULOS 11
- 1.2 MODELAGEM DE SISTEMAS DE SOFTWARE 12
- 1.3 O PARADIGMA DA ORIENTAÇÃO A OBJETOS 13
 - 1.3.1 Classes e Objetos 14
 - 1.3.2 Mensagens 15
 - 1.3.3 O Papel da Abstração na Orientação a Objetos 16
- 1.4 A LINGUAGEM DE MODELAGEM UNIFICADA (UML) 19

Cap. 2 - O PROCESSO DE DESENVOLVIMENTO DE SOFTWARE 23

- 2.1 ATIVIDADES TÍPICAS DE UM PROCESSO DE DESENVOLVIMENTO 24
 - 2.1.1 Levantamento de Requisitos 24
 - 2.1.2 Análise 26
 - 2.1.3 Projeto (Desenho) 27
 - 2.1.4 Implementação 27
 - 2.1.5 Testes 27
 - 2.1.6 Implantação 27
- 2.2 O COMPONENTE HUMANO (PARTICIPANTE DO PROCESSO) 28
 - 2.2.1 Gerentes de Projeto 28
 - 2.2.2 Analistas 29
 - 2.2.3 Projetistas 29
 - 2.2.4 Arquitetos de Software 29
 - 2.2.5 Programadores 29
 - 2.2.6 Especialistas do Domínio 30
 - 2.2.7 Avaliadores da Qualidade 30
- 2.3 MODELOS DE CICLO DE VIDA 30
 - 2.3.1 O Modelo de Ciclo de Vida em Cascata 30
 - 2.3.2 O Modelo de Ciclo de Vida Iterativo e Incremental 31

Cap. 3 - MODELAGEM DE CASOS DE USO 35

- 3.1 MODELO DE CASOS DE USO 35
 - 3.1.1 Casos de Uso 35
 - 3.1.2 Atores 37
 - 3.1.3 Relacionamentos 39
- 3.2 DIAGRAMA DE CASOS DE USO 42



3.3 DOCUMENTAÇÃO DE CASOS DE USO	46
3.4 MODELO DE CASOS DE USO NO PROCESSO DE DESENVOLVIMENTO	48
3.5 ESTUDO DE CASO	49
3.5.1 Descrição da Situação	49
3.5.2 Regras do Negócio	50
3.5.3 Documentação do MCU	50

Cap. 4 - MODELAGEM DE CLASSES DE ANÁLISE 55

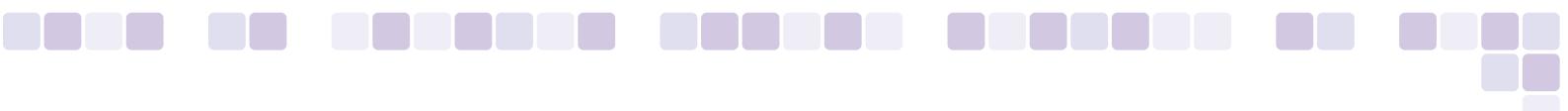
4.1 DIAGRAMA DE CLASSES	56
4.1.1 Classes	56
4.1.2 Associações	57
4.1.3 Generalizações e Especializações	64
4.2 TÉCNICAS PARA IDENTIFICAÇÃO DE CLASSES	66
4.2.1 Utilização de uma Taxonomia de Conceitos	66
4.2.2 Categorização BCE	67
4.3 MODELO DE CLASSES NO PROCESSO DE DESENVOLVIMENTO	69
4.4 ESTUDO DE CASO	69

Cap. 5 - MODELAGEM DE INTERAÇÕES 73

5.1 ELEMENTOS DA MODELAGEM DE INTERAÇÕES	74
5.2 DIAGRAMA DE SEQUÊNCIA	75
5.2.1 Linhas de Vida	76
5.2.2 Mensagens	76
5.2.3 Ocorrências de Execução	77
5.2.4 Criação e Destruição de Objetos	78
5.2.5 Modularização de Interações	79
5.3 PROCEDIMENTO DE CONSTRUÇÃO DE UM DIAGRAMA DE INTERAÇÃO	84
5.4 MODELO DE INTERAÇÕES NO PROCESSO DE DESENVOLVIMENTO	86
5.5 ESTUDO DE CASO	87

Cap. 6 - MODELAGEM DE CLASSES DE PROJETO 91

6.1 TRANSFORMAÇÃO DE CLASSES DE ANÁLISE EM CLASSES DE PROJETO	91
6.1.1 Especificação de Classes de Fronteira	91
6.1.2 Especificação de Classes de Controle	92
6.1.3 Especificações de Classes de Entidade	92
6.2 ESPECIFICAÇÃO DE ATRIBUTOS	93
6.2.1 Notação da UML para Atributos	93



6.3 ESPECIFICAÇÃO DE OPERAÇÕES 96

6.3.1 Notação da UML para Operações 96

6.4 ESPECIFICAÇÃO DE ASSOCIAÇÕES 97

6.4.1 Navegabilidade de Associações 98

6.5 MODELO DE CLASSES DE PROJETO NO PROCESSO DE DESENVOLVIMENTO 99

6.6 ESTUDO DE CASO 100

REFERÊNCIAS 103



VISÃO GERAL

Introduziremos alguns conceitos iniciais importantes para o bom entendimento dos assuntos posteriores.

*Angue risus at
e velit at tellus.
massa portitor
sectetur magna.*

Fala Professor

1.1 ORGANIZAÇÃO DOS CAPÍTULOS

O Capítulo 1 apresenta uma breve introdução à utilização do paradigma da orientação a objetos e da UML, tendo como objetivo fornecer uma visão geral sobre a análise e o projeto de sistemas de software sob o ponto de vista de orientação a objetos. Os principais conceitos do paradigma da orientação a objetos são introduzidos neste capítulo.

O Capítulo 2 descreve as principais atividades constituintes de um processo de desenvolvimento de software. Também descrevemos os principais profissionais envolvidos nesse processo, juntamente com suas respectivas atribuições. O processo de desenvolvimento em cascata é apresentado com o objetivo de dar uma motivação para o surgimento do processo incremental e evolutivo. Em seguida, este último é também descrito e apresentado como a forma atual de se desenvolver sistemas orientados a objetos. Na maioria dos demais capítulos, são feitas alusões à utilização da UML em um processo de desenvolvimento incremental e evolutivo.

No Capítulo 3, apresentamos o modelo de casos de uso e os diversos elementos do diagrama de casos de uso da UML. Além disso, são fornecidas diversas dicas práticas que podem ser utilizadas na construção desse modelo. Este capítulo também enfatiza o modelo de casos de uso como um ponto central de um processo de desenvolvimento que utilize a UML como linguagem de modelagem.

O Capítulo 4 descreve a construção do modelo de classes de análise de um sistema de software orientado a objetos (SSOO). Os principais elementos de notação definidos pela UML para a construção do diagrama de classes são descritos, bem como diversas técnicas úteis na identificação das classes iniciais de um SSOO.

A modelagem de interações entre objetos em um SSOO é discutida no Capítulo 5. Nesse capítulo, apresento a ideia de que as construções do modelo de classe e do modelo de interações são interdependentes: a

construção de um modelo fornece informações para a construção do outro, e vice-versa. Os diagramas de interação foram os mais atingidos (em termos de mudanças) com a nova versão da Linguagem de Modelagem Unificada, a UML 2.0. Nesta segunda edição, também apresentamos alguns novos elementos de notação introduzidos pela UML 2.0.

O Capítulo 6 retoma a discussão sobre o modelo de classes, agora com um enfoque nas características de modelagem referentes à fase de projeto.

1.2 MODELAGEM DE SISTEMAS DE SOFTWARE

De acordo com Bezerra (2007) historicamente, diversos tipos de bens serviram de base para o desenvolvimento da economia, dentre os quais propriedade, mão de obra, máquinas e capital. Atualmente, está surgindo um novo tipo de bem econômico: a informação. Nos dias de hoje, a empresa que dispõe de mais informações sobre seu processo de negócio está em vantagem em relação a suas competidoras.

Há um ditado segundo o qual "a necessidade é a mãe das invenções". Em consequência do crescimento da importância da informação, surgiu a necessidade de gerenciar informações de uma forma adequada e eficiente e, dessa necessidade, surgiram os denominados sistemas de informações.

Conceitos



Um *sistema de informações* é uma combinação de pessoas, dados, processos, interfaces, redes de comunicação e tecnologia que interagem com o objetivo de dar suporte e melhorar o processo de negócio de uma organização empresarial com relação às informações que nela fluem. Considerando o caráter estratégico da informação nos dias de hoje, pode-se dizer também que os sistemas de informações têm o objetivo de prover vantagens para uma organização do ponto de vista competitivo.

Para o mesmo autor uma característica intrínseca de sistemas de software é a complexidade de seu desenvolvimento, que aumenta à medida que cresce o tamanho do sistema. Para se ter uma ideia da complexidade envolvida na construção de alguns sistemas, pense no tempo e nos recursos materiais necessários para se construir uma casa de cachorro. Para construir essa casa, provavelmente tudo de que se precisa é de algumas ripas de madeira, alguns pregos, uma caixa de ferramentas e certa dose de amor por seu cachorro. Depois de alguns dias, a casa estaria pronta. O que dizer da construção de uma casa para sua família? Decerto, tal empreitada não seria realizada tão facilmente. O tempo e os

recursos necessários seriam uma ou duas ordens de grandeza maiores do que o necessário para a construção da casa de cachorro. O que dizer, então, da construção de um edifício? Certamente, para se construir habitações mais complexas (casas e edifícios), algum planejamento adicional é necessário. Engenheiros e arquitetos constroem plantas dos diversos elementos da habitação antes do início da construção propriamente dita. Na terminologia da construção civil, plantas hidráulicas, elétricas, de fundação, etc., são projetadas e devem manter consistência entre si. Provavelmente, uma equipe de profissionais estaria envolvida na construção, e aos membros dessa equipe seriam delegadas diversas tarefas, no tempo adequado para cada uma delas.

Na construção de sistemas de software, assim como na construção de sistemas habitacionais, também há uma graduação de complexidade. Para a construção de sistemas de software mais complexos, também é necessário um planejamento inicial. O equivalente ao projeto das plantas da engenharia civil também deve ser realizado. Essa necessidade leva ao conceito de modelo, tão importante no desenvolvimento de sistemas. De uma perspectiva mais ampla, um modelo pode ser visto como uma representação idealizada de um sistema a ser construído. Maquetes de edifícios e de aviões e plantas de circuitos eletrônicos são apenas alguns exemplos de modelos (Bezerra, 2007).

1.3 O PARADIGMA DA ORIENTAÇÃO A OBJETOS

Indispensável ao desenvolvimento atual de sistemas de software é o paradigma da orientação a objetos. Esta seção descreve o que esse termo significa e justifica por que a orientação a objetos é importante para a modelagem de sistemas. Pode-se começar pela definição da palavra paradigma.

Uma definição de **paradigma** apropriada ao contexto deste material: um paradigma é uma forma de abordar um problema.



Conceitos

Kay pensou em como construir um sistema de software a partir de agentes autônomos que interagem entre si. Ele, então, estabeleceu os seguintes princípios da orientação a objetos:

1. Qualquer coisa é um objeto.
2. Objetos realizam tarefas por meio da requisição de serviços a outros objetos.

3. Cada objeto pertence a uma determinada classe. Uma classe agrupa objetos similares.
4. A classe é um repositório para comportamento associado ao objeto.
5. Classes são organizadas em hierarquias.

Falbo (2005) menciona que a construção de uma solução computadorizada consiste no mapeamento do problema a ser resolvido no mundo real num modelo de solução no Espaço de Soluções, isto é, o meio computacional. A modelagem envolve, então, a identificação de objetos e operações relevantes no mundo real e o mapeamento desses em representações abstratas no Espaço de Soluções.

À distância existente entre o problema no mundo real e o modelo abstrato construído, convencionou-se chamar gap semântico e, obviamente, quanto menor ele for, mais direto será o mapeamento e, portanto, mais rapidamente serão construídas soluções para o problema. Sob essa ótica, é fácil perceber que o gap semântico representa a área de atuação da Engenharia de Software. Diversas técnicas e métodos têm sido propostos para as diferentes fases do processo de desenvolvimento, buscando minimizá-lo. A Orientação a Objetos é um dos paradigmas existentes para apoiar o desenvolvimento de sistemas, que busca fornecer meios para se diminuir o gap semântico (Falbo, 2005).

1.3.1. Classes e Objetos

Ainda de acordo com Falbo (2005) o mundo real é povoado por elementos que interagem entre si, onde cada um deles desempenha um papel específico. A esses elementos, chamamos objetos. Objetos podem ser coisas concretas ou abstratas, tais como um carro, uma reserva de passagem aérea, uma organização, uma planta de engenharia, um componente de uma planta de engenharia, etc.

Do ponto de vista da modelagem de sistemas, um objeto é uma entidade que incorpora uma abstração relevante no contexto de uma aplicação.

Bezerra, 2007, diz que os seres humanos costumam agrupar os objetos. Provavelmente, os seres humanos realizam esse processo mental de agrupamento para tentar gerenciar a complexidade de entender as coisas do mundo real. Realmente, é bem mais fácil entender a ideia cavalo do que entender todos os cavalos que existem. Na terminologia da orientação a objetos, cada ideia é denominada classe de objetos, ou simplesmente classe. Uma classe é uma descrição dos atributos e serviços comuns a um grupo de objetos. Sendo assim, pode-se entender uma

classe como sendo um molde a partir do qual objetos são construídos. Ainda sobre terminologia, diz-se que um objeto é uma instância de uma classe.

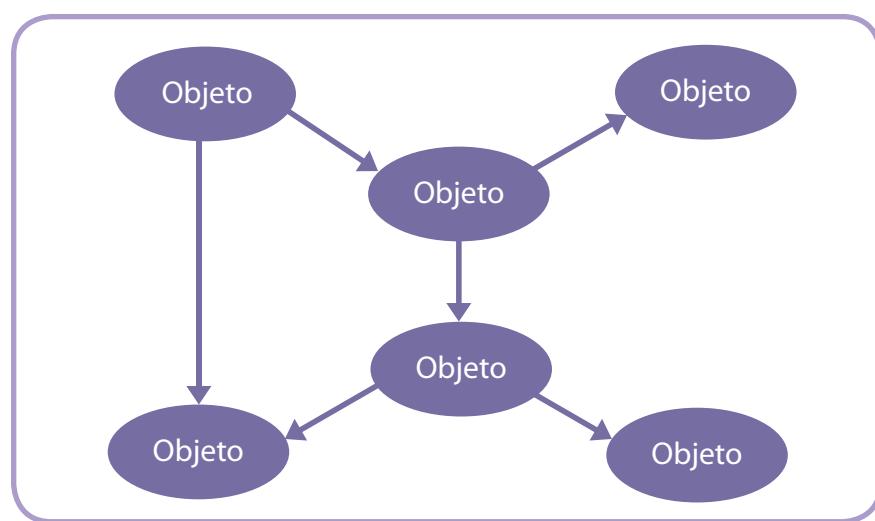
Por exemplo: quando se pensa em um cavalo, logo vem à mente um animal de quatro patas, cauda, crina, etc. Pode ser que algum dia você veja dois cavalos, um mais baixo que o outro, um com cauda maior que o outro, ou mesmo, por um infeliz acaso, um cavalo com menos patas que o outro. No entanto, você ainda terá certeza de estar diante de dois cavalos. Isso porque a ideia (classe) cavalo está formada na mente dos seres humanos, independentemente das pequenas diferenças que possam haver entre os exemplares (objetos) da ideia cavalo.

1.3.2 Mensagens

A abstração incorporada por um objeto é caracterizada por um conjunto de operações que podem ser requisitadas por outros objetos, ditos clientes. Métodos são implementações reais de operações. Para que um objeto realize alguma tarefa, é necessário enviar a ele uma mensagem, solicitando a execução de um método específico. Um cliente só pode acessar um objeto através da emissão de mensagens, isto é, ele não pode acessar ou manipular diretamente os dados associados ao objeto. Os objetos podem ser complexos e o cliente não precisa tomar conhecimento de sua complexidade interna. O cliente precisa saber apenas como se comunicar com o objeto e como ele reage (Falbo, 2005).

As mensagens são o meio de comunicação entre objetos e são responsáveis pela ativação de todo e qualquer processamento. Dessa forma, é possível garantir que clientes não serão afetados por alterações nas implementações de um objeto que não alterem o comportamento esperado de seus serviços.

Figura 1-1: Objetos interagem através do envio de mensagens (Bezerra, 2007).



1.3.3 O Papel da Abstração na Orientação a Objetos

Nesta seção, apresentamos os principais conceitos do paradigma da orientação a objetos. Discutimos também o argumento de que todos esses conceitos são, na verdade, a aplicação de um único conceito mais básico: o princípio da abstração.

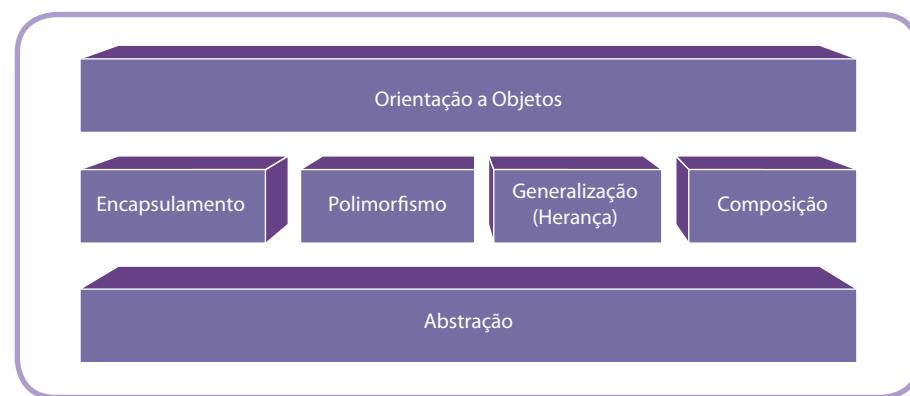
Conceitos



Abstração consiste de focalizar nos aspectos essenciais inerentes a uma entidade e ignorar propriedades “acidentais”. Em termos de desenvolvimento de sistemas, isto significa concentrar-se no que um objeto é e faz antes de se decidir como ele será implementado.

Esse processo mental nos permite gerenciar a complexidade de um objeto, ao mesmo tempo em que concentramos nossa atenção nas características essenciais do mesmo. Note que uma abstração de algo é dependente da perspectiva (contexto) sobre a qual uma coisa é analisada: o que é importante em um contexto pode não ser importante em outro. Nas próximas seções, descrevemos alguns conceitos fundamentais da orientação a objetos e estabelecemos sua correlação com o conceito de abstração.

Figura 1-2: Princípios da orientação a objetos podem ser vistos como aplicações de um princípio mais básico, o da abstração (Bezerra, 2007).



Encapsulamento

Objetos possuem comportamento. O termo comportamento diz respeito a operações realizadas por um objeto, conforme este objeto receba mensagens.

Conceitos



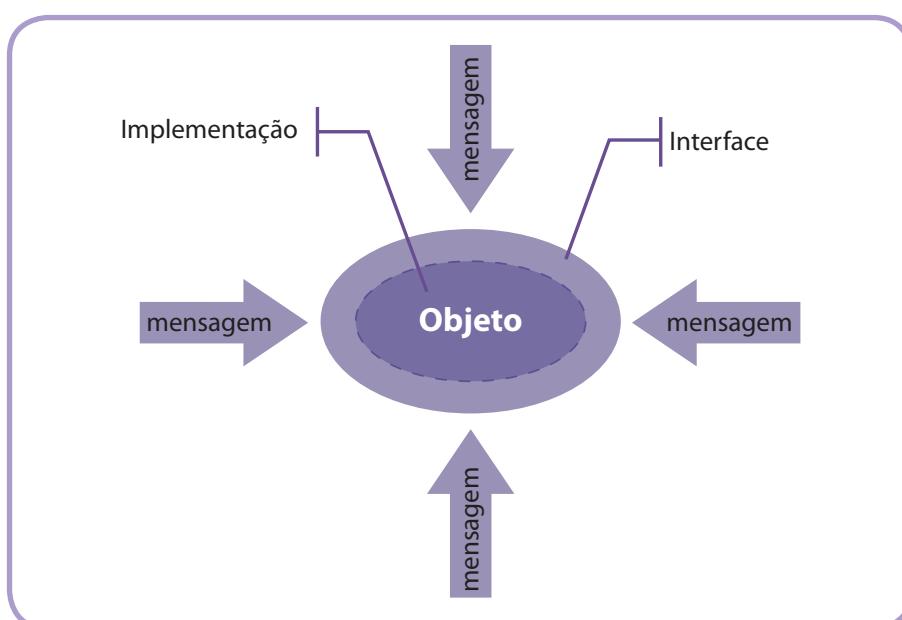
O mecanismo de encapsulamento é uma forma de restringir o acesso ao comportamento interno de um objeto.

Falbo, 2005, afirma que no mundo real, um objeto pode interagir com outro sem conhecer seu funcionamento interno. Uma pessoa, por exemplo, geralmente utiliza uma televisão sem saber efetivamente qual a sua estrutura interna ou como seus mecanismos internos são ativados. Para utilizá-la, basta saber realizar algumas operações básicas, tais como ligar/desligar a TV, mudar de um canal para outro, regular volume, cor, etc. Como estas operações produzem seus resultados, mostrando um programa na tela, não interessa ao telespectador.

O encapsulamento consiste na separação dos aspectos externos de um objeto, acessíveis por outros objetos, de seus detalhes internos de implementação, que ficam ocultos dos demais objetos. A interface de comunicação de um objeto deve ser definida de forma a revelar o menos possível sobre o seu funcionamento interno (Falbo, 2005).

Através do encapsulamento, a única coisa que um objeto precisa saber para pedir a colaboração de outro objeto é conhecer a sua interface. Nada mais. Isso contribui para a autonomia dos objetos, pois cada objeto envia mensagens a outros objetos para realizar certas operações, sem se preocupar em como se realizaram as operações.

Figura 1-3: Princípio do encapsulamento: visto externamente, o objeto e sua interface (Bezerra, 2007).



Polimorfismo

O **polimorfismo** significa a habilidade de tomar várias formas (Falbo, 2005).



Conceitos

O polimorfismo é uma poderosa ferramenta para o desenvolvimento de sistemas flexíveis. Polimorfismo significa a habilidade de tomar várias formas. No contexto da orientação a objetos, o polimorfismo está intrinsecamente ligado à comunicação entre objetos. De fato, polimorfismo pode ser melhor caracterizado, neste contexto, como o fato de um objeto emissor de uma mensagem não precisar conhecer a classe do objeto receptor. Assim, uma mensagem pode ser interpretada de diferentes maneiras, dependendo da classe do objeto receptor, ou seja, é o objeto que receptor que determina a interpretação da mensagem, e não o objeto emissor. O emissor precisa saber apenas que o receptor pode realizar certo comportamento, mas não a que classe ele pertence e, portanto, que operação é efetivamente executada. Um objeto sabe qual é a sua classe, e, portanto, a correta implementação da operação requisitada. A mensagem é associada ao método a ser realmente executado, através da identificação da operação e da classe do objeto receptor (Falbo, 2005).

Generalização (Herança)

Muitas vezes, um conceito geral pode ser especializado, adicionando-se novas características. Tomemos, como exemplo, o conceito que temos de estudantes. De modo geral, há características que são intrínsecas a quaisquer estudantes. Entretanto, é possível especializar este conceito para mostrar especificidades de subtipos de estudantes, tais como estudantes de 1º grau, estudantes de 2º grau, estudantes de graduação e estudantes de pós-graduação, entre outros.

Da maneira inversa, se pode extrair de um conjunto de conceitos, características comuns que, quando generalizadas, formam um conceito geral. Por exemplo, ao avaliarmos os conceitos que temos de carros, motos, caminhões e ônibus, podemos notar que esses têm características comuns que podem ser generalizadas em um supertipo veículos automotores terrestres.

As abstrações de especialização e generalização são muito úteis para a estruturação de sistemas. Com elas, é possível construir hierarquias de classes, subclasses, subsubclasses, e assim por diante.

Conceitos

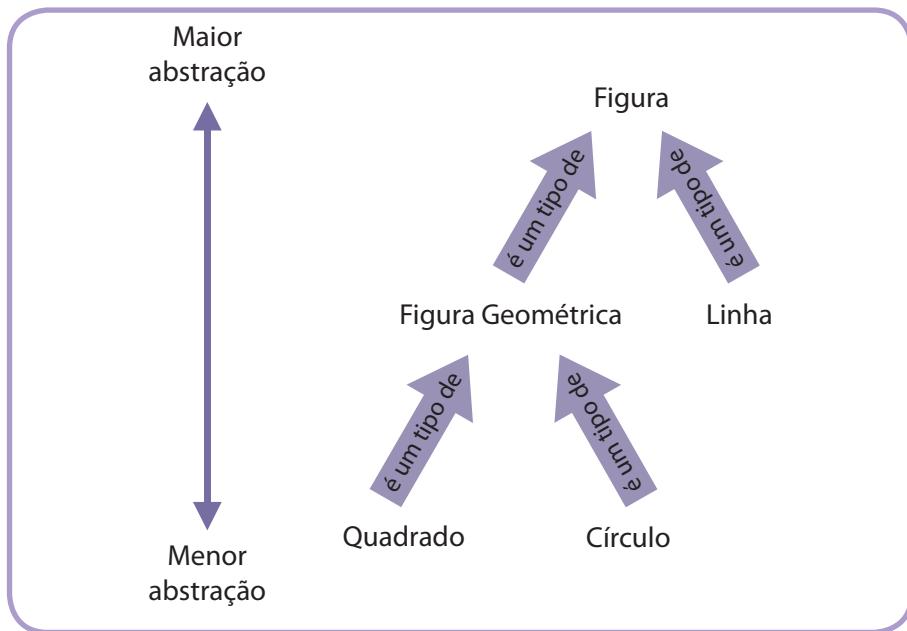


A **herança** é um mecanismo para modelar similaridades entre classes, representando as abstrações de generalização e especialização (Falbo, 2005).

Uma das principais vantagens da herança é facilitar a modificação de modelos. A herança nos permite conceber uma nova classe como um

refinamento de outras classes. A nova classe pode herdar as similaridades e definir apenas a funcionalidade nova. O desenvolvimento orientado a objetos é fortemente baseado na identificação de objetos e na construção de hierarquias de classes, utilizando o mecanismo de herança.

Figura 1-4: Princípio da generalização: classes podem ser organizadas em hierarquias (Bezerra, 2007).



Composição

Uma forma especial de relacionamento entre objetos é a composição. Parte do poder dos softwares orientados a objetos advém de sua habilidade de manipular objetos complexos, como sendo compostos de vários outros objetos mais simples. Um carro, por exemplo, é composto por motor, rodas, carroceria, etc. Um motor, por sua vez, é composto de bloco, válvulas, pistões, e assim por diante (Falbo, 2005).

Composição permite que criemos objetos a partir da reunião de outros objetos.



Conceitos

1.4 A LINGUAGEM DE MODELAGEM UNIFICADA (UML)

A Linguagem de Modelagem Unificada (*Unified Modeling Language - UML*) é a linguagem padrão para especificar, visualizar, documentar e construir artefatos de um sistema, podendo ser utilizada em todos

os processos de desenvolvimento orientados a objetos, ao longo de seus ciclos de desenvolvimento, usando diferentes tecnologias de implementação (Falbo, 2005).

A UML teve origem em uma tentativa de se unificar os principais métodos orientados a objetos utilizados até então: a OMT e o Método de Booch. A este esforço juntou-se também Ivar Jacobson, fundindo também seu método OOSE. Contudo, percebeu-se que não era possível estabelecer um único método adequado para todo e qualquer desenvolvimento. De fato, um método é composto por uma notação para os artefatos produzidos e de um processo descrevendo que artefatos construir e como construi-los. A notação pode ser unificada, mas a decisão de quais artefatos produzir e que passos seguir não é passível de padronização, já que varia de projeto para projeto. Assim, ao invés de criarem um método unificado, Rumbaugh, Booch e Jacobson propuseram a UML, incorporando as principais notações para os produtos de seus métodos e de vários outros, com a colaboração de várias empresas e autores. A UML foi aprovada em novembro de 1997 pela OMG - *Object Management Group* - pondo fim a uma guerra de métodos OO (Falbo, 2005).

Os diagramas contemplados pela UML e estudados nesta disciplina são:

Diagrama de Casos de Uso: Os casos de uso descrevem a funcionalidade do sistema percebida por atores externos. Um ator interage com o sistema, podendo ser um usuário, dispositivo ou outro sistema.

Diagrama de Classes: Denota a estrutura estática de um sistema, isto é, as classes, os relacionamentos entre suas instâncias (objetos), restrições e hierarquias. O diagrama é considerado estático, pois a estrutura descrita é sempre válida em qualquer ponto no ciclo de vida do sistema.

Diagramas de Interação, podendo ser:

Diagramas de Sequência: mostram a colaboração dinâmica entre um número de objetos, sendo seu objetivo principal mostrar a sequência de mensagens enviadas entre objetos. É um gráfico bidimensional, onde a dimensão vertical representa o tempo e a dimensão horizontal os diferentes objetos.

Diagramas de Colaboração: têm exatamente o mesmo propósito dos diagramas de sequência, apresentando, contudo, um formato diferente. São desenhados como diagramas de objetos, onde são mostradas as mensagens trocadas entre os objetos. Este não será visto no material.

Vale ressaltar mais uma vez, que a UML é uma linguagem de modelagem, não um método de desenvolvimento OO. Os métodos

consistem, pelo menos em princípio, de uma linguagem de modelagem e um procedimento de uso dessa linguagem. A UML não prescreve explicitamente esse procedimento de utilização. Assim, a UML deve ser aplicada no contexto de um processo, lembrando que domínios de problemas e projetos diferentes podem requerer processos diferentes.

A definição completa da UML está contida na Especificação da Linguagem de Modelagem Unificada da OMG. Essa especificação pode ser obtida gratuitamente no site da OMG (www.uml.org). Embora essa documentação seja bastante completa, ela está longe de fornecer uma leitura fácil, pois é direcionada a pesquisadores ou a desenvolvedores de ferramentas de suporte (ferramentas CASE) ao desenvolvimento de sistemas.

Indicações



1. Explique e relate os termos objeto, classe, generalização e mensagem. Dê exemplos de cada um desses conceitos.

Resposta:

Objeto: um objeto é qualquer coisa que possamos ver, sentir, tocar ou idealizar. Exemplos de objetos são: eu, esse texto, sua conta de e-mail, etc. Computacionalmente falando, um objeto é uma abstração de algo do mundo real. Em vista disso, um objeto é bem mais simples que a sua contrapartida original e representa as características e comportamentos relevantes desse original.

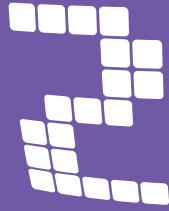
Classes: corresponde a um agrupamento de objetos. Ela retém os comportamentos que serão característicos aos objetos que dela fazem parte. Por exemplo: Pessoa é uma classe na qual todos nós estamos contidos; porém, quando pensamos em alguém especificamente, estamos pensando no objeto; quando idealizamos uma imagem de uma pessoa, assim o fizemos usando as características da classe. Por exemplo: Camisa é uma classe; a camisa que estou usando agora é um objeto dessa classe.

Herença: é a capacidade que uma classe tem de herdar características da classe que está acima dela hierarquicamente. Por exemplo: Camisa é uma subclasse da classe Roupa, logo ela herda comportamentos e funções. Todas as duas têm a comportamento de servir como vestimentas para quem a usa.

Mensagem: é um pedido ou informação passado de um objeto para outro. Por exemplo: quando giramos o objeto chave na ignição do carro, ele passa a mensagem para o carro de que queremos que ele funcione; em resposta, o carro executa essa tarefa.

Atividades





O PROCESSO DE DESENVOLVIMENTO DE SOFTWARE

Neste capítulo, focamos na importância de um processo de desenvolvimento de software, bem como na apresentação de suas principais atividades.

Angue visus ait.
et velit at tellus.
massa portitor
sectetur magna.

Fala Professor

Bezerra (2007) afirma que o desenvolvimento de software é uma atividade complexa. Essa complexidade corresponde à sobreposição das complexidades relativas ao desenvolvimento dos seus diversos componentes: software, hardware, procedimentos, etc. Isso se reflete no alto número de projetos de software e não chegam ao fim, ou que extrapolam recursos de tempo e de dinheiro alocados.

Para dar uma ideia da complexidade no desenvolvimento de sistemas de software, são listados, a seguir, alguns dados levantados no *Chaos Report*, um estudo clássico feito pelo *Standish Group* sobre projetos de desenvolvimento Chaos (1994):

- Porcentagem de projetos que terminam dentro do prazo estimado: 10%.
- Porcentagem de projetos que são descontinuados antes de chegarem ao fim: 25%.
- Porcentagem de projetos acima do custo esperado: 60%.
- Atraso médio nos projetos: um ano.

Tentativas de lidar com essa complexidade e de minimizar os problemas envolvidos no desenvolvimento de software envolvem a definição de processos de desenvolvimento de software.

Segundo Falbo (2005), “Um processo de software pode ser visto como o conjunto de atividades, métodos, práticas e transformações que guiam pessoas na produção de software. Um processo eficaz deve, claramente, considerar as relações entre as atividades, os artefatos produzidos no desenvolvimento, as ferramentas e os procedimentos necessários e a habilidade, o treinamento e a motivação do pessoal envolvido”.

Exemplos de processos de desenvolvimento propostos são o ICONIX, o RUP (*Rational Unified Process*), o EUP (*Enterprise Unified Process*), XP (*Extreme Programming*) e o OPEN (*Object-Oriented Process, Environment and Notation*).

2.1 ATIVIDADES TÍPICAS DE UM PROCESSO DE DESENVOLVIMENTO

Um processo de software não pode ser definido de uma maneira universal. Para ser eficaz e conduzir à construção de produtos de boa qualidade, cada processo de software deve ser adequado às especificidades do projeto em questão, tais como as características da aplicação (domínio do problema, tamanho, complexidade, etc), a tecnologia a ser adotada na sua construção (paradigma de desenvolvimento, linguagem de programação, mecanismo de persistência, etc), a organização em que o produto será desenvolvido e a equipe de desenvolvimento, etc.

Cada processo tem suas particularidades em relação ao modo de arranjar e encadear as atividades de desenvolvimento. Entretanto, podem-se distinguir atividades que, com uma ou outra modificação, são comuns à maioria dos processos existentes. Nesta seção, descrevemos essas atividades, posto que duas dessas atividades, a análise e o projeto, fazem parte do assunto principal desta disciplina.

2.1.1 Levantamento de Requisitos

Conceitos



Requisitos são uma descrição das necessidades ou dos desejos para um produto (Larman, 2007).

O objetivo básico da atividade de *levantamento de requisitos* é identificar e documentar o que é realmente necessário, em uma forma que comunica claramente essa informação ao cliente e aos membros da equipe de desenvolvimento (Larman, 2007).

Nessa etapa os desenvolvedores, juntamente com os clientes, tentam levantar e definir as necessidades dos futuros usuários do sistema a ser desenvolvido. Essas necessidades são geralmente denominadas requisitos (Bezerra, 2007).

Normalmente, os requisitos de um sistema são identificados a partir de um domínio. Denomina-se domínio a área de conhecimento ou de atividade específica caracterizada por um conjunto de conceitos e de terminologia compreendidos por especialista nessa área. No contexto do desenvolvimento de software, um domínio corresponde à parte do mundo real que é relevante, no sentido de que algumas informações e processos desse domínio precisam ser incluídos no sistema em desenvolvimento. O domínio também é chamado de *domínio do problema ou domínio do negócio*.

Durante o levantamento de requisitos, a equipe de desenvolvimento tenta entender o domínio que deve ser automatizado pelo sistema de software. O levantamento de requisitos compreende, também, um estudo exploratório das necessidades dos usuários e da situação do sistema atual (se este existir). Há várias técnicas utilizadas para isso, como, por exemplo: leitura de obras de referência e livros-texto, observação do ambiente do usuário, realização de entrevistas com os usuários, entrevistas com *especialistas do domínio*. (ver a Seção 2.2.6), reutilização de análises anteriores e comparação com sistemas preexistentes do mesmo domínio do negócio.

O produto do levantamento de requisitos é o *documento de requisitos*, que declara os diversos tipos de requisitos do sistema. É normal esse documento ser escrito em uma notação informal (em linguagem natural). Para Bezerra (2007) as principais seções de um documento de requisitos são:

1. *Requisitos funcionais*: definem as funcionalidades do sistema. Alguns exemplos de requisitos funcionais são os seguintes.

- a) "O sistema deve permitir que cada professor realize o lançamento de notas das turmas nas quais lecionou."
- b) "O sistema deve permitir que um aluno realize a sua matrícula nas disciplinas oferecidas em um semestre letivo."
- c) "Os coordenadores de escola devem poder obter o número de aprovações, reprovações e trancamentos em cada disciplina oferecida em um determinado período."

2. *Requisitos não funcionais*: declaram as características de qualidade que o sistema deve possuir e que estão relacionadas às suas funcionalidades.

Alguns tipos de requisitos não funcionais são os seguintes:

- a) Confiabilidade: corresponde a medidas quantitativas da confiabilidade do sistema, tais como tempo médio entre falhas, recuperação de falhas ou quantidade de erros por milhares de linhas de código-fonte.
- b) Desempenho: requisitos que definem tempos de resposta esperados para as funcionalidades do sistema.
- c) Portabilidade: restrições sobre as plataformas de hardware e de software nas quais o sistema será implantado e sobre o grau de facilidade para transportar o sistema para outras plataformas.

d) Segurança: limitações sobre a segurança do sistema em relação a acessos não autorizados.

e) Usabilidade: requisitos que se relacionam ou afetam a usabilidade do sistema. Exemplos incluem requisitos sobre a facilidade de uso e a necessidade ou não de treinamento dos usuários.

O levantamento de requisitos é a etapa mais importante em termos de retorno em investimentos feitos para o projeto de desenvolvimento. Muitos sistemas foram abandonados ou nem chegaram a uso porque os membros da equipe não dispensaram tempo suficiente para compreender as necessidades do cliente em relação ao novo sistema. De fato, um sistema de informações é normalmente utilizado para automatizar processos de negócio de uma organização. Portanto, esses processos devem ser compreendidos antes da construção do sistema de informações. Em um estudo baseado em 6.700 sistemas feito em 1997, Carper Jones mostrou que os custos resultantes da má realização dessa etapa de entendimento podem ser duzentas vezes maiores que o realmente necessário (Jones, 1997).

Para Bezerra (2007) o documento de requisitos serve como um termo de consenso entre a equipe técnica (desenvolvedores) e o cliente. Esse documento constitui a base para as atividades subsequentes do desenvolvimento do sistema e fornece um ponto de referência para qualquer validação futura do software construído. O envolvimento do cliente desde o início do processo de desenvolvimento ajuda a assegurar que o produto desenvolvido realmente atenda às necessidades identificadas. Além disso, o documento de requisitos estabelece o *escopo do sistema* (isto é, o que faz parte e o que não faz parte do sistema).

2.1.2. Análise

A análise enfatiza a *investigação do problema*, cujo objetivo é levar o analista a investigar e a descobrir. Para que essa etapa seja realizada em menos tempo e com maior precisão, deve-se ter um bom método de trabalho.

Essa etapa é importantíssima porque ninguém é capaz de entender com perfeição um problema usual de sistemas de informação na primeira vez que o olha. Assim, esse tempo de análise deve ser bem aproveitado na investigação do problema.

Pode-se dizer que o resultado da análise é o enunciado do problema, e o projeto será a sua resolução. Problemas mal enunciados podem até ser resolvidos, mas a solução não corresponderá às expectativas.

Uma frase fundamental na fase de análise, segundo Larman (2007), é:
“Faça a coisa certa!”.

2.1.3. Projeto (Desenho)

A fase de projeto enfatiza a *proposta de uma solução* que atenda aos requisitos da análise. Então, se a análise é uma investigação para tentar descobrir o que o cliente quer, o projeto consiste em propor uma solução com base no conhecimento adquirido na análise.

De forma complementar, não adianta nada compreender perfeitamente os requisitos do sistema e ser incapaz de construir um software adequado a esses requisitos.

A frase fundamental da fase de projeto, segundo Larman (2007), é:
“Faça certo a coisa!”.

2.1.4. Implementação

Nesta fase, o projeto deve ser traduzido para uma forma passível de execução pela máquina. A atividade de implementação realiza essa tarefa, isto é, cada unidade de software do projeto detalhado é implementada.

Em um processo de desenvolvimento orientado a objetos, a implementação envolve a criação do código-fonte correspondente às classes de objetos do sistema utilizando linguagens de programação como C#, C++, Java, etc. Além da codificação desde o início, a implementação pode também reutilizar componentes de software, bibliotecas de classes e frameworks para agilizar a atividade.

2.1.5. Testes

Esta atividade inclui diversos níveis de testes, a saber, teste de unidade, teste de integração e teste de sistema. Inicialmente, cada unidade de software implementada deve ser testada e os resultados documentados. A seguir, os diversos componentes devem ser integrados sucessivamente, até se obter o sistema. Finalmente, o sistema como um todo deve ser testado.

2.1.6. Implantação

Uma vez testado, o software deve ser colocado em produção. Para que isso aconteça, contudo, é necessário treinar os usuários, configurar o ambiente de produção e, muitas vezes, converter bases de dados. O propósito desta atividade é estabelecer que o software satisfaça os

requisitos dos usuários. Isso é feito instalando o software no ambiente do usuário e conduzindo testes de aceitação. Quando o software tiver demonstrado prover as capacidades requeridas, ele pode ser aceito.

2.2 O COMPONENTE HUMANO (PARTICIPANTE DO PROCESSO)

O desenvolvimento de software é uma tarefa altamente cooperativa. Tecnologias complexas demandam especialistas em áreas específicas. Uma equipe de desenvolvimento de sistemas de software pode envolver vários especialistas, como, por exemplo, profissionais de informática para fornecer o conhecimento técnico necessário ao desenvolvimento do sistema de software e especialistas do domínio para o qual o sistema de software deve ser desenvolvido.

Para Bezerra (2007) uma equipe de desenvolvimento de software típica consiste em um gerente, analistas, projetistas, programadores, clientes e grupos de avaliação de qualidade. Esses participantes do processo de desenvolvimento são descritos a seguir, de acordo com o mesmo autor.

Contudo, é importante notar que a descrição dos participantes do processo tem mais um fim didático. Na prática, a mesma pessoa desempenha diferentes funções e, por outro lado, uma mesma função é normalmente desempenhada por várias pessoas.

Atenção



Em virtude de seu tamanho e sua complexidade, o desenvolvimento de sistemas de software é um empreendimento realizado em equipe.

2.2.1 Gerentes de Projeto

Como o próprio nome diz, o gerente de projetos é o profissional responsável pela gerência ou coordenação das atividades necessárias à construção do sistema. Esse profissional também é responsável por fazer o orçamento do projeto de desenvolvimento, como, por exemplo, estimar o tempo necessário para o desenvolvimento do sistema, definir qual o processo de desenvolvimento, o cronograma de execução das atividades, a mão de obra especializada, os recursos de hardware e software, etc.

2.2.2 Analistas

O analista de sistemas é o profissional que deve ter conhecimento do *domínio do negócio*. Esse profissional deve entender os problemas do domínio do negócio para que possa definir os requisitos do sistema a ser desenvolvido. Analistas devem estar aptos a se comunicar com especialistas do domínio para obter conhecimento acerca dos problemas e das necessidades envolvidas na organização empresarial. O analista não precisa ser um especialista, contudo, ele deve ter suficiente domínio do vocabulário da área de conhecimento na qual o sistema será implantado para que, ao se comunicar com o especialista de domínio, este não precise ser interrompido a todo momento para explicar conceitos básicos da área.

2.2.3 Projetistas

O projetista de sistemas é o integrante da equipe de desenvolvimento cujas funções são (1) avaliar as alternativas de solução (da definição) do problema resultante da análise e (2) gerar a especificação de uma solução computacional detalhada. A tarefa do projetista de sistemas é muitas vezes chamada de *projeto físico*.

2.2.4 Arquitetos de Software

Um profissional encontrado principalmente em grandes equipes reunidas para desenvolver sistemas complexos é o arquiteto de software. O objetivo desse profissional é elaborar a arquitetura do sistema como um todo. É ele quem toma decisões sobre quais são os subsistemas que compõem o sistema como um todo e quais são as interfaces entre esses subsistemas.

2.2.5 Programadores

Esse profissional é o responsável pela *implementação do sistema*. É comum haver vários programadores em uma equipe de desenvolvimento. Um programador pode ser proficiente em uma ou mais linguagens de programação, além de ter conhecimento sobre bancos de dados e poder ler os modelos resultantes do trabalho do projetista.

Na verdade, a maioria das equipes de desenvolvimento possui analistas que realizam alguma programação, e programadores que realizam alguma análise.

2.2.6. Especialistas do Domínio

Outro componente da equipe de desenvolvimento é o *especialista do domínio*, também conhecido como *especialista do negócio*. Esse componente é o indivíduo, ou grupo de indivíduos, que possui conhecimento acerca da área ou do negócio em que o sistema em desenvolvimento estará inserido. Um termo mais amplo que especialista de domínio é cliente. Podem-se distinguir dois tipos de clientes: o *cliente usuário* e o *cliente contratante*. Cliente usuário é o indivíduo que efetivamente utilizará o sistema. Cliente usuário normalmente é um especialista do domínio. É com esse tipo de cliente que o analista de sistemas interage para levantar os requisitos do sistema. O cliente contratante é o indivíduo que solicita o desenvolvimento do sistema. Ou seja, é esse usuário quem encomenda e patrocina os custos de desenvolvimento e manutenção.

2.2.7 Avaliadores da Qualidade

O desempenho e a confiabilidade são exemplos de características que devem ser encontradas em um sistema de software de boa qualidade. Avaliadores de qualidade asseguram a adequação do processo de desenvolvimento e do produto de software sendo desenvolvido aos padrões de qualidade estabelecidos pela organização.

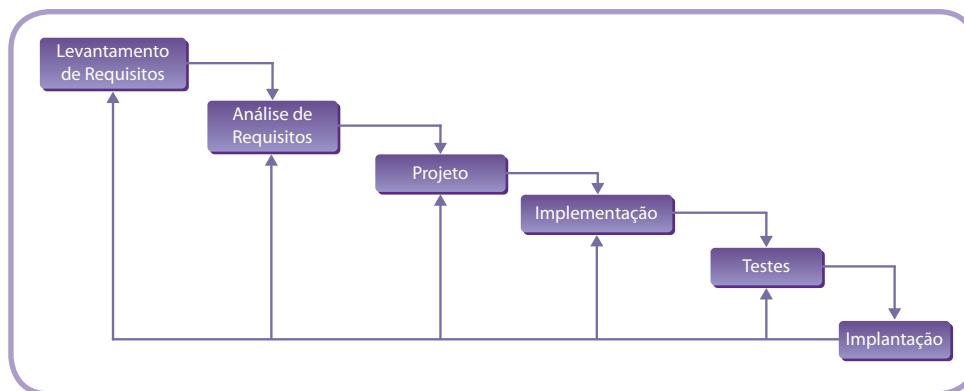
2.3 MODELOS DE CICLO DE VIDA

O desenvolvimento de um sistema envolve diversas fases, descritas na Seção 2.1. A um encadeamento específico dessas fases para a construção do sistema dá-se o nome de Modelo de Ciclo de Vida. Há diversos Modelos de Ciclo de vida, e a diferença entre um e outro está na maneira como as diversas fases são encadeadas. Nesta seção, dois modelos de ciclo de vida de sistema são descritos: *modelo em cascata* e o *modelo iterativo e incremental*.

2.3.1 O Modelo de Ciclo de Vida em Cascata

Também chamado de Modelo em Sequencial Linear, esse modelo tem seu diagrama parecido com uma série de cascatas, daí seu nome. Inicialmente descrito por Royce em 1970, foi a primeira realização de uma sequência padrão de tarefas. Eventualmente, pode haver uma retroalimentação de uma fase para a fase anterior, mas, em um ponto de vista macro, as fases seguem sequencialmente (ver Figura 2-1).

Figura 2-1: A abordagem de desenvolvimento de software em cascata (Serafini, 2008).

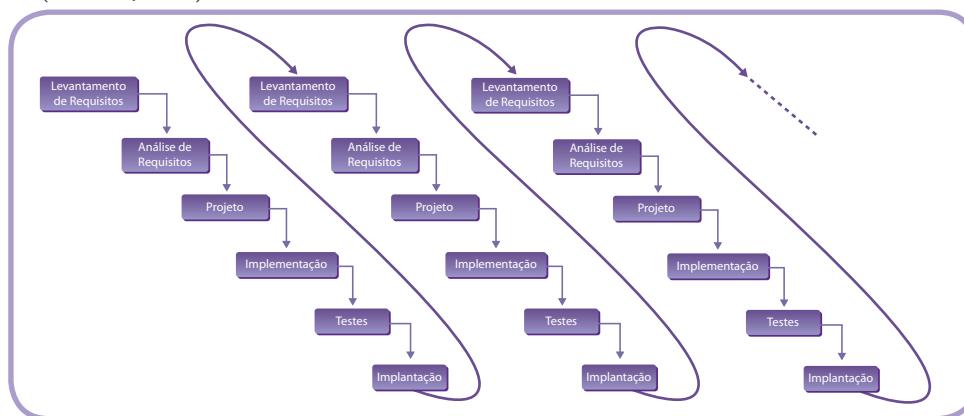


2.3.2 O Modelo de Ciclo de Vida Iterativo e Incremental

O modelo de ciclo de vida incremental e iterativo foi proposto como uma resposta aos problemas encontrados no modelo em cascata. Um processo de desenvolvimento segundo essa abordagem divide o desenvolvimento de um produto de software em ciclos. Em cada ciclo de desenvolvimento, podem ser identificadas as fases de análise, projeto, implementação e testes. Essa característica contrasta com a abordagem clássica, na qual as fases de análise, projeto, implementação e testes são realizadas uma única vez.

Bezerra (2007) explica que cada um dos ciclos considera um subconjunto de requisitos. Os requisitos são desenvolvidos uma vez que sejam alocados a um ciclo de desenvolvimento. No próximo ciclo, um outro subconjunto dos requisitos é considerado para ser desenvolvido, o que produz um novo incremento do sistema que contém extensões e refinamentos sobre o incremento anterior. Assim, o desenvolvimento evolui em versões, ao longo da construção incremental e iterativa de novas funcionalidades, até que o sistema completo esteja construído. Note que apenas uma parte dos requisitos é considerada em cada ciclo de desenvolvimento. Na verdade, um modelo de ciclo de vida iterativo e incremental pode ser visto como uma generalização da abordagem em cascata: o software é desenvolvido em incrementos e cada incremento é desenvolvido em cascata (ver Figura 2-2).

Figura 2-2: No processo incremental e iterativo, cada iteração é uma “minicascata” (Serafini, 2008).



Atenção

No modelo de ciclo de vida incremental e iterativo, um sistema de software é desenvolvido em vários passos similares (iterativo). Em cada passo, o sistema é estendido com mais funcionalidades (incremental).

Atenção

Os requisitos a serem considerados primeiramente devem ser selecionados com base nos riscos que eles fornecem. Os requisitos mais arriscados devem ser considerados tão logo seja possível.

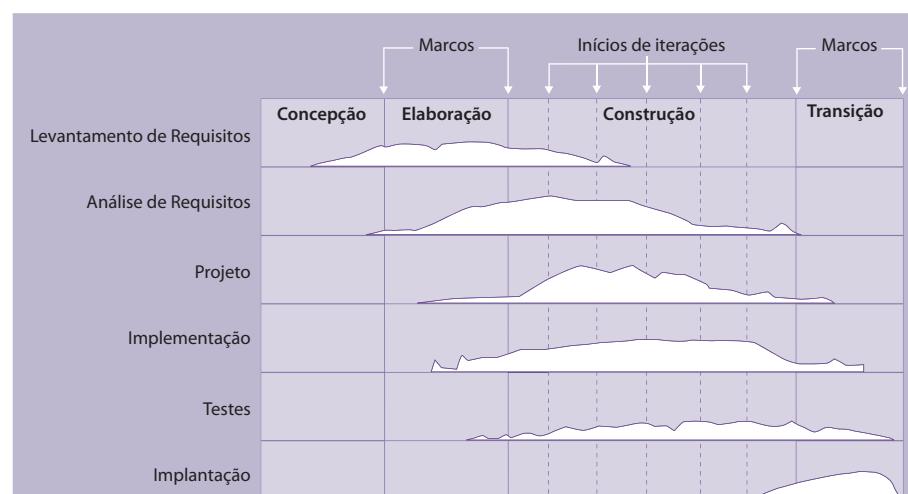
Organização Geral de um Processo Incremental e Iterativo

Para Bezerra (2007) o ciclo de vida de processo incremental e iterativo pode ser estudado segundo duas dimensões: dimensão temporal e dimensão de atividades (ou de fluxos de trabalho). A Figura 2-3 ilustra essas duas dimensões.

Na dimensão temporal, o processo está estruturado em *fases*, em cada qual há uma ou mais *iterações*. Cada iteração tem uma duração preestabelecida (de duas a seis semanas). Ao final de cada iteração, é produzido um incremento, ou seja, uma parte do sistema final. Um incremento pode ser liberado para os usuários, ou pode ser somente um incremento interno.

A dimensão de atividades compreende as realizadas durante a iteração de uma fase: levantamento de requisitos, análise de requisitos, projeto, implementação, testes e implantação (as mesmas atividades descritas na Seção 2.1). Essas atividades são apresentadas verticalmente na Figura 2-3.

Figura 2-3: Estrutura geral de um processo de desenvolvimento incremental e iterativo (Booch, 2006).



Em cada uma das fases, diferentes artefatos de software são produzidos, ou artefatos começados em uma fase anterior são estendidos com novos detalhes.

Cada fase é concluída com um *marco* (mostrado na parte superior da Figura 2-3). Um marco é um ponto do desenvolvimento no qual decisões sobre o projeto são tomadas e importantes objetivos são alcançados. Os marcos são úteis para o gerente de projeto estimar os gastos e o andamento do cronograma de desenvolvimento.

As fases do processo unificado delimitadas pelos marcos são as seguintes: concepção, elaboração, construção e transição. Essas fases são descritas a seguir de acordo com Falbo (2005):

- na *concepção*, é estabelecido o escopo do projeto e suas fronteiras, discriminando os principais casos de uso do sistema. Deve-se estimar os custos e prazos globais para o projeto como um todo e prover estimativas detalhadas para a fase de elaboração. Ao término desta fase, são examinados os objetivos do projeto para se decidir sobre a continuidade do desenvolvimento;
- na fase de *elaboração*, o propósito é analisar mais refinadamente o domínio do problema, estabelecer uma arquitetura de fundação sólida, desenvolver um plano de projeto para o sistema a ser construído e eliminar os elementos de projeto que oferecem maior risco. Nessa fase, também são planejadas as iterações da próxima fase, a de construção. Isso envolve definir a duração de cada iteração e o que será desenvolvido em cada iteração;
- na *construção*, um produto completo é desenvolvido de maneira iterativa e incremental para que esteja pronto para a transição à comunidade usuária;
- na *transição*, o software é disponibilizado à comunidade usuária. Após o produto ter sido colocado em uso, naturalmente surgem novas considerações que irão demandar a construção de novas versões para permitir ajustes do sistema, corrigir problemas ou concluir algumas características que foram postergadas..

Em cada iteração, uma proporção maior ou menor de cada uma dessas atividades é realizada, dependendo da fase em que se encontra o desenvolvimento. Por exemplo, a Figura 2-3 permite perceber que, na fase de transição, a atividade de implantação é a predominante. Por outro lado, na fase de construção, as atividades de análise, projeto e implementação são as predominantes. Normalmente, a fase de construção é a que possui

mais iterações. No entanto, as demais fases também podem conter iterações, dependendo da complexidade do sistema.

O principal representante da abordagem de desenvolvimento incremental e iterativa é o denominado *Processo Unificado Racional* (*Rational Unified Process*, RUP). Esse processo de desenvolvimento é patenteado pela empresa Rational, na qual trabalham os três amigos, Jacobson, Booch e Rumbaugh (em 2002, a IBM comprou a Rational).

Indicações



A descrição feita nesta seção é uma versão simplificada do Processo Unificado. Maiores detalhes sobre o Processo Unificado podem ser obtidos em (RUP, 2002).

Atividades



1. O que os seguintes termos significam: Análise e Projeto; Análise e Projeto Orientados a Objetos; UML? Como eles se relacionam uns com os outros?

Resposta:

Análise e Projeto: o termo "análise" está relacionado à definição do problema a ser solucionado no domínio do negócio. Já o termo "projeto" está ligado à solução do problema já entendido e definido na fase de análise. Para que o projeto seja feito, é necessário, então, uma análise previamente realizada do problema.

Análise e Projeto Orientados a Objetos: creio que, como diferença dos termos anteriores, só posso dizer que, em vez de estar centrado no problema e no processo de solução, está centrado nos objetos que farão parte da solução do problema.

UML: a UML é uma Linguagem de Modelagem de Objetos que veio da unificação de vários métodos e foi adotada como padrão pela OMG. A UML é usada na fase de análise e projeto para gerar a documentação específica dessas fases.

MODELAGEM DE CASOS DE USO

O modelo de casos de uso (MCU) é uma representação das *funcionalidades* externamente observáveis do sistema e dos *elementos externos* ao sistema que interagem com ele. O MCU é um modelo de análise (ver Seção 2.1.2) que representa um refinamento dos *requisitos funcionais* (ver Seção 2.1.1) do sistema em desenvolvimento. A ferramenta da UML utilizada na modelagem de casos de uso é o *diagrama de casos de uso*.

Neste capítulo, estudamos o *modelo de casos de uso*. Nossa objetivo aqui é apresentar os principais componentes desse modelo, assim como descrever dicas úteis na criação e documentação do mesmo. Além disso, também descrevemos a documentação suplementar que normalmente deve ser criada e associada ao modelo de casos de uso.

Fala Professor

Quae risus ac
e velit at tellus.
nassa portitor
sectetur magna.

3.1 MODELO DE CASOS DE USO

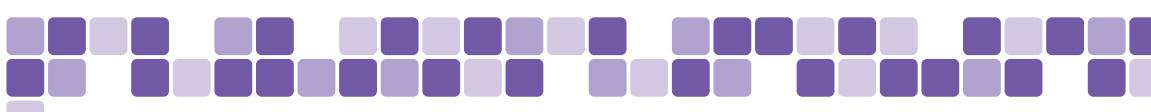
O MCU representa os possíveis usos de um sistema, conforme percebidos por um observador externo a este sistema. Cada um desses usos está associado a um ou mais requisitos funcionais identificados para o sistema. A construção desse modelo envolve a definição de diversos componentes: *casos de uso*, *atores* e *relacionamentos* entre eles. Bezerra (2007) descreve esses componentes separadamente.

3.1.1 Casos de Uso

Nenhum sistema computacional existe isoladamente. Todo sistema interage com atores humanos ou outros sistemas, que utilizam esse sistema para algum propósito e esperam que o sistema se comporte de acordo com as maneiras previstas.

Um **caso de uso** especifica um comportamento de um sistema segundo uma perspectiva externa e é uma descrição de um conjunto de sequências de ações realizadas pelo sistema para produzir um resultado de valor observável por um ator (Booch, 2006).

Conceitos



Um modelo de casos de uso típico contém vários casos de uso. A quantidade exata de casos de uso obviamente depende da complexidade do sistema em desenvolvimento: quanto mais complexo o sistema, maior a quantidade de casos de uso.

Atenção



Um caso de uso representa uma determinada funcionalidade de um sistema conforme percebida externamente. Representa também os agentes externos que *interagem* com o sistema. Um caso de uso, entretanto, não revela a estrutura e o comportamento internos do sistema.

Cada caso de uso de um sistema se define pela descrição narrativa (textual) das interações que ocorrem entre o(s) elemento(s) externo(s) e o sistema. A UML não define uma estrutura textual a ser utilizada na descrição de um caso de uso. Consequentemente, há vários estilos de descrição propostos para definir casos de uso, sendo que a escolha de um ou de outro fica a cargo da equipe de desenvolvimento, ou, então, pode ser uma restrição definida pelos clientes que encomendaram o sistema.

Formatos

O formato de uma descrição de caso de uso diz respeito à estrutura utilizada para organizar a sua narrativa textual. Os formatos comumente utilizados são o *contínuo* e o *numerado*, apresentados a seguir. Nesta explicação, é utilizado um exemplo correspondente ao caso de uso de saque de determinada quantia em um caixa eletrônico de um sistema bancário.

O formato contínuo foi introduzido por Ivar Jacobson e seus colaboradores. Nesse formato, a narrativa se dá através de texto livre. Como um exemplo desse tipo de formato, considere o caso de uso Realizar Saque em um caixa eletrônico. A descrição contínua deste caso de uso é fornecida no Quadro 3-1.

Quadro 3.1: Exemplo de descrição contínua

Este caso de uso inicia-se quando o Cliente chega ao caixa eletrônico e insere seu cartão. O Sistema requisita a senha do Cliente. Após o Cliente fornecer sua senha e esta ser validada, o Sistema exibe as opções de operações possíveis. O Cliente opta por realizar um saque. Então o Sistema requisita o total a ser sacado. O Cliente fornece o valor da quantidade que deseja sacar. O Sistema fornece a quantia desejada e imprime o recibo para o Cliente. O Cliente retira a quantia e o recibo, e o caso de uso termina.

No formato numerado, a narrativa é descrita por uma série de passos numerados. Considere como exemplo o mesmo caso de uso Realizar Saque. Sua descrição no formato numerado é fornecida no Quadro 3-1.

Quadro 3.2: Exemplo de descrição numerada

- 1) Cliente insere seu cartão no caixa eletrônico.
- 2) Sistema apresenta solicitação de senha.
- 3) Cliente digita senha.
- 4) Sistema valida a senha e exibe menu de operações disponíveis.
- 5) Cliente indica que deseja realizar um saque.
- 6) Sistema requisita o valor da quantia a ser sacada.
- 7) Cliente fornece o valor da quantia que deseja sacar.
- 8) Sistema fornece a quantia desejada e imprime o recibo para o Cliente.
- 9) Cliente retira a quantia e o recibo, e o caso de uso termina.

3.1.2 Atores

Um **ator** é uma entidade *externa* ao sistema que, de alguma maneira, *interage* na história ao caso de uso (Larman, 2007).



Conceitos

O termo “externo” nessa definição indica que atores não fazem parte do sistema. O termo “interage” significa que um ator troca informações com o sistema (envia informações para o sistema processar, ou recebe informações processadas provenientes do sistema). Atores representam a forma pela qual um sistema percebe o seu ambiente.

Atores de um sistema podem ser agrupados em diversas categorias. As categorias de atores definidas por Bezerra (2007), junto com exemplos de cada uma, são listadas a seguir:

1. *Cargos* (por exemplo, Empregado, Cliente, Gerente, Almoxarife, Vendedor, etc.).
2. *Organizações ou divisões de uma organização* (por exemplo, Empresa Fornecedor, Agência de Impostos, Administradora de Cartões, Almoxarifado, etc.).
3. *Outros sistemas de software* (por exemplo, Sistema de Cobrança, Sistema de Estoque de Produtos, etc.).

4. *Equipamentos* com os quais o sistema deve se comunicar (por exemplo, Leitora de Código de Barras, Sensor, etc.).

O mesmo autor ainda menciona que para todas as categorias de atores listadas anteriormente, os casos de uso representam alguma forma de interação, no sentido de troca de informações, entre o sistema e o ator. Normalmente, o ator inicia a sequência de interações correspondente a um caso de uso. Uma situação menos frequente, mas também possível, é um evento interno acontecer para que a sequência de interações do caso de uso em questão seja acionada.

Um aspecto importante a notar é que um ator corresponde a um papel representado em relação ao sistema. Por exemplo: a mesma pessoa pode agir (desempenhar um papel) como um ator, em um momento; e como outro ator, em outro momento. Para ser mais específico, em um sistema de vendas de livros via Internet, um indivíduo pode desempenhar o papel de *Cliente* que compra mercadorias. Esse mesmo indivíduo pode também desempenhar o papel de *Agendador*, ou seja, o funcionário da loja que agenda a entrega de encomendas. Outro exemplo: uma pessoa pode representar o papel de *Funcionário* de uma instituição bancária que faz a manutenção de um caixa eletrônico (para reabastecimento de numerário, por exemplo). Em outra situação de uso, essa mesma pessoa pode ser o *Cliente* do banco que realiza o saque de uma quantia no caixa eletrônico.

Um ator pode participar de muitos casos de uso (de fato, essa situação é comum na prática). Do mesmo modo, um caso de uso pode envolver a participação de vários atores, o que resulta na classificação dos atores em *primários* ou *secundários*. Um *ator primário* é aquele que inicia uma sequência de interações de um caso de uso. São eles os agentes externos para os quais o caso de uso traz benefício direto. As funcionalidades principais do sistema são definidas tendo em mente os objetivos dos atores primários. Já um *ator secundário* supervisiona, opera, mantém ou auxilia na utilização do sistema pelo atores primários. Atores secundários existem apenas para que os atores primários possam utilizar o sistema. Por exemplo, considere um programa para navegar pela Internet (ou seja, um navegador ou *browser*). Para que o *Usuário* (ator primário) requisite uma página ao programa, outro ator (secundário) está envolvido: o *Servidor Web*. Note que o ator primário *Usuário* é, de certa forma, auxiliado pelo ator secundário, *Servidor Web*, posto que é através deste último que o primeiro consegue alcançar seu objetivo (visualizar uma página da Internet) (Bezerra, 2007).

3.1.3 Relacionamentos

Um ator pode se relacionar a um ou mais casos de uso por meio do relacionamento de comunicação.

Existem também três tipos de relacionamentos entre casos de uso. Casos de uso podem ser descritos como versões especializadas de outros casos de uso (relacionamento de generalização/especialização); casos de uso podem ser incluídos como parte de outro caso de uso (relacionamento de inclusão); ou casos de uso podem estender o comportamento de outro caso de uso básico (relacionamento de extensão). O objetivo desses relacionamentos é tornar um modelo mais compreensível, evitar redundâncias entre casos de uso e permitir descrever casos de uso em camadas. Os significados (semântica) desses relacionamentos são discutidos nas próximas seções de acordo com a visão de Bezerra (2007). Antes de passar às próximas seções, entretanto, é bom enfatizar que todos esses relacionamentos possuem uma notação gráfica definida pela UML. Deixamos a apresentação dessas notações para a Seção 3.2.

Relacionamento de Comunicação

Um relacionamento de comunicação informa a que caso e uso o ator está associado. O fato de um ator estar associado a um caso de uso através de um relacionamento de comunicação significa que esse ator interage (troca informações) com o sistema por meio daquele caso de uso. Um ator pode se relacionar com mais de um caso de uso do sistema. O relacionamento de comunicação é, de longe, o mais comumente utilizado de todos os relacionamentos.

Relacionamento de Inclusão

O relacionamento de inclusão entre casos de uso significa que o caso de uso base incorpora explicitamente o comportamento de outro caso de uso. O local em que esse comportamento é incluído deve ser indicado na descrição do caso de uso base, através de uma referência explícita à chamada ao caso de uso incluído. Um relacionamento de inclusão é empregado quando há uma porção de comportamento que é similar ao longo de um ou mais casos de uso e não se deseja repetir a sua descrição. Para evitar redundância e assegurar reuso, extrai-se essa descrição e compartilha-se a mesma entre diferentes casos de uso. Um relacionamento de inclusão é representado por uma dependência, estereotipada como inclui (*include*). Como terminologia, o *caso de uso inclusor* é aquele que inclui o comportamento de outro caso de uso; já o *caso de uso incluído* é aquele cujo comportamento é incluído por outros.

Como exemplo de utilização do relacionamento de inclusão, considere um sistema de controle de transações bancárias. Alguns casos de uso desse sistema são **Obter Extrato**, **Realizar Saque** e **Realizar Transferência**. Esses casos de uso têm uma sequência de interações em comum, a saber, a sequência de interações para validar a senha do cliente do banco. Essa sequência de interações em comum pode ser descrita em um caso de uso **Fornecer Identificação**.

Dessa forma, todos os casos de uso que utilizam essa sequência de interações podem fazer referência ao caso de uso **Fornecer Identificação** através do relacionamento de inclusão.

Relacionamento de Extensão

Considere dois casos de uso, A e B. Um relacionamento de extensão de A para B indica que um ou mais dos cenários de B podem incluir o comportamento especificado por A. Nesse caso, diz-se que B *estende* A. O caso de uso A é chamado de *estendido*, e o caso de uso B de *extensor*.

Um relacionamento de extensão entre casos de uso significa que o caso de uso base incorpora implicitamente o comportamento de um outro caso de uso em um local especificado em sua descrição como sendo um ponto de extensão. O caso de uso base poderá permanecer isolado, mas, sob certas circunstâncias, seu comportamento poderá ser estendido pelo comportamento de um outro caso de uso.

Um relacionamento de extensão é utilizado para modelar uma parte de um caso de uso que o usuário considera como opcional. Desse modo, separa-se o comportamento opcional do comportamento obrigatório. Um relacionamento de extensão também poderá ser empregado para modelar um subfluxo separado, que é executado somente sob determinadas condições. Assim como o relacionamento de inclusão, o relacionamento de extensão é representado como uma associação de dependência, agora estereotipada como *estende* (*extend*).

Como um exemplo de possível utilização do relacionamento de extensão, considere um processador de textos. Considere que um dos casos de uso deste sistema seja **Editar Documento**, que descreve a sequência de interações que ocorre quando um usuário edita um documento. No cenário típico desse caso de uso, o ator abre o documento, modifica-o, salva as modificações e fecha o documento. Mas, em outro cenário, o ator pode desejar que o sistema faça uma verificação ortográfica no documento. Em outro, ele pode querer realizar a substituição de um fragmento de texto por outro. Tanto a verificação ortográfica quanto a substituição de texto são esporádicas, não usuais, opcionais. Dessa

forma, os casos de uso Corrigir Ortografia e Substituir Texto podem ser definidos como extensões de Editar Documento.

Relacionamento de Generalização

Os relacionamentos descritos anteriormente (inclusão e extensão) implicam o reuso do comportamento de um caso de uso na definição de outros casos de uso.

Outro relacionamento no qual o reuso de comportamento é evidente é o de generalização, o qual pode existir entre dois casos de uso ou entre dois atores. Esse relacionamento permite que um caso de uso (ou um ator) herde características de um caso de uso (ator) mais genérico, este último normalmente chamado de caso de uso (ator) *base*. O caso de uso (ator) herdeiro pode especializar o comportamento do caso de uso (ator) base. Vamos examinar cada uma dessas duas alternativas.

Na generalização entre casos de uso, sejam A e B dois casos de uso. Quando B herda de A, as sequências do comportamento de A valem também para B. Quando for necessário, B pode redefinir as sequências de comportamento de A. Além disso, B (o caso de uso herdeiro) participa em qualquer relacionamento no qual A (o caso de uso pai) participa. Ou seja, todo ator que puder realizar o caso de uso pai pode também realizar qualquer caso de uso filho.

Como um exemplo de *generalização entre atores*, analise uma biblioteca na qual pode haver dois tipos de usuários: alunos e professores. Os dois tipos de usuário podem realizar empréstimos de títulos de livros e reservas de exemplares. Suponha que, com relação àqueles processos do negócio (empréstimos e reservas), não haja diferença entre um professor e um aluno. Suponha, ainda, que somente o professor pode requisitar a compra de títulos de livros à biblioteca. Nessa situação, poder-se-ia definir um ator chamado *Usuário* e outro ator chamado *Professor*, que herdaria de *Usuário*. Assim, o ator *Professor* herda todos os casos de uso do ator *Usuário*, significando que um professor pode interagir com todos os casos de uso com que um usuário comum interage. Além disso, um professor também interage com o caso de uso de requisição de compra de novos títulos de livros, sendo que este caso de uso é específico para professores.

Quando Usar Relacionamentos no MCU

Uma dúvida comum é saber que tipo de relacionamento utilizar em dada situação. Na verdade, não há regras para saber quando utilizar um ou outro tipo de relacionamento; há somente heurísticas. Sendo assim,

para a escolha do relacionamento a utilizar, as seguintes heurísticas descritas por Bezerra (2007) podem ser seguidas:

- *Inclusão*. Use inclusão quando o mesmo comportamento se repetir em mais de um caso de uso. Este comportamento comum está *necessariamente* contido em *todos* os cenários dos casos de uso inclusores.
- *Extensão*. Use extensão quando um comportamento *eventual* de um caso de uso tiver de ser descrito.
- *Generalização entre casos de uso*. Use generalização entre casos de uso quando você identificar dois ou mais casos de uso com comportamentos semelhantes.
- *Generalização entre atores*. Use generalização quando precisar definir um ator que desempenhe um papel que já é desempenhado por outro ator em relação ao sistema, mas que também possui comportamento particular adicional.

A Tabela 3-1 resume as possibilidades de existência de relacionamentos entre os elementos do modelo de casos de uso.

Tabela 3-1: Possibilidades de relacionamentos entre elementos do modelo (Bezerra, 2007).

Elementos	Comunicação	Extensão	Inclusão	Herança
Caso de Uso e Caso de Uso		X	X	X
Autor e Autor				X
Caso de Uso e Autor	X			

3.2 DIAGRAMA DE CASOS DE USO

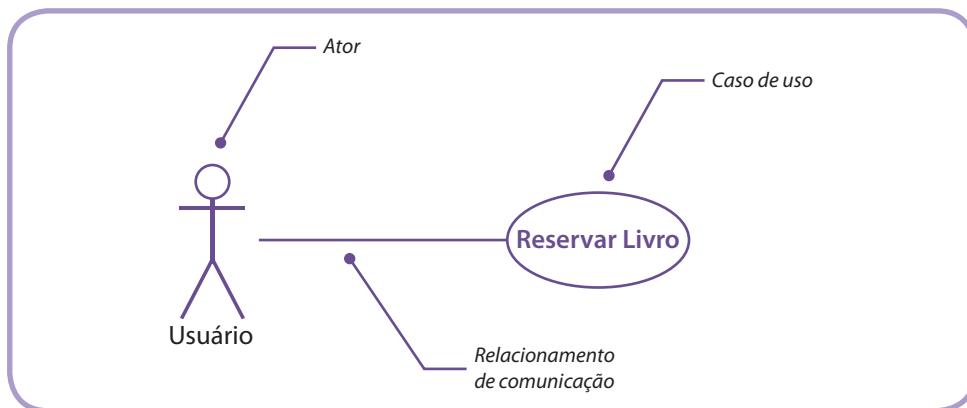
Na Seção 3.1, descrevemos os principais componentes de um MCU: atores, casos de uso e relacionamentos. Apresentamos, também, diversos estilos de descrição possíveis para um caso de uso. O conjunto de descrições dos casos de uso e atores corresponde à *perspectiva textual* do MCU. Nesta seção, nosso interesse recai sobre a *perspectiva gráfica* do MCU, representada pelo diagrama de casos de uso (DCU).

O DCU é um dos diagramas da UML e corresponde a uma visão externa de alto nível do sistema. Esse diagrama representa *graficamente* os atores, casos de uso e relacionamentos entre esses elementos. O DCU tem o

objetivo de ilustrar, em um nível alto de abstração, quais elementos externos interagem com que funcionalidades do sistema. Nesse sentido, a finalidade de um DCU é apresentar um tipo de “diagrama de contexto” que apresenta os elementos externos de um sistema e as maneiras segundo as quais eles as utilizam.

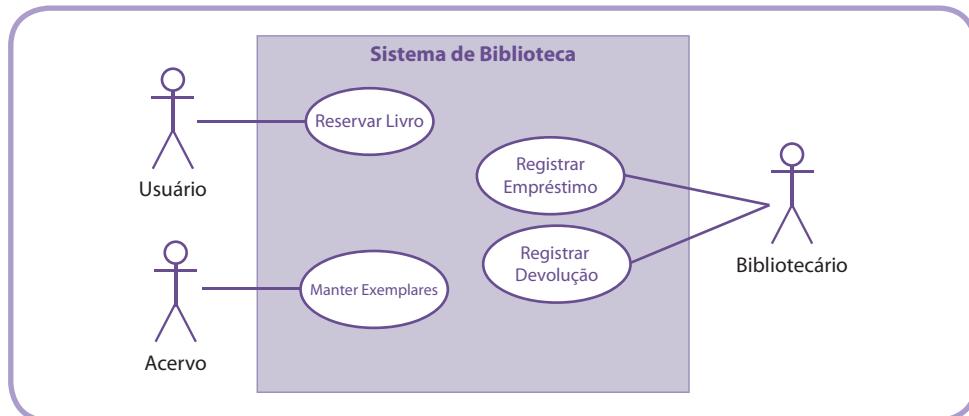
A notação utilizada para ilustrar atores em um DCU é a figura de um boneco, com o nome do ator definido abaixo da figura. Note que essa notação não corresponde ao significado de ator em sua completude, porque um ator nem sempre corresponde a seres humanos, como a notação leva a entender (ver Seção 3.1.2). Cada caso de uso é representado por uma elipse. O nome do caso de uso é posicionado dentro ou abaixo da elipse. Um relacionamento de comunicação é representado por um segmento de reta ligando ator e caso de uso. Um ator pode estar associado através do relacionamento de comunicação a vários casos de uso em um DCU. Pela UML, também é possível imprimir um sentido ao segmento de reta correspondente a um relacionamento de comunicação, para denotar o sentido das informações. No entanto, essa situação tem pouco uso prático e, normalmente, o segmento de reta do relacionamento de comunicação é definido sem o sentido. A Figura 3-1 ilustra a notação da UML para representar atores, casos de uso e relacionamentos de comunicação. Esses três elementos são os mais comumente utilizados.

Figura 3-1: Notação para ator, caso de uso e relacionamento de comunicação (Bezer-
ra, 2007).



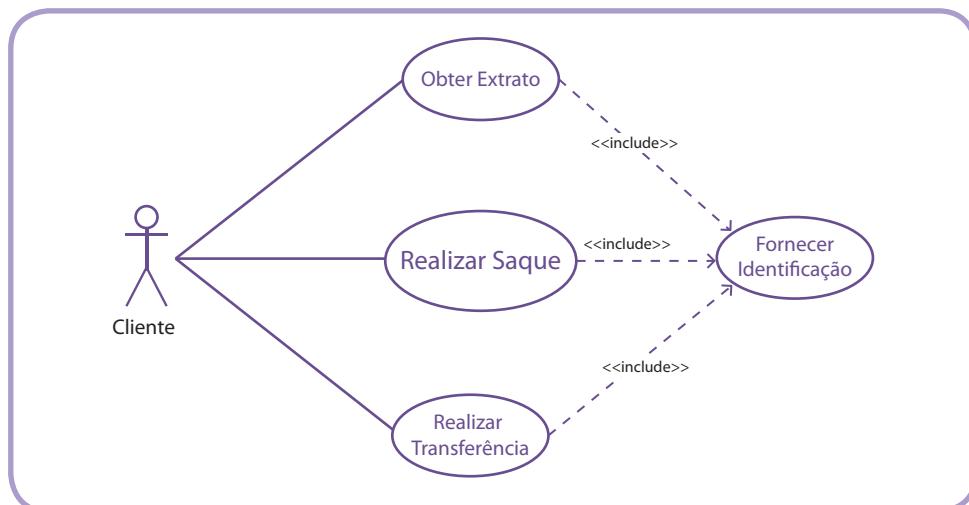
Pode-se também representar a fronteira do sistema em um diagrama de casos de uso. Essa fronteira é representada por um retângulo no interior do qual são inseridos os casos de uso. Os atores são posicionados do lado de fora do retângulo, para enfatizar a divisão entre o interior e o exterior do sistema. A Figura 3-2 apresenta um exemplo de diagrama de casos de uso no qual se utiliza uma fronteira. Por simplicidade, esse diagrama exibe um único caso de uso.

Figura 3-2: Exemplo de diagrama de casos de uso utilizando um retângulo de fronteira (Bezerra, 2007).



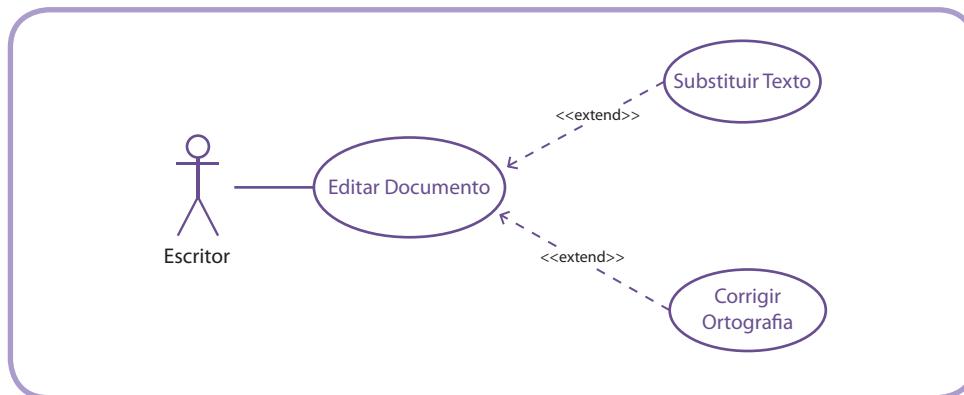
O relacionamento de inclusão (ver Seção 3.1.3), em que um caso de uso A inclui um caso de uso B, é representado por uma seta direcionada de A para B. O eixo dessa seta é tracejado e rotulado com o estereótipo predefinido `<<include>>`. A Figura 3-3 ilustra a representação do relacionamento de inclusão em um DCU. Esse diagrama informa que os casos de uso `Obter Extrato`, `Realizar Saque` e `Realizar Transferência` têm uma sequência de interações em comum, a saber, a sequência para autenticar o cliente do banco que está representada pelo caso de uso `Fornecer Identificação`.

Figura 3-3: Exemplo de relacionamento de inclusão (Falbo, 2005).



O relacionamento de extensão, em que um caso de uso A estende um caso de uso B, é representado por uma seta direcionada de A para B. Essa seta, de eixo também tracejado, é rotulada com outro estereótipo predefinido pela UML, o `extend`. A Figura 3-4 ilustra a representação desse relacionamento. Essa figura mostra que os casos de uso `Corrigir Ortografia` e `Substituir Texto` têm sequências de interações que são *eventualmente* utilizadas quando o ator `Escritor` estiver utilizando a caso de uso `Edita Documento`.

Figura 3-4: Exemplo de relacionamento de extensão (Bezerra, 2007).



A Figura 3-5 ilustra exemplos do relacionamento de generalização em suas duas formas: entre casos de uso e entre atores. A generalização entre casos de uso indica que os casos de uso Realizar Pagamento com Cartão de Crédito e Realizar Pagamento com Dinheiro são casos especiais do caso de uso Realizar Pagamento. Já a generalização entre os atores Usuário e Professor indica que este último pode interagir com qualquer caso de uso com que um usuário comum interage.

Figura 3-5: Exemplo do relacionamento de generalização (Bezerra, 2007).

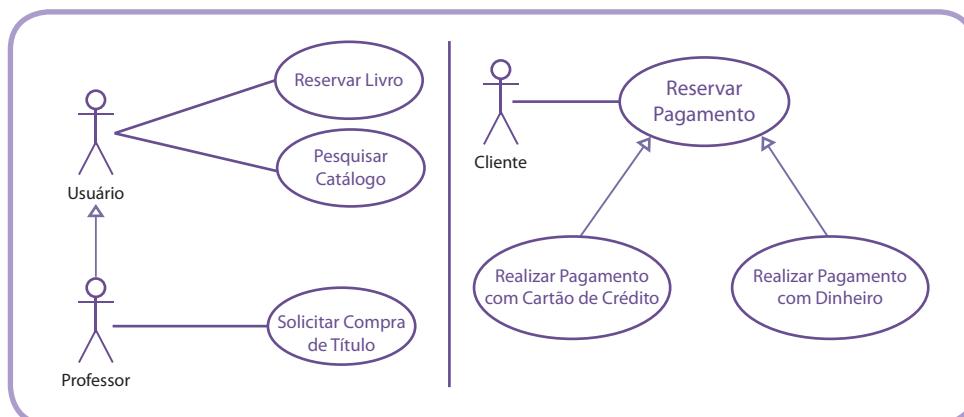
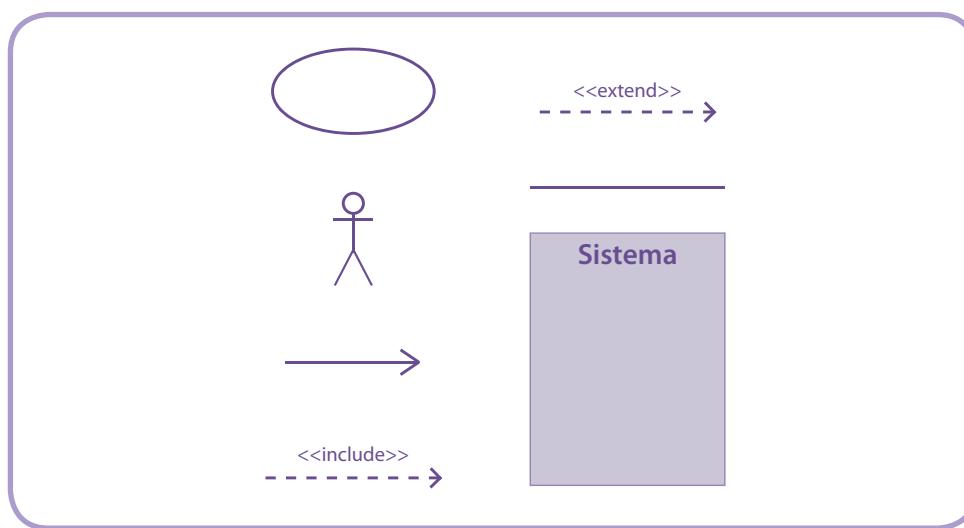


Figura 3-6: Elementos gráficos da UML para o desenho de um DCU.



Além disso, o Professor pode participar em outros casos de uso específicos a ele, como, por exemplo, Solicitar Compra de Título.

3.3 DOCUMENTAÇÃO DE CASOS DE USO

Conforme mencionamos na Seção 3.1.1, a UML não define uma estruturação específica a ser utilizada na descrição de um caso de uso. Por conta disso, há diversas propostas de descrição. Nesta seção, é apresentada uma proposta para a descrição de um caso de uso expandido. No entanto, antes de começar a apresentação, o leitor deve atentar para o fato de que essa proposta é apenas uma sugestão. Pode ser que uma equipe de desenvolvimento não precise utilizar todos os itens aqui mencionados; pode até ser que mais detalhes sejam necessários. De qualquer modo, a equipe de desenvolvimento deve utilizar os itens de descrição que forem realmente úteis e mais inteligíveis para o usuário.

Vamos tomar como base a proposta definida por Bezerra (2007):

Nome

O primeiro item que deve constar da descrição de um caso de uso é o seu nome, que deve ser o mesmo utilizado no DCU. Cada caso de uso deve ter um nome único.

Identificador

O identificador é um código único para cada caso de uso que permite fazer referência cruzada entre diversos documentos relacionados com o MCU (por exemplo, a descrição de um cenário do caso de uso pode fazer referência a esse identificador). Uma convenção de nomenclatura que recomendamos é usar o prefixo CSU seguido de um número sequencial. Por exemplo: CSU01, CSU02.

Sumário

Uma pequena declaração do objetivo do ator ao utilizar o caso de uso (no máximo, duas frases).

Autor Primário

O nome do ator que inicia o caso de uso. (Note que pode ser que o ator não inicie o caso de uso, mas, ainda assim, seja alvo do resultado produzido pelo caso de uso.) Um caso de uso possui apenas um ator primário.

Ator(es) Secundário(s)

Os nomes dos demais elementos externos participantes do caso de uso, os atores secundários (ver Seção 3.1.2). Um caso de uso possui zero ou mais atores secundários.

Precondições

Pode haver alguns casos de uso cuja realização não faz sentido em qualquer momento, mas, ao contrário, somente quando o sistema está em um determinado estado com certas propriedades.

Uma precondição de um caso de uso define que hipóteses são assumidas como verdadeiras para que o caso de uso tenha início. Este item da descrição pode conter zero ou mais precondições.

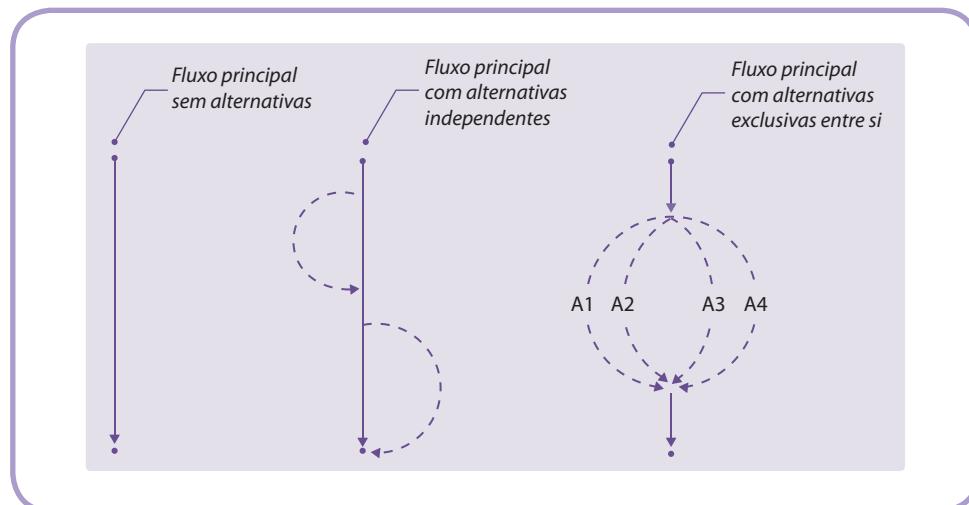
Fluxo Principal

O fluxo principal de um caso de uso, por vezes chamado de fluxo básico, corresponde à sua descrição da sequência de passos usual. Isso significa que fluxo principal descreve o que *normalmente* acontece quando o caso de uso é utilizado. Toda descrição de caso de uso deve ter um fluxo principal. O texto descritivo desse fluxo (assim como dos fluxos alternativos e de exceção, descritos a seguir) deve ser claro e conciso. Além disso, nessa descrição, o modelador deve se ater ao domínio do problema, e não à solução deste. Portanto, o jargão computacional não deve ser utilizado na descrição de casos de uso; ao contrário, casos de uso devem ser escritos do ponto de vista do usuário e usando a terminologia deste.

Fluxo(s) Alternativo(s)

Por vezes, um caso de uso pode ser utilizado de diversas maneiras possíveis, o que resulta na existência de diversos *cenários* para o mesmo. Esses fluxos podem ser utilizados para descrever o que acontece quando o ator opta por utilizar o caso de uso de uma forma alternativa, diferente da descrita no fluxo principal, para alcançar o seu objetivo. Fluxos alternativos também podem ser utilizados para descrever situações de escolha exclusivas entre si (em que há diversas alternativas e somente uma deve ser realizada). A Figura 3-7 ilustra de forma esquemática essas situações de uso dos fluxos alternativos. As linhas tracejadas representam fluxos alternativos. A linha sólida representa o fluxo principal. Note que a descrição de um caso de uso pode ter somente o fluxo principal, sem fluxos alternativos.

Figura 3-7: Fluxos alternativos em um caso de uso (Bezerra, 2007).



Pós-Condições

Em alguns casos, em vez de gerar um resultado observável, o estado do sistema pode mudar após um caso de uso ser realizado. Essa situação é especificada como uma pós-condição. Uma pós-condição é um estado que o sistema alcança após certo caso de uso ter sido executado.

A pós-condição deve declarar *qual* é esse estado, em vez de declarar como esse estado foi alcançado. Um exemplo típico de pós-condição é a declaração de que uma (ou mais de uma) informação foi modificada, removida ou criada no sistema. Pós-condições são normalmente descritas utilizando o tempo pretérito.

Regras de Negócio

São políticas, condições ou restrições que devem ser consideradas na execução dos processos existentes em uma organização.

Alguns exemplos de *regras do negócio* (não pertencentes a uma mesma organização) são apresentadas aqui:

- o valor total do pedido é igual à soma dos totais dos itens do pedido acrescidos de 10% de taxa de entrega;
- o número máximo de alunos por turma é igual a 30.

3.4 MODELO DE CASOS DE USO NO PROCESSO DE DESENVOLVIMENTO

Casos de uso formam uma base natural pela qual é possível planejar e realizar as iterações do desenvolvimento. Para isso, os casos de uso devem

ser divididos em grupos. Cada grupo é alocado a uma iteração. Então, o desenvolvimento do sistema segue a alocação realizada: em cada iteração, o grupo de casos de uso correspondente é detalhado e desenvolvido. O processo continua até que todos os grupos de casos de uso tenham sido desenvolvidos e o sistema esteja completamente construído.

Um fator importante para o sucesso do desenvolvimento do sistema é considerar os casos de uso mais importantes primeiramente.



Atenção

3.5. ESTUDO DE CASO

A partir desta seção, um estudo de caso proposto por Bezerra (2007) começa a ser desenvolvido. Esse estudo tem o objetivo de consolidar os principais conceitos teóricos descritos e oferecer uma visão prática sobre como os modelos apresentados nesta apostila são desenvolvidos. Batizamos o sistema de nosso estudo de caso com o nome de *Sistema de Controle Acadêmico* (SCA).

O desenvolvimento de nosso estudo de caso é feito de forma incremental: em cada capítulo deste material em que houver uma seção denominada “Estudo de caso”, uma parte do desenvolvimento é apresentada. É importante notar que, para manter a descrição em um nível de simplicidade e clareza aceitáveis para um material didático, muitos detalhes do desenvolvimento são deliberadamente ignorados.

3.5.1 Descrição da Situação

O estudo de caso é sobre uma faculdade que precisa de uma aplicação para controlar alguns processos acadêmicos, como inscrições em disciplinas, lançamento de notas, alocação de recursos para turmas, etc. Após o levantamento de requisitos inicial desse sistema, os analistas chegaram à seguinte lista de requisitos funcionais:

R1. O Sistema deve permitir que alunos visualizem as notas obtidas por semestre letivo.

R2. O sistema deve permitir o lançamento das notas das disciplinas lecionadas em um semestre letivo e controlar os prazos e atrasos neste lançamento.

R3. O sistema deve manter informações cadastrais sobre disciplinas no currículo escolar.

Dentre outros requisitos...

3.5.2 Regras do Negócio

Algumas regras iniciais de negócio também foram identificadas para o sistema.

RN01. Quantidade máxima de inscrições por semestre letivo: em um semestre letivo, um aluno não pode se inscrever em uma quantidade de disciplina cuja soma de créditos ultrapasse 20.

RN02. Quantidade de alunos possíveis: uma oferta de disciplina em uma turma não pode ter mais de 40 alunos inscritos.

RN03. Pré-requisitos para uma disciplina: um aluno não pode se inscrever em uma disciplina para a qual não possua os pré-requisitos necessários.

Dentre outras regras de negócio...

3.5.3 Documentação do MCU

Nesse sistema de controle acadêmico, o analista identificou e documentou os seguintes atores:

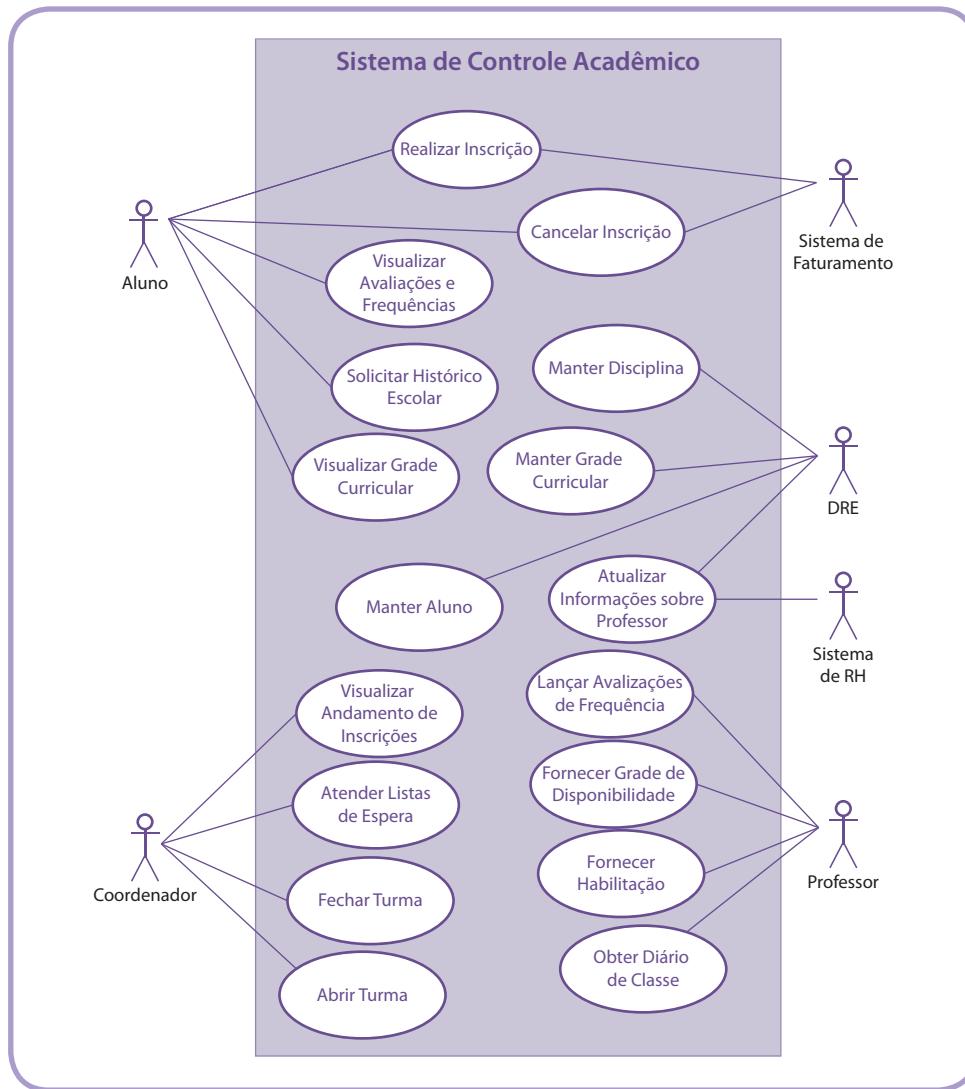
- *Aluno*: indivíduo que está matriculado na faculdade e tem interesse em se inscrever em disciplinas do curso.
- *Professor*: indivíduo que leciona disciplinas na faculdade.
- *Coordenador*: pessoa interessada em agendar as alocações de turmas e professores, e visualizar o andamento de inscrições dos alunos.
- *Departamento de Registro Escolar (DRE)*: departamento da faculdade interessado em manter informações sobre os alunos matriculados e respectivos históricos escolares.
- *Sistema de Recursos Humanos*: este sistema é responsável por fornecer informações cadastrais sobre os professores.
- *Sistema de Faturamento*: este sistema tem interesse em obter informações sobre inscrições dos alunos para realizar o controle de pagamento de mensalidades.

O analista também identificou os casos de uso a seguir, e os organizou em três pacotes: *Gerenciamento de Inscrições*, *Gerenciamento de Recursos Acadêmicos* e *Acompanhamento de Semestre Letivo*.

- Gerenciamento de Inscrições
 - o Realizar Inscrição
 - o Cancelar Inscrição (*Aluno cancela inscrições em uma ou mais ofertas de disciplinas em que havia solicitado inscrição*)
 - o Visualizar Grade Curricular (*Aluno visualiza a grade curricular atual*)
 - o Visualizar Andamento de Inscrições
 - o Abrir Turma (*Coordenador abre uma turma*)
 - o Fechar Turma (*Coordenador fecha uma turma*)
 - o Atender Listas de Espera
- Gerenciamento de Recursos Acadêmicos
 - o Manter Grade Curricular (*Coordenador define informações sobre uma grade curricular*)
 - o Manter Disciplina
 - o Manter Aluno (*DRE mantém informações sobre aluno*)
 - o Fornecer Grade de Disponibilidade
 - o Fornecer Habilidades (*Professor informa as disciplinas da grade curricular que está apto a lecionar*).
 - o Atualizar Informações sobre Professor
- Acompanhamento de Semestre Letivo
 - o Lançar Avaliações e Frequências
 - o Obter Diário de Classe (*Professor obtém o diário de classe para determinado mês do semestre letivo*)
 - o Visualizar Avaliações e Frequências
 - o Solicitar Histórico Escolar (*Aluno solicita a produção de seu histórico escolar*)

A seguir, são apresentados o diagrama de casos de uso e as descrições de alguns dos casos de uso no formato essencial e expandido. A descrição dos demais casos de uso fica como exercício para o leitor.

Figura 3-8: Diagrama de casos de uso para o SCA (Bezerra, 2007).



Realizar Inscrição (CSU01)

Sumário: Aluno usa o sistema para realizar inscrição em disciplinas.

Ator Primário: Aluno

Atores Secundários: Sistema de Faturamento

Precondições: O Aluno está identificado pelo sistema.

Fluxo Principal

1. O Aluno solicita a realização de inscrição.

2. O sistema apresenta as disciplinas para as quais o Aluno tem pré-requisitos (conforme a RN03), excetuando-se as que este já tenha cursado.
3. O Aluno define a lista de disciplinas que deseja cursar no próximo semestre letivo e as relaciona para inscrição.
4. Para cada disciplina selecionada, o sistema designa o Aluno para uma turma que apresente uma oferta para tal disciplina.
5. O sistema registra a inscrição do Aluno, envia os dados sobre a mesma para o Sistema de Faturamento e o caso de uso termina.

Fluxo Alternativo (1): Inclusão em lista de espera

4a: Não há oferta disponível para alguma disciplina selecionada pelo Aluno (conforme a RN02)

1. O sistema reporta o fato e fornece a possibilidade de inserir o Aluno em uma lista de espera.
2. O Aluno aceita.
3. O sistema o insere na lista de espera e apresenta a posição na qual o Aluno foi inserido na lista.
4. O caso de uso retorna ao passo 4 do fluxo principal.

Fluxo Alternativo (2): Não inclusão em lista de espera

4b: Não há oferta disponível para alguma disciplina selecionada pelo Aluno (conforme a RN02)

1. O sistema reporta o fato e fornece a possibilidade de inserir o Aluno em uma lista de espera.
2. O Aluno não aceita.
3. O caso de uso prossegue a partir do passo 4.

Fluxo Alternativo (3): Violação de RN01

4c: O Aluno atingiu a quantidade máxima de inscrições possíveis em um semestre letivo (conforme a RN01).

1. O sistema informa ao Aluno a quantidade de disciplinas que ele pode selecionar.

2. O caso de uso retoma ao passo 2 do fluxo principal.

Pós-condições: O aluno foi inscrito em uma das turmas de cada uma das disciplinas desejadas, ou foi adicionado a uma ou mais listas de espera.

Regras de Negócio: RN01, RN02, RN03

Atividades



1. Altere os seguintes “nomes de casos de uso” de acordo com as nomenclaturas apresentadas neste capítulo:

- a) Cliente realiza transferência de fundos em um caixa eletrônico.
- b) Usuários compram livros na livraria.
- c) É produzido um relatório de vendas para o gerente.
- d) Hóspede se registra em um hotel.

2. Desenhe diagramas de casos de uso para os seguintes sistemas:

- a) A biblioteca de uma universidade.
- b) O seu aparelho celular.

MODELAGEM DE CLASSES DE ANÁLISE

Bezerra (2007) afirma que o modelo de casos de uso de um sistema é construído para formar a visão de casos de uso do sistema, a qual fornece uma perspectiva do sistema a partir de um ponto de vista *externo*. De posse da visão de casos de uso, os desenvolvedores precisam prosseguir no desenvolvimento do sistema.

A funcionalidade externa de um sistema orientado a objetos é fornecida por meio de colaborações entre objetos. Externamente ao sistema, os atores visualizam resultados de cálculos, relatórios produzidos, confirmações de requisições realizadas, etc. Internamente, os objetos do sistema colaboram uns com os outros para produzir os resultados visíveis de fora. Essa colaboração pode ser vista sob o *aspecto dinâmico* e sob o *aspecto estrutural estático* (Bezerra, 2007).

O *aspecto dinâmico* de uma colaboração entre objetos descreve a troca de mensagens entre os mesmos e a sua reação a eventos que ocorrem no sistema. O aspecto dinâmico de uma colaboração é representado pelo Modelo de Interações e é estudado no Capítulo 5.

O *aspecto estrutural estático* de uma colaboração permite compreender como o sistema está estruturado internamente para que as funcionalidades externamente visíveis sejam produzidas. Esse aspecto é dito *estático* porque não apresenta informações sobre como os objetos do sistema interagem no decorrer o tempo (isso é representado no aspecto dinâmico da colaboração). Também é dito *estrutural* porque a estrutura das classes de objetos componentes do sistema e as relações entre elas são representadas.

Este capítulo inicia a descrição do aspecto estrutural estático de um sistema orientado a objetos. Esse aspecto é representado pelo Modelo de Classes, da mesma forma que o aspecto funcional é representado pelo Modelo de Casos de Uso. A ferramenta da UML utilizada para representar o aspecto estrutural estático é o *diagrama de classes*. O *modelo de classes* é composto desse diagrama e da descrição textual associada ao mesmo.



Fala Professor

4.1 DIAGRAMA DE CLASSESS

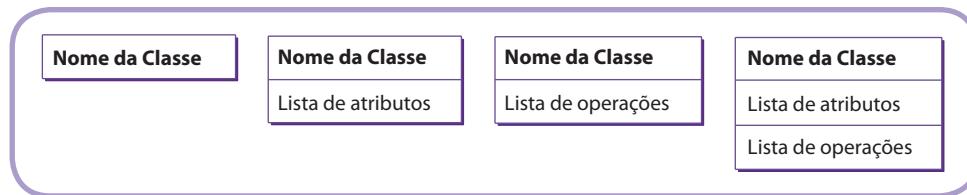
O *diagrama de classes* é utilizado na construção do modelo de classes desde o nível de análise até o nível de especificação. De todos os diagramas da UML, esse é o mais rico em termos de notação. Nesta seção, são apresentados os elementos do diagrama de classes utilizados para a construção do modelo de classes de nível de análise.

4.1.1. Classes

Não há nada mais central e crucial para qualquer método orientado a objetos do que o processo de descoberta de quais classes devem ser incluídas no modelo. O cerne de um modelo Orientado a Objetos é exatamente seu conjunto de classes (Falbo, 2005).

Uma classe é representada por uma “caixa” com, no máximo, três compartimentos exibidos. No primeiro compartimento (de cima para baixo) é exibido o nome da classe. Por convenção, esse nome é apresentado no singular e com as palavras componentes começando por maiúsculas. No segundo compartimento, são declarados os atributos, que correspondem às informações que um objeto armazena. Finalmente, no terceiro compartimento, são declaradas as operações, que correspondem às ações que um objeto sabe realizar.

Figura 4-1: Possíveis notações para uma classe na UML (Bezerra, 2007).



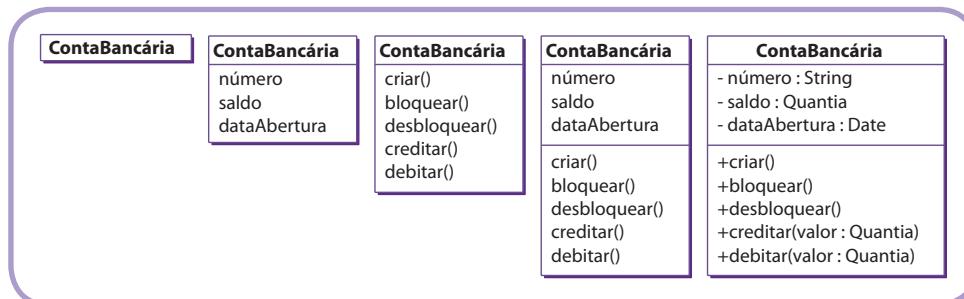
As possíveis notações da UML para representar classes são apresentadas na Figura 4-1. O grau de abstração desejado em um dado momento do desenvolvimento do modelo de classes direciona a utilização de uma ou outra notação.

Estruturalmente, uma classe é composta de *atributos* e de *operações*. Os atributos correspondem à descrição dos dados armazenados pelos objetos de uma classe. A cada atributo de uma classe está associado um conjunto de valores que esse atributo pode assumir. As operações correspondem à descrição das ações que os objetos de uma classe sabem realizar. Ao contrário dos atributos (para os quais cada objeto tem o seu próprio valor), objetos de uma classe compartilham as mesmas operações. O nome de uma operação normalmente contém um verbo e um complemento, e terminam com um par de parênteses. (Na descrição

do modelo de classes de especificação, é apresentada a verdadeira utilidade desse par de parênteses) (Bezerra, 2007).

A Figura 4-2 ilustra exemplos de representação de uma mesma classe, ContaBancária, em diferentes graus de abstração.

Figura 4-2: Diferentes graus de abstração na notação de classe (Bezerra, 2007).

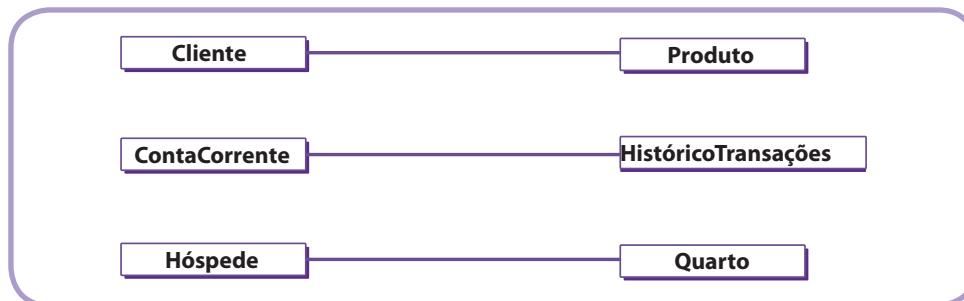


4.1.2 Associações

Da Seção 1.3.1, sabe-se que cada ocorrência de uma classe é chamada de *objeto* ou *instância*. Um ponto importante acerca de objetos de um sistema é o fato de que eles podem se relacionar uns com os outros. A existência de um relacionamento entre dois objetos possibilita a troca de mensagens entre os mesmos. Portanto, em última análise, relacionamentos entre objetos permitem que eles colaborem entre si a fim de produzir as funcionalidades do sistema (Bezerra, 2007).

No diagrama de classes, podemos representar a existência de relacionamentos entre objetos. Para representar o fato de que objetos podem se relacionar uns com os outros, existe outro elemento na notação do diagrama de classes: a *associação*. Esse elemento representa relacionamento estrutural que especifica objetos de um item conectados a objetos de outro item (Booch, 2005). Uma associação é representada no diagrama de classes por uma linha (normalmente um segmento de reta) ligando as classes às quais pertencem os objetos relacionados. Na Figura 4-3, apresentamos alguns exemplos de associações entre objetos: (1) no domínio de vendas, um *cliente* compra *produtos*; (2) no domínio bancário, uma *conta corrente* possui um *histórico de transações*; (3) em um hotel, há vários *hóspedes*, assim como há vários *quartos*. Os *hóspedes* do hotel ocupam *quartos*.

Figura 4-3: Exemplos de associações entre objetos.



Associações possuem diversas características importantes: multiplicidades, nome, direção de leitura e papéis. Nas próximas seções, descrevemos essas características.

Multiplicidade

Uma associação representa um relacionamento estrutural existente entre objetos. Em muitas situações de modelagem, é importante determinar a quantidade de objetos que podem ser conectados pela instância de uma associação. Essa quantidade é chamada de multiplicidade (Booch, 2005). Cada associação em um diagrama de classes possui duas multiplicidades, uma em cada extremo da linha que a representa. Os símbolos possíveis para representar uma multiplicidade estão descritos na Tabela 4-1. Note que, em dois dos casos apresentados nessa Tabela, a UML define mais de um símbolo para representar a mesma multiplicidade. Primeiramente, o símbolo “1” é equivalente à utilização do símbolo “1..1”. Outra equivalência ocorre entre os símbolos “*” e “0..*” (Bezerra, 2007).

Tabela 4-1: Simbologia para representar multiplicidades

Nome	Simbologia
Apenas Um	1
Zero ou Muitos	0..*
Um ou Muitos	1..*
Zero ou Um	0..1
Intervalo Específico	li..ls

Como exemplo, observe a Figura 4-4, que exibe duas classes, `Pedido` e `Cliente`, e uma associação entre as mesmas. A leitura dessa associação nos informa que pode haver um objeto da classe `Cliente` que esteja associado a vários objetos da classe `Pedido` (isso é representado pela parte `*` do símbolo `0..*`). Além disso, essa mesma leitura nos informa que pode haver um objeto da classe `Cliente` que não esteja associado a pedido algum (isso é representado pela parte `0` do símbolo `0..*`). O diagrama da Figura 4-4 também representa a informação de que um objeto da classe `Pedido` está associado a um, e somente um, objeto da classe `Cliente`.

Figura 4-4: Exemplo de utilização dos símbolos de multiplicidade.



A Tabela 4-1 também exibe a forma geral do símbolo de multiplicidade para intervalos específicos. As expressões l_i e l_s devem ser substituídas por valores correspondentes aos limites inferior e superior, respectivamente, do intervalo que se quer representar. Por exemplo, vamos representar informações sobre velocistas e corridas nas quais eles participem. Suponha ainda que, em uma corrida, deve haver, no máximo, 6 velocistas participantes. O fragmento de diagrama de classes que representa esta situação é exibido na Figura 4-5. O valor para l_i é 2 (uma corrida está associada a, no mínimo, 2 velocistas), e o valor para l_s é 6 (uma corrida está associada a, no máximo, 6 velocistas).

Figura 4-5: Exemplo de utilização de intervalo de valores para multiplicidade.



Nome de Associação, Direção de Leitura e Papéis

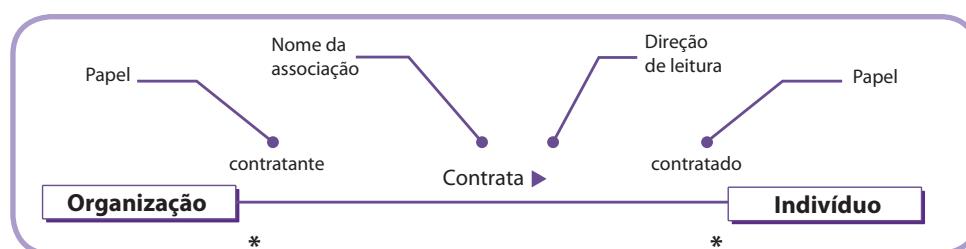
Para melhor esclarecer o significado de uma associação no diagrama de classes, a UML define três recursos de notação: *nome de associação*, *direção de leitura* e *papel*. O nome da associação é posicionado na linha da associação, a meio caminho das classes envolvidas.

A direção de leitura indica como a associação deve ser lida. Essa direção é representada por um pequeno triângulo posicionado próximo a um dos lados do nome da associação. O nome de uma associação deve fornecer algum significado semântico à mesma.

Quando um objeto participa de uma associação, ele tem um papel específico nela. Uma característica complementar à utilização de nomes e de direções de leitura é a indicação de *papéis* (*roles*) para cada uma das classes participantes em uma associação.

A Figura 4-6 apresenta uma associação na qual são representados os papéis (contratante e contratado) e o seu nome (*Contrata*). É também representada a direção de leitura, que indica que uma organização contrata indivíduos, e não o contrário (Bezerra, 2007).

Figura 4-6: Exemplo de utilização de nome de associação, direção de leitura e de papéis (Bezerra, 2007).



Sempre que o significado de uma associação não for fácil de inferir, é recomendável representar papéis ou pelo menos o nome da associação. Isso evita que haja interpretações erradas no significado da associação.

Embora não seja usual, pode haver diversos motivos pelos quais um objeto precisa saber sobre outro. Consequentemente, pode haver diversas associações definidas entre duas classes no diagrama de classes. Por exemplo, considere duas classes: Empregado e Departamento. Considere, ainda, que um departamento precisa saber quais são seus empregados e quem é o seu gerente. Nesse caso, há duas associações diferentes, embora as classes envolvidas sejam as mesmas (ver Figura 4-7).

Figura 4-7: Associações diferentes entre duas classes (Falbo, 2005).

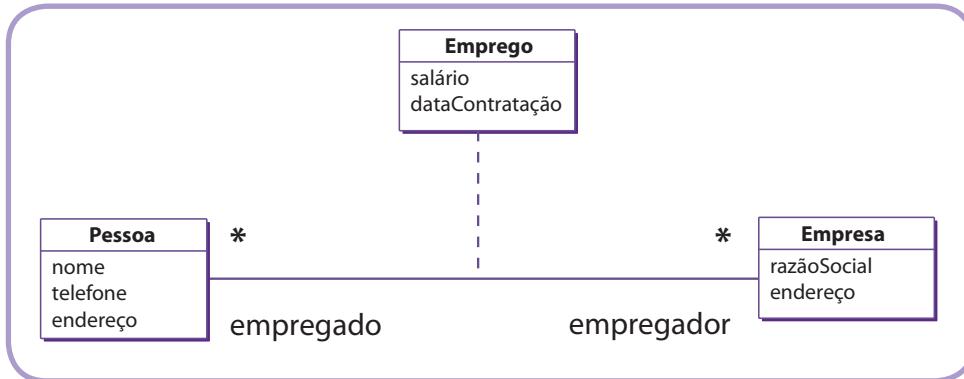


Classes Associativas

Classes associativas são classes que estão ligadas a associações, em vez de estarem ligadas a outras classes. São também chamadas de *classes de associação*. Esse tipo de classe normalmente aparece quando duas ou mais classes estão associadas, e é necessário manter informações sobre a associação existente entre as mesmas. Embora seja mais comum encontrar classes associativas ligadas a associações de conectividade *muitos para muitos*, uma classe associativa pode estar ligada a associações de qualquer conectividade.

Na UML, uma classe associativa é representada pela mesma notação utilizada para uma classe comum. A diferença é que esta classe é ligada por uma linha tracejada a uma associação. Para ilustrar a utilização de uma classe associativa em um diagrama de classes, apresentamos a Figura 4-8. Esse diagrama informa que uma pessoa trabalha como empregado em várias empresas. Uma empresa, por sua vez, tem vários empregados. A classe associativa Emprego permite saber, para cada par de objetos [empregado, empregador], qual o salário e a data de contratação do empregado em relação àquele empregador. De forma geral, se precisarmos representar a existência de uma informação que somente faz sentido quando considerarmos todos os objetos participantes da associação, podemos utilizar o conceito de classe associativa (Bezerra, 2007).

Figura 4-8: Exemplo de classe associativa (Larman, 2007).



Pelo exemplo anterior, pode-se notar que uma classe associativa é um elemento híbrido: tem características de uma classe, mas também características de uma associação.

Por fim, de forma geral, um diagrama de classes que contém uma classe associativa pode ser modificado para retirá-la, sem perda de informação no modelo em questão. Isso pode ser feito em dois passos: (1) eliminação da associação correspondente à classe associativa e (2) criação de associações diretas desta última com as classes que antes eram conectadas pela associação eliminada no passo 1. Como exemplo de aplicação desses passos, a Figura 4-9 apresenta um diagrama equivalente ao da Figura 4-8, em que a classe associativa foi substituída por uma classe ordinária. Note que a classe Emprego tem participação obrigatória em ambas as associações (Bezerra, 2007).

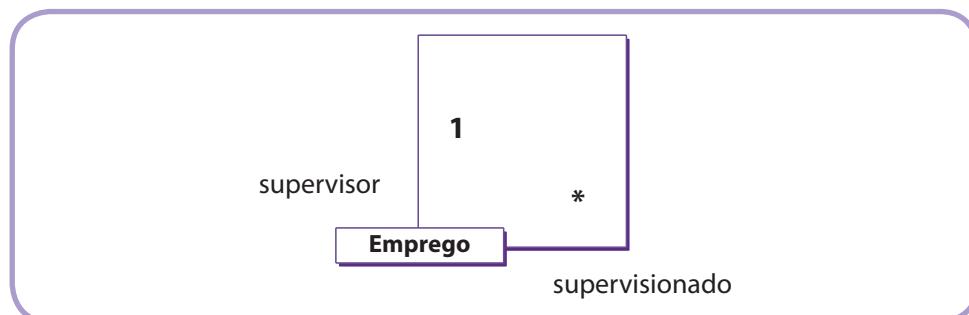
Figura 4-9: Uma classe associativa pode ser substituída por uma classe ordinária (Bezerra, 2007).



Associações Reflexivas

Uma associação reflexiva (também denominada *autoassociação*) associa objetos da mesma classe. Cada objeto tem um papel distinto nessa associação. Por exemplo, considere a Figura 4-10, em que existe uma associação reflexiva entre objetos de *Empregado*. Nessa figura, apresentamos um exemplo de uso de autoassociação, em que há objetos que assumem o papel de supervisor e outros objetos que assumem o papel de supervisionado. Nesse exemplo, o nome da associação poderia ter sido omitido, uma vez que os papéis foram definidos. Além disso, conforme mostra a Figura 4-10, em associações reflexivas, a utilização de papéis é bastante importante para evitar qualquer confusão na leitura da associação (Bezerra, 2007).

Figura 4-10: Exemplo de associação reflexiva (Bezerra, 2007).



Não se deve confundir o significado de uma associação reflexiva. Ela não indica que um objeto se associa a ele próprio (um empregado não é supervisor dele próprio; uma disciplina não é pré-requisito dela mesma, etc.). Em vez disso, uma autoassociação indica que um objeto de uma classe se associa com outros objetos da mesma classe.

Agregações e Composições

De todos os significados diferentes que uma associação pode ter, há uma categoria especial de significados, que representa relações *todo-parte*. Uma relação todo-parte entre dois objetos indica que um dos objetos está *contido* no outro. Podemos também dizer que um objeto *contém* o outro.

A UML define dois tipos de *relacionamentos todo-parte*, a *agregação* e a *composição*. A seguir, são relacionadas, de acordo com Bezerra (2007), algumas características particulares das agregações e composições que as diferem das associações simples.

- Agregações/composições são *assimétricas*, no sentido de que, se um objeto A é parte de um objeto B, o objeto B não pode ser parte do objeto A.
- Agregações/composições propagam comportamento, no sentido de que um comportamento que se aplica a um todo automaticamente se aplica às suas partes.
- Nas agregações/composições, as partes são normalmente criadas e destruídas pelo todo. Na classe do objeto todo, são definidas operações para adicionar e remover as partes.

Em uma situação prática, podemos aplicar a seguinte regra para verificar se faz sentido utilizarmos um relacionamento todo-parte (agregação ou composição). Sejam duas classes associadas, X e Y. Se uma das

perguntas a seguir for respondida com um sim, provavelmente há um relacionamento todo-parte envolvendo X e Y, no qual X é o todo, e Y é a parte. (Do contrário, é melhor utilizarmos a associação simples).

- 1) X tem um ou mais Y?
- 2) Y é parte de X?

Serafini (2008) difere agregação e composição:

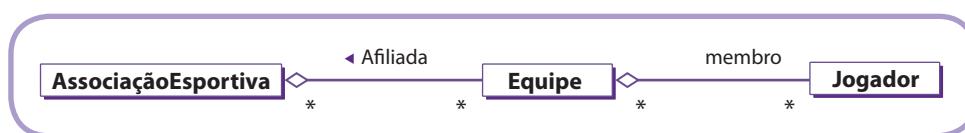
Agregação: ocorre quando usamos também um objeto “Todo” e outro objeto “parte”, porém a “vida” dos objetos é independente.

Composição: quando o objeto “Todo” é constituído pelo objeto “parte”, de tal forma que sem o objeto “Todo” o objeto “parte” não tem sentido.

Graficamente, a UML fornece notações diferentes para a agregação e a composição. Uma agregação é representada como uma linha que conecta as classes relacionadas, com um diamante (losango) branco perto da classe que representa o todo. Já uma composição é representada na UML por meio de um diamante negro, para contrapor com o diamante branco da agregação.

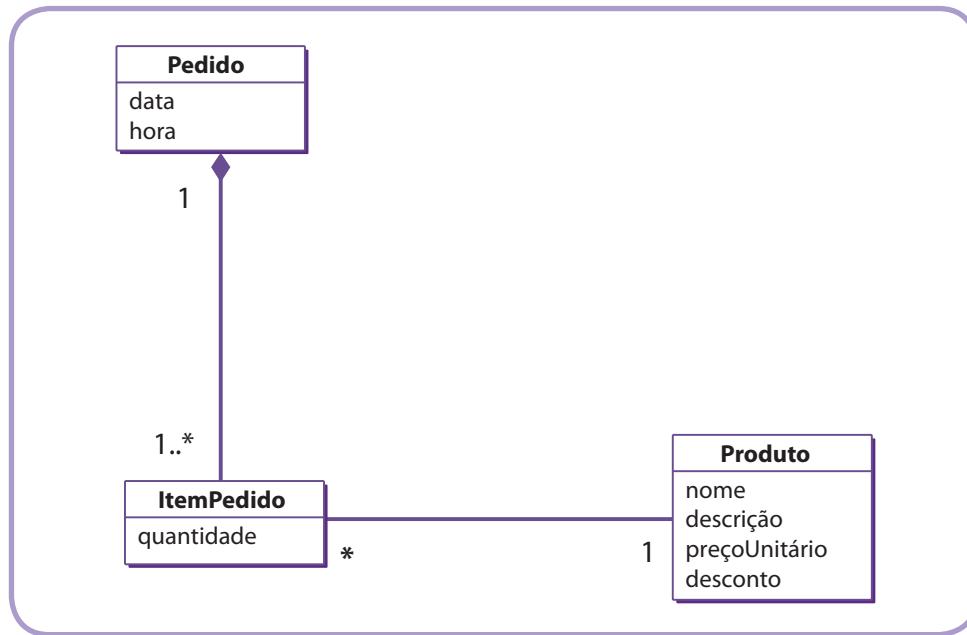
Como exemplo de agregação, analise a Figura 4-11, que apresenta um fragmento de diagrama de classes. Esse diagrama indica que uma associação esportiva é formada por diversas equipes, sendo que cada uma é formada por diversos jogadores. Por outro lado, um jogador pode fazer parte de diversas equipes.

Figura 4-11: Exemplo de agregação (Bezerra, 2007).



Como exemplo de composição, considere os itens de um pedido de compra. É comum um pedido de compra incluir vários itens, cada um relacionado a um produto faturado. Os itens têm identidade própria (é possível distinguir um item de outro no mesmo pedido). Essa situação é representada no diagrama da Figura 4-12.

Figura 4-12: Exemplo de composição (Bezerra, 2007).



4.1.3. Generalizações e Especializações

Além de relacionamentos entre objetos, o modelo de classes também pode representar relacionamentos entre classes. Esses últimos denotam relações de generalidade ou especificidade entre as classes envolvidas. Por exemplo, o conceito *mamífero* é mais genérico que o conceito *ser humano*. Outro exemplo: o conceito *carro* é mais específico que o conceito *veículo*.

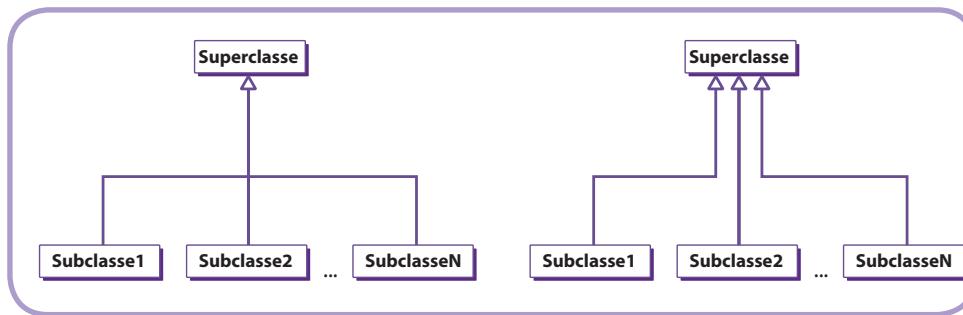
O relacionamento de herança é também chamado de relacionamento de *generalização/especialização*. Isso porque a generalização e a especialização são dois pontos de vista do mesmo relacionamento: dadas duas classes A e B, se A é uma generalização de B, então B é uma especialização de A.

De acordo com Booch (2005) uma generalização é um relacionamento entre itens gerais e tipos mais específicos desses itens.

Bezerra (2007) lembra que os termos para denotar o relacionamento de herança são bastante variados. O termo *subclasse* é utilizado para denotar a classe que herda as propriedades de outra classe através de uma generalização. Diz-se, ainda, que a classe que possui propriedades herdadas por outras classes é a *superclasse*. Outros termos (mais utilizados em linguagens de programação orientada a objetos) são *classe base* (sinônimo para superclasse) e *classe herdeira* (sinônimo para subclasse). Diz-se também que uma subclasse é uma *especialização* de sua superclasse (a subclasse especializa a superclasse), e que uma superclasse é uma *generalização* de suas subclasses (a superclasse generaliza as subclasses).

No diagrama de classes, a herança é representada na UML por uma flecha partindo da subclasse em direção à superclasse. Alternativamente, podemos juntar as flechas de todas as subclasses de uma classe, como mostra a Figura 4-13. A utilização de um ou de outro estilo é questão de gosto.

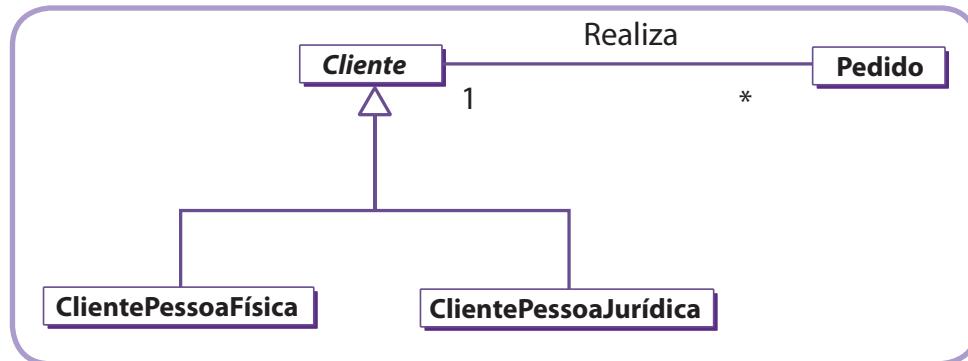
Figura 4-13: Representações alternativas para o relacionamento de generalização na UML (Bezerra, 2007).



Para entender a semântica de um relacionamento de herança, é preciso lembrar que uma classe representa um conjunto de objetos que partilham um conjunto comum de propriedades (atributos, operações, associações). No entanto, alguns objetos, embora bastante semelhantes a outros, podem possuir um conjunto de propriedades que outros não possuem. Por exemplo: todos os funcionários de uma empresa partilham os atributos nome, endereço, númeroMatrícula e salárioBase; no entanto, um tipo especial de funcionário - o dos vendedores - possui atributos específicos, por exemplo, zonaGeográfica (zona em que trabalham) e taxaComissão (porcentagem de comissão que recebem pelas vendas realizadas). Os vendedores pertencem à classe dos funcionários, mas eles por si próprios formam uma outra classe, a dos vendedores. Portanto, vendedores possuem todas as características inerentes a funcionários, além de possuírem características próprias. Diz-se, assim, que a classe de vendedores *herda* propriedades da classe de funcionários (Bezerra, 2007).

É importante notar que não somente atributos e operações são herdados pelas subclasses, mas também as associações que estão definidas na superclasse. A Figura 4-14 é um exemplo dessa propriedade: existe uma associação entre Cliente e Pedido. Não há necessidade de se criar uma associação entre Pedido e ClientePessoaFísica e entre Pedido e ClientePessoaJurídica, pois as subclasses herdam a associação de sua superclasse. Por outro lado, se existe uma associação que não é comum a todas as subclasses, mas, sim, particular a algumas subclasses, então essa associação deve ser representada em separado, envolvendo somente as subclasses em questão.

Figura 4-14: Não há necessidade de se criar uma associação entre Pedido e as sub-classes de Cliente (Bezerra, 2007).



4.2 TÉCNICAS PARA IDENTIFICAÇÃO DE CLASSES

4.2.1 Utilização de uma Taxonomia de Conceitos

Diversas abordagens podem ser aplicadas com o objetivo de identificar as classes de um SSOO. Um exemplo é a utilização de uma taxonomia de conceitos, conforme apresentado por Bezerra (2007):

- *Conceitos concretos*, como edifícios, carros, salas de aula, etc.
- *Papéis* desempenhados por seres humanos, como professores, alunos, empregados, clientes, etc.
- *Eventos*, ou seja, ocorrências em uma data e em uma hora particulares, como reuniões, pedidos, aterrissagens, aulas, etc.
- *Lugares*, ou seja, áreas reservadas para pessoas ou coisas, como escritórios, filiais, locais de pouso, salas de aula, etc.
- *Organizações*, ou seja, coleções de pessoas ou de recursos, como departamentos, projetos, campanhas, turmas, etc.
- *Conceitos abstratos*, isto é, princípios ou ideias não tangíveis, como reservas, vendas, inscrições, etc.

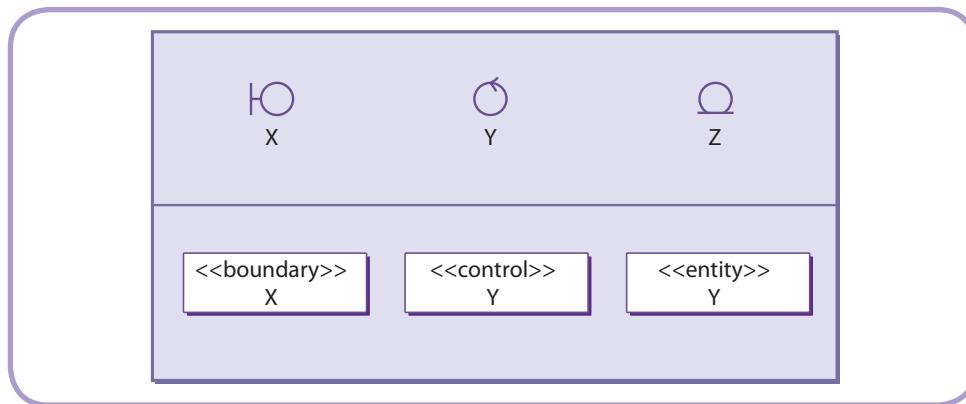
Além de estudar taxonomias de conceitos, é uma boa abordagem analisar também as funcionalidades e a documentação de sistemas similares no mesmo domínio do problema. O estudo do vocabulário dos especialistas do domínio é também uma boa fonte para identificação de classes.

4.2.2 Categorização BCE

Outra técnica de identificação é a *Análise de Robustez*. De acordo com essa técnica, os objetos que compõem um SSOO podem ser divididos em três categorias: *objetos de fronteira*, *objetos de controle* e *objetos de entidade*. Chamamos essa categorização de BCE (para lembrar os nomes originais das categorias: *boundary*, *control* e *entity*). Essa categorização fornece uma espécie de “arcabouço” que podemos tomar como ponto de partida para a identificação de classes em cada caso de uso de um sistema.

No contexto do diagrama de classes, a UML fornece estereótipos predefinidos, tanto textuais, quanto gráficos, para cada categoria BCE. A Figura 4-15 apresenta de forma esquemática os modos de representar objetos em cada uma dessas três categorias.

Figura 4-15: Notação da UML para objetos, segundo a categorização BCE.



Objetos de Fronteira

Objetos de fronteira realizam a comunicação do sistema com atores. Em outras palavras, são objetos de fronteira que permitem ao sistema *interagir com seu ambiente*.

Objetos de Controle

Um objeto de controle é também chamado de *controlador*. Objetos de controle servem como uma ponte de comunicação entre objetos de fronteira e objetos de entidade. São responsáveis por coordenar a execução de alguma funcionalidade específica do sistema.

Objetos de Entidade

Um objeto de entidade representa um conceito encontrado no *domínio do problema*. Normalmente servem como um repositório para alguma informação manipulada pelo sistema.

Categorização BCE na Identificação de Classes

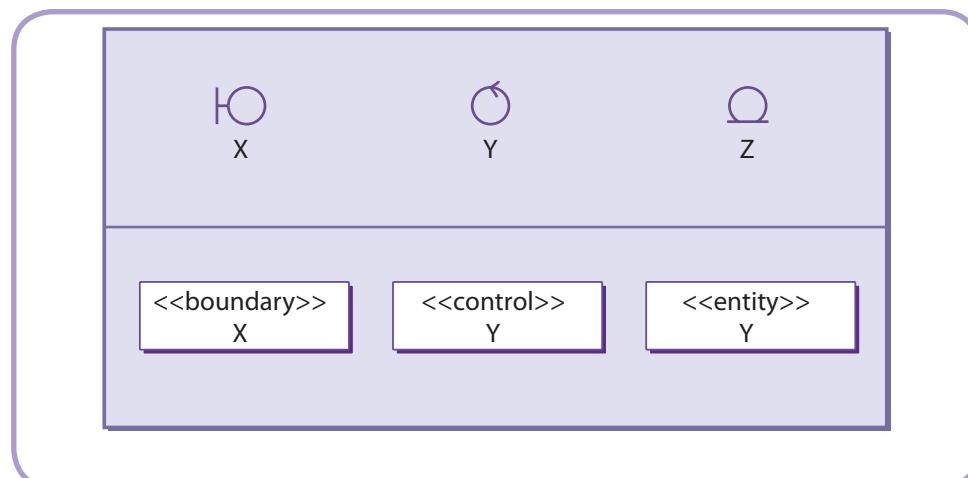
A categorização proposta por Jacobson implica que cada objeto é especialista em realizar um dos três tipos de tarefa, a saber: comunicar-se com atores (fronteira), manter as informações (entidade) ou coordenar a realização de um caso de uso (controle).

De acordo com Bezerra (2007), pelo exposto nas três Seções anteriores, podemos concluir que a categorização BCE funciona como uma espécie de "receita de bolo" para identificar objetos participantes da realização de um caso de uso. Em particular, essa técnica preconiza que, para cada caso de uso, as seguintes regras podem ser aplicadas na análise:

1. Adicionar um objeto de fronteira para cada ator participante do caso de uso. Dessa forma, as particularidades de comunicação com cada ator do caso de uso ficam encapsuladas no objeto de fronteira correspondente;
2. Adicionar um objeto de controle para o caso de uso. Isso porque um caso de uso representa um determinado processo do negócio. O controlador de um caso de uso tem então a atribuição de coordenar a realização desse processo do negócio.

A Figura 4-16 ilustra, de forma esquemática, o que acontece quando os objetos de um caso de uso de um SSOO são identificados conforme a categorização BCE. O ator se comunica com um objeto de fronteira, que repassa (pelo envio de mensagens) as ações desse ator para o objeto de controle (controlador). O controlador, por sua vez, coordena a colaboração de um ou mais objetos de entidade para a realização da tarefa desejada pelo ator. Note que, de acordo com o esquema dessa Figura, os objetos de entidade também se comunicam entre si com o envio de mensagem. Após a conclusão da tarefa, o objeto de controle repassa o resultado para um objeto de fronteira (que pode ser o mesmo no qual a realização começou, ou outro objeto). Este objeto de fronteira, por sua vez, apresenta o resultado para o ator.

Figura 4-16: A realização de um caso de uso envolve objetos de fronteira, de controle e de entidade (Bezerra, 2007).



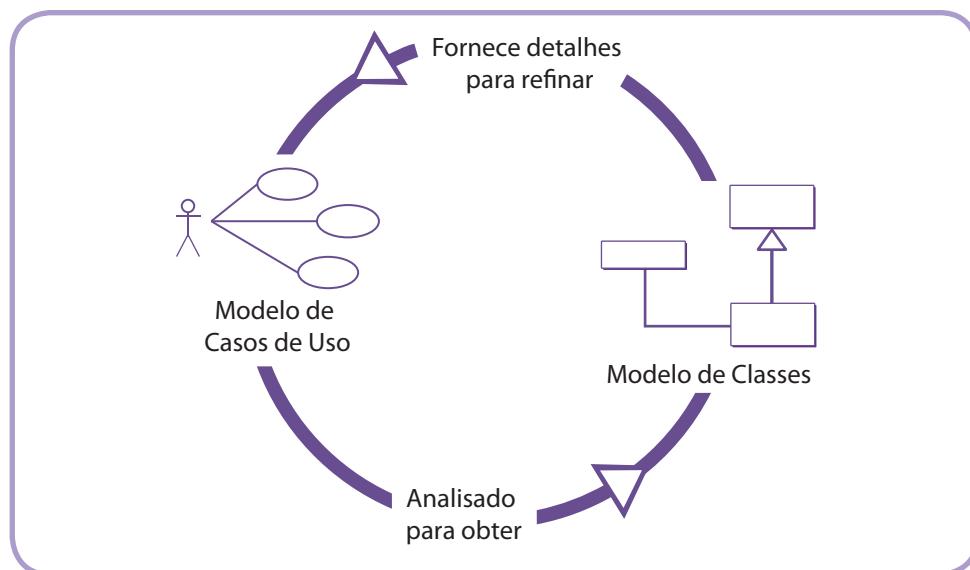
Classes de entidade, assim como seus atributos, são relativamente fáceis de identificar. Isso porque tipicamente correspondem a conceitos pertencentes ao domínio do problema. Essas classes são identificadas na *análise do domínio*. Já as classes de fronteira e de controle são normalmente identificadas na *análise da aplicação*.

4.3 MODELO DE CLASSES NO PROCESSO DE DESENVOLVIMENTO

Em um desenvolvimento *dirigido a casos de uso* (ver Seção 3.4), após a descrição dos casos de uso essenciais, é possível realizar a identificação de classes. Nesse ponto, uma ou mais das técnicas de identificação de classes (ver Seção 4.2) podem ser aplicadas. Durante a aplicação dessas técnicas, as classes identificadas são refinadas para retirar inconsistências e redundâncias. Finalmente, as classes são documentadas e o diagrama de classes inicial é construído, resultando no modelo de classes de análise.

A Figura 4-17 ilustra a interdependência entre a construção do modelo de casos de uso e o modelo de classes. De fato, esta é mais uma consequência do modo de pensar orientado a objetos: detalhes são adicionados aos poucos, à medida que o problema é entendido.

Figura 4-17: Interdependência entre o modelo de casos de uso e o modelo de classes (Bezerra, 2007).



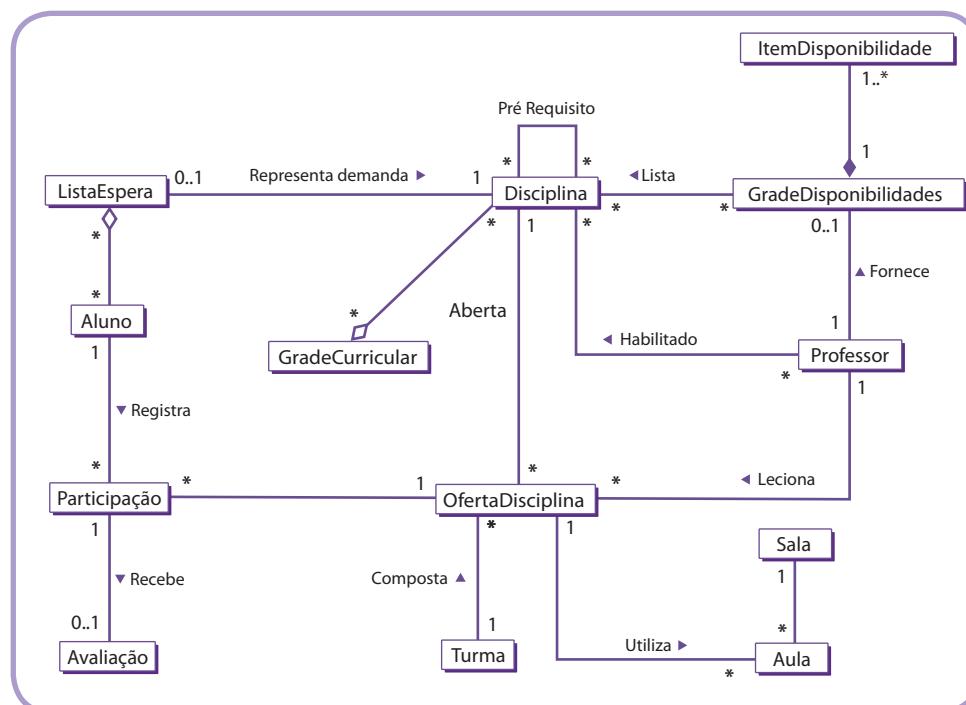
4.4 ESTUDO DE CASO

Esta seção continua o desenvolvimento da modelagem do estudo de caso proposto por Bezerra (2007) iniciado na Seção 3.5. Aqui, é apresentado o modelo de classes de domínio inicial desse estudo de caso.

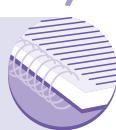
Note que os diagramas aqui apresentados não exibem atributos nem operações das classes. Embora o mapeamento de determinadas propriedades das classes já possa ser feito neste momento, essa tarefa é adiada para os capítulos seguintes, quando descrevemos o modelo de interações (Capítulo 5) e o modelo de classes de projeto (Capítulo 6). Isso porque é graças à construção do modelo de interações de um sistema que identificamos as operações que uma classe deve possuir.

A Figura 4-18 apresenta um diagrama de classes em que somente são apresentadas as classes do domínio do problema, ou seja, as classes de entidade. Entretanto, note que esse diagrama é somente a versão inicial do modelo de classes do SCA. Durante a fase de projeto, diversas outras classes surgem. Não que outro modelo de classes deva ser construído; em vez disso, o que acontece é que o *modelo de classes de análise* é estendido para ser transformado no *modelo de classes de projeto*.

Figura 4-18: Diagrama de classes de análise que apresenta as classes de entidade do SCA (Bezerra, 2007).



Atividades



1. Constura o modelo de classes de análise de um sistema de informações para controlar o campeonato de Fórmula 1.
2. Construa um diagrama de classes de análise para a seguinte situação: Pacotes são enviados de uma localidade a outra. Pacotes têm um peso específico. Localidades são caracterizadas pelas facilidades de transporte (por exemplo, ferroviárias, aeroportos e auto-estradas). Algumas localidades são vizinhas, isto é, existe uma rota direta de transporte entre tais localidades.

A rota de transporte entre as localidades tem um certo comprimento (a distância entre as localidades). Trens, aviões e caminhões são usados para o transporte de pacotes. Cada um destes meios de transporte pode suportar uma carga máxima de peso. A cada momento, durante o seu transporte, é necessário saber a posição (localidade) de cada pacote. Também é necessário manter o controle de que meio de transporte está sendo utilizado em cada parte da rota para um certo pacote.

MODELAGEM DE INTERAÇÕES

Em capítulos anteriores, dois modelos são descritos: o modelo de casos de uso e o modelo de classes de análise. Bezerra (2007) resume o que esses dois modelos fornecem de informação acerca do sistema.

O modelo de casos de uso descreve quais os requisitos funcionais do sistema e quais são as entidades do ambiente (atores) que interagem com o sistema. Este modelo nos informa também quais são as ações do sistema conforme percebidas pelos atores, e quais as ações do ator, conforme percebidas pelo sistema. Com esse modelo, podem ser respondidas questões sobre o que o sistema deve fazer e para quem. Todavia, o modelo de casos de uso nada informa sobre qual deve ser o comportamento interno do sistema para que uma determinada funcionalidade se realize. Ou seja, para que um caso de uso seja realizado, produzindo um resultado de valor para o ator, as questões a seguir são relevantes. (Note que, para essas perguntas, não encontramos respostas no modelo de casos de uso de um sistema, não importa quanto detalhado esse modelo seja).

1. Quais são as operações que devem ser executadas internamente ao sistema?
2. A que classes essas operações pertencem?
3. Quais objetos participam da realização desse caso de uso?

Por outro lado, o modelo de classes de análise fornece uma visão estrutural e estática inicial do sistema. A construção deste modelo resulta em um esboço das classes e de suas responsabilidades. No entanto, algumas questões também não são respondidas por esse modelo:

1. De que forma os objetos colaboram para que determinado caso de uso seja realizado?
2. Em que ordem as mensagens são enviadas durante essa realização?
3. Que informações precisam ser enviadas em uma mensagem de um objeto a outro?
4. Será que há responsabilidades ou mesmo classes que ainda não foram identificadas?

Verificando os parágrafos anteriores dá a entender que os modelos de casos de uso e de classes são representações incompletas do sistema. E realmente o são. Para responder às questões levantadas nos parágrafos anteriores, o modelo de interações do sistema precisa ser criado. Esse modelo representa as *mensagens* (ver a Seção 1.3.2) trocadas entre os objetos para a execução de cenários dos casos de uso do sistema. A modelagem de interações é uma parte da *modelagem dinâmica* de um SSOO.

Ainda para o mesmo autor, a construção do modelo de interações pode ser vista como uma consolidação do entendimento dos aspectos dinâmicos do sistema. Graças à construção do modelo de interações, as classes, as responsabilidades e os colaboradores identificados podem ser validados. Esse modelo permite ainda refinar o modelo de classes, pois as operações (e até alguns atributos) de cada classe são identificadas na construção do modelo de interações. Nesse capítulo, descrevemos os elementos do modelo de interações. São também apresentadas dicas sobre a construção desse modelo.

5.1 ELEMENTOS DA MODELAGEM DE INTERAÇÕES

A interação entre objetos para dar suporte à funcionalidade de um caso de uso denomina-se *realização de um caso*. A realização de um caso de uso descreve o comportamento de um ponto de vista *interno* ao sistema. A realização de um caso de uso é representada por diagramas de interação.

Os diagramas da UML que dão suporte à modelagem de interações são conhecidos genericamente como *diagramas de interação*. Bezerra (2007) lembra que até a versão 1.X da UML, havia dois diagramas para dar suporte à construção do modelo de interações: o *diagrama de sequência* (*sequence diagram*) e o *diagrama de comunicação* (*communication diagram*). A diferença entre esses dois tipos está na ênfase dada às interações entre os objetos. No diagrama de sequência, a ênfase está na ordem temporal das mensagens trocadas entre os objetos. Já o diagrama de comunicação enfatiza os relacionamentos que há entre os objetos que participam da realização de um cenário. O diagrama de sequência e o diagrama de comunicação são equivalentes entre si. No entanto, a popularidade e o uso do diagrama de sequência são bem maiores do que o diagrama de comunicação, por isso estudaremos mais detalhes o diagrama de sequência. A Figura 5-1 e a Figura 5-2 apresentam esboços das estruturas típicas do diagrama de sequência e do diagrama de comunicação, respectivamente.

Figura 5-1: Estrutura típica de um diagrama de sequência (Bezerra, 2007).

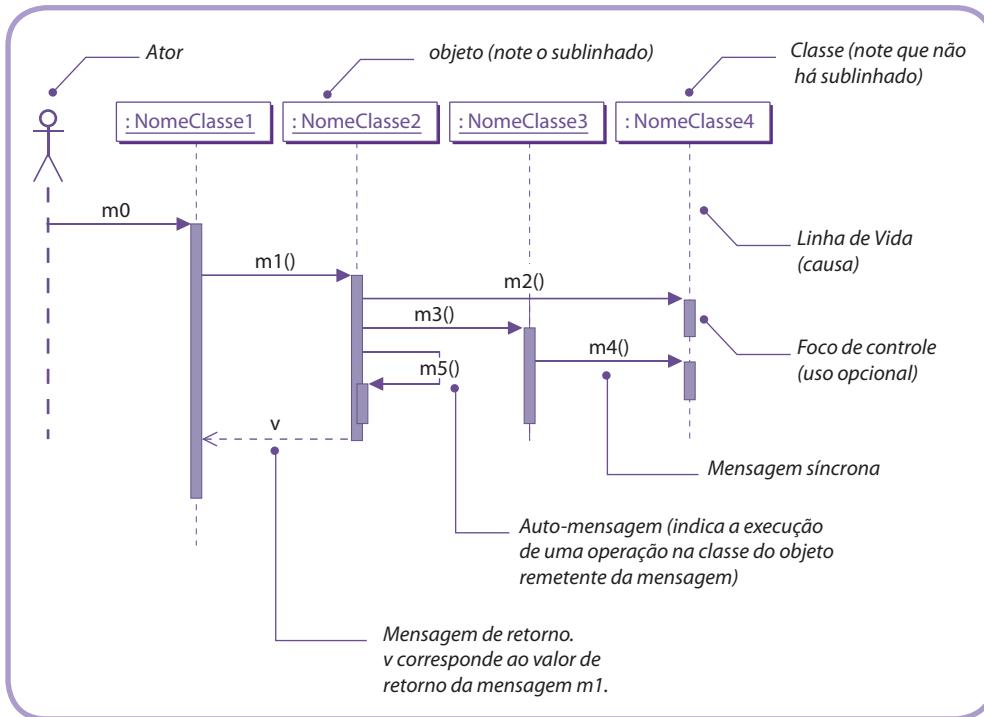
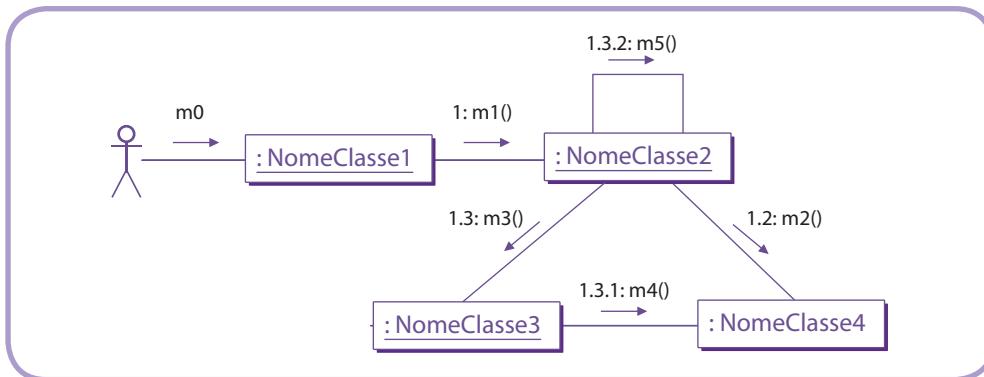


Figura 5-2: Estrutura típica de um diagrama de comunicação (Bezerra, 2007).



5.2 DIAGRAMA DE SEQUÊNCIA

Falbo (2005) afirma que um diagrama de sequência mostra a colaboração dinâmica entre os vários objetos de um sistema.

Assim como os outros diagramas da UML, o diagrama de sequência possui um conjunto de elementos gráficos. Na construção de um diagrama de sequência, podemos utilizar as notações gerais de modelagem de interações. Podemos também utilizar algumas notações particulares ao diagrama de sequência (ou seja, que não valem para o diagrama de comunicação). Algumas situações para as quais existem notações particulares no diagrama de sequência são as seguintes: linhas de vida, envio de mensagens, ocorrências de execução, criação e

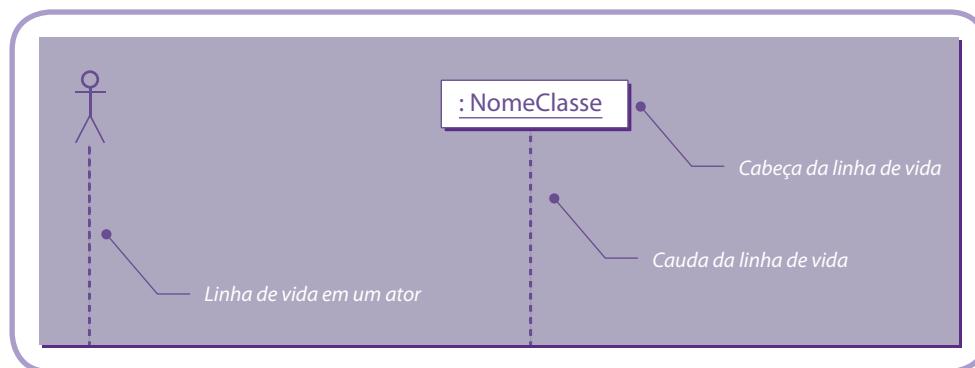
destruição de objetos. Descreveremos, de acordo com Bezerra (2007), a notação e o significado desses elementos de notação particulares nesta seção (descrevemos dicas para a construção propriamente dita de um diagrama de sequência na Seção 5.3).

5.2.1 Linhas de Vida

De acordo com Booch (2006) a linha de vida do objeto é a linha tracejada vertical que representa a existência de um objeto em um período de tempo. Uma linha de vida é composta de duas partes, a *cabeça* e a *cauda*.

A representação gráfica de uma linha de vida também possui uma cauda, que corresponde a uma linha vertical tracejada, conforme esquematizado na Figura 5-3. Para o caso de atores representados em um diagrama de sequência, também é comum pendurar uma linha tracejada nesses elementos, da qual partem as mensagens para o interior do sistema. Em um diagrama de sequência, normalmente o ator é posicionado na extrema esquerda do diagrama de sequência (ver também a Figura 5-1).

Figura 5-3: Representação de linhas de vida em um diagrama de sequência (Bezerra, 2007).



A ordem horizontal utilizada para posicionar os objetos no diagrama de sequência não tem nenhum significado predefinido. Entretanto, a ordem normalmente utilizada é a seguinte (da esquerda para a direita): ator primário, objeto(s) de fronteira com o(s) qual(is) o ator interage, objeto(s) de controle, objetos de entidade e atores secundários. Essa ordem de disposição facilita a leitura do diagrama.

5.2.2 Mensagens

A notação para uma mensagem em um diagrama de sequência é uma flecha (geralmente desenhada na horizontal) ligando uma linha de vida a outra. O objeto do qual parte a seta é aquele que está enviando a mensagem (objeto remetente). O objeto para o qual a seta aponta é aquele que está recebendo a mensagem (objeto receptor). O formato da

“ponta” da seta indica o tipo de mensagem sendo enviada. Os formatos possíveis em um diagrama de sequência estão ilustrados na Figura 5-4. O *rótulo da mensagem* é posicionado acima dessa linha.

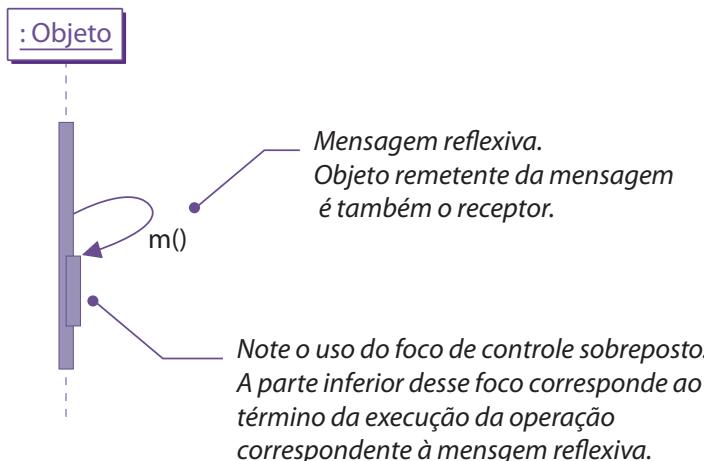
Figura 5-4: Notações para os diversos tipos de mensagem em um diagrama de sequência (Bezerra, 2007).

	Mensagem síncrona
	Mensagem assíncrona
	Mensagem de retorno
	Mensagem de criação de objeto

A passagem do tempo é percebida no diagrama de sequência observando-se a direção vertical no sentido de cima para baixo. Quanto mais abaixo uma mensagem aparece no diagrama, mais tarde no tempo esta mensagem foi enviada.

A Figura 5-5 ilustra o uso de uma *mensagem reflexiva* em um diagrama de sequência. Pode-se notar que uma mensagem reflexiva é representada por uma seta saindo e retomando para o próprio objeto.

Figura 5-5: Mensagem reflexiva em um diagrama de sequência (Bezerra, 2007).



5.2.3 Ocorrências de Execução

O mesmo autor, referência para a explicação destas seções, explica que uma ocorrência de execução representa o tempo em que o objeto está ativo, ou seja, o tempo em que ele realiza alguma operação. Graficamente, ocorrências de execução correspondem a blocos

retangulares posicionados sobre a linha de vida de um objeto. O topo de uma ocorrência de execução coincide, no receptor, com o recebimento de uma mensagem. A parte de baixo dessa ocorrência de execução coincide com o término de uma operação realizada pelo objeto.

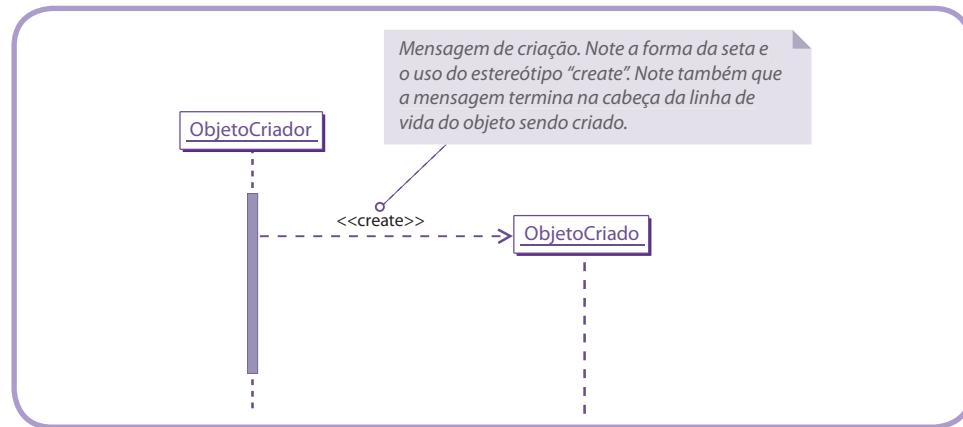
O modelador pode optar por não utilizar ocorrências de execução nos objetos presentes em certo diagrama de sequência. Por outro lado, o uso de ocorrências de execução muitas vezes torna desnecessário o uso de mensagens de retorno. Isso porque o final de uma ocorrência de execução corresponde ao retorno do fluxo de controle para o emissor da mensagem (no caso de mensagens síncronas).

5.2.4 Criação e Destrução de Objetos

Duas operações frequentes em um SSOO são a criação e destruição de objetos. A criação de um objeto é o momento em que esse objeto passa a existir no sistema, e passa a realizar suas responsabilidades. Por exemplo, em nosso estudo de caso, a realização do caso de uso Abrir Turma tem o potencial de criar objetos da classe ListaEspera, conforme vimos na Seção 4.4. Já a destruição de um objeto é o momento no qual este objeto deixa de existir no sistema.

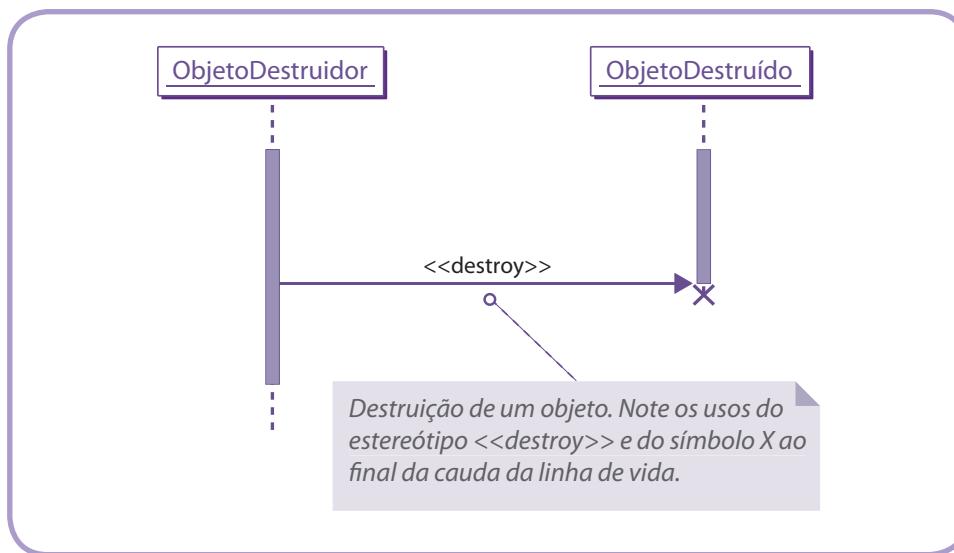
Se o objeto existe desde o início da interação (ou seja, se ele não é criado durante a interação sendo modelada), ele deve ser posicionado no topo do diagrama. No entanto, pode ser que a própria interação crie um ou mais objetos de certa classe. Nesse caso, a posição *vertical* de um objeto em um diagrama de sequência indica o momento no qual ele é criado (instanciado) e começa a participar da interação. Ou seja, o retângulo que representa o objeto deve ser posicionado mais abaixo no diagrama. A instanciação de um objeto é sempre requisitada por outro objeto participante da interação através de uma mensagem. A mensagem de criação pode ser rotulada de duas maneiras alternativas: (1) com o estereótipo «create» ou (2) com o nome do método construtor que será ativado. Além disso, a flecha da mensagem é pontilhada e aponta para o objeto em vez de para a linha de vida. Na Figura 5-6 apresentamos um exemplo da notação de criação de objetos (Bezerra, 2007).

Figura 5-6: Criação de objeto em um diagrama de sequência (Bezerra, 2007).



Um diagrama de sequência pode mostrar também a destruição de objetos no decorrer de uma interação. Um objeto normalmente é destruído quando ele não é mais necessário na interação. O símbolo X na parte de baixo da linha de vida de um objeto significa que ele está sendo destruído. Além disso, a mensagem de destruição é rotulada com o estereótipo «destroy». A Figura 5-7 fornece um exemplo da notação para destruição de objetos.

Figura 5-7: Destrução de objeto em um diagrama de sequência (Bezerra, 2007).



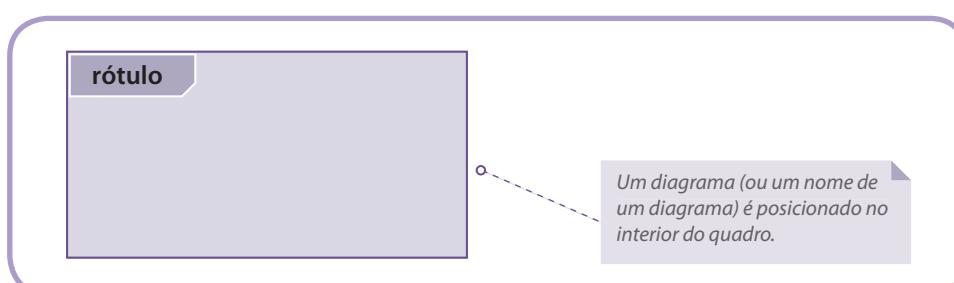
5.2.5 Modularização de Interações

A versão UML 2.0 introduziu diversos elementos gráficos novos para construção modular de diagramas de interação: quadros de interação, fragmentos combinados, referências, operadores, etc. Nesta seção, descrevemos esses elementos.

Quadros

Um *quadro de interação* (tradução para *interaction frame*) serve para encapsular um diagrama de sequência. Um quadro fornece um contexto ou uma fronteira no interior do qual podemos posicionar os elementos de um diagrama de sequência, tais como linhas de vidas e mensagens. Graficamente, um *quadro* é um retângulo. A notação gráfica para um quadro definida pela UML 2.0 é apresentada na Figura 5-8.

Figura 5-8: Notação para um quadro na UML (Bezerra, 2007).



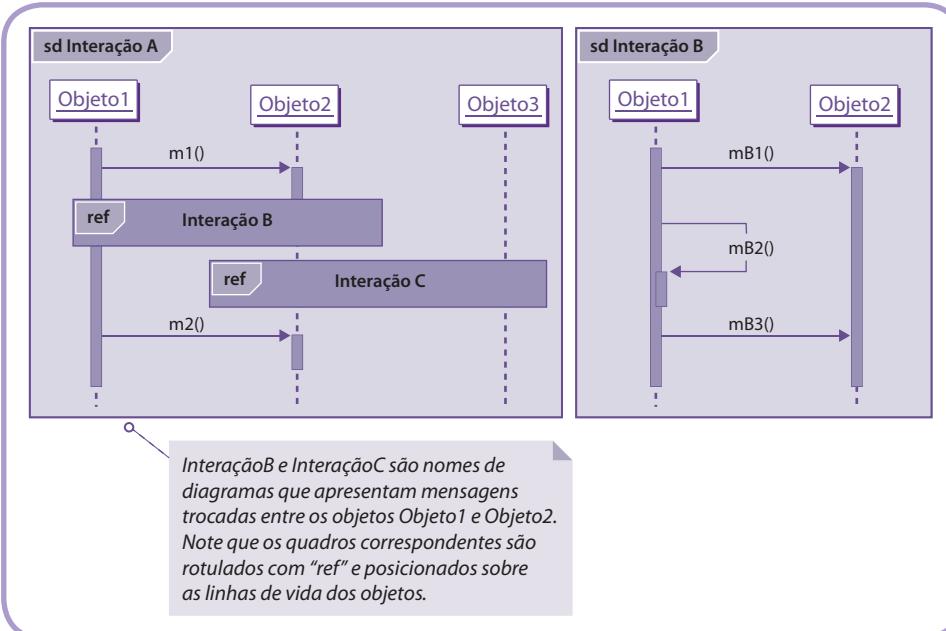
Conforme podemos perceber na Figura 5-8, na parte superior esquerda de um quadro, há uma espécie de “orelha”, na forma de um pentágono. No interior desse pentágono devemos definir um rótulo. Esse rótulo pode ser utilizado com três diferentes objetivos pelo modelador. Enumeramos esses objetivos abaixo. Nos próximos parágrafos desta seção, damos detalhes acerca de cada um desses objetivos.

- Para dar um nome ao diagrama que aparece dentro do quadro.
- Para fazer referência a um diagrama definido separadamente.
- Para definir o fluxo de controle da interação.

Bezerra (2007) detalha o primeiro objetivo (*dar um nome ao diagrama dentro do quadro*). A partir da UML 2.0, todo diagrama de interação pode ser posicionado dentro de um quadro. Se isso acontecer, a orelha do quadro deve conter uma expressão da forma `TipoDiagrama NomeDiagrama`. Nessa expressão, o elemento `NomeDiagrama` é o nome dado ao diagrama cuja interação está definida dentro do quadro. Qualquer diagrama de interação pode ser posicionado dentro de um quadro. Para identificar qual o tipo de diagrama que está contido no quadro, utilizamos o elemento `TipoDiagrama` da expressão. Os valores possíveis para este elemento são os seguintes: `sd` (para diagramas de sequência), `comm` (para diagramas de comunicação) e `activity` (para diagramas de atividade). O objetivo de dar um nome a um diagrama de interação é permitir que este seja referenciado durante a definição de outros diagramas.

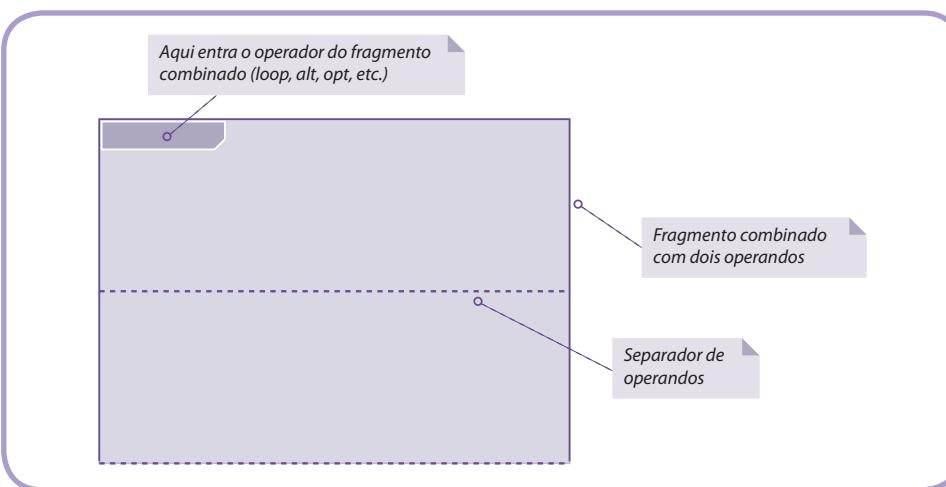
O segundo objetivo da orelha de um quadro é *fazer referência a um diagrama definido separadamente*. De fato, além de permitir atribuir um nome a um diagrama, a orelha de um quadro pode também servir para indicar que este corresponde a uma *ocorrência de interação*. Uma *ocorrência de interação* é uma interação que é referenciada por outra. Podemos utilizar ocorrências de interação para fatorar interações comuns a vários diagramas de sequência em um único quadro e reutilizar esse quadro diversas vezes. Graficamente, uma ocorrência de interação é um quadro rotulado com a palavra-chave `ref`. O nome da interação é posicionado no interior desse quadro rotulado com a palavra-chave `ref` à sua esquerda. Essa palavra-chave indica que uma interação está sendo referenciada dentro de outra interação. Na Figura 5-9, observa-se que o quadro rotulado como `InteraçãoA` faz referência a duas outras interações, que são representadas como ocorrências de interação, e estão indicadas no diagrama como `InteraçãoB` e `InteraçãoC`.

Figura 5-9: Ocorrências de interação (Bezerra, 2007).



A ocorrência de interação implica que um quadro pode conter outros quadros em seu interior, ou seja, *subquadros*. Na verdade, a UML 2.0 define dois tipos de subquadros. Um deles é a ocorrência de interação, à qual nos referimos há pouco. O outro tipo corresponde ao chamado *fragmento combinado*. Este elemento é utilizado para alcançar o terceiro objetivo enumerado anteriormente, *definir o fluxo de controle da interação*. Este elemento fornece ao modelador a capacidade de definir lógica procedural (fluxo de controle) nos seus diagramas de interação. Graficamente, um fragmento combinado corresponde a uma ou mais interações (sequências de mensagens) encapsuladas em um quadro. O operador da interação é representado dentro do pentágono do quadro correspondente ao fragmento combinado. Os operandos do fragmento combinado (que são as interações) são posicionados dentro do quadro. Se houver mais de um operando, estes são separados por uma linha tracejada. Veja a notação definida pela UML 2.0 para um fragmento combinado na Figura 5-10.

Figura 5-10: Notação da UML para um fragmento combinado (Bezerra, 2007).



Bezerra (2007) explica que, internamente, um fragmento combinado é composto de um ou mais *operandos da interação*, de zero ou mais *condições de guarda*, de um *operador de interação*. Um *operando* em um fragmento combinado corresponde a uma sequência de mensagens. Essa sequência é executada (ou seja, as mensagens correspondentes são enviadas) somente sob circunstâncias específicas. A forma de execução dessa sequência é definida pelo *operador de interação* e pelas condições de guarda que são utilizadas. Uma *condição de guarda*, também chamada de *restrição da interação*, é uma expressão condicional (de valor verdadeiro ou falso) que é associada a um operando da interação em um fragmento combinado. Uma condição de guarda impõe uma restrição acerca da execução do operando ao qual está associada. Essa restrição deve ser suficiente para definir se o operando (interação) correspondente deve ou não ser executado. Se a condição tiver valor verdadeiro, o operando é executado; do contrário, o operando não é executado. A definição de uma condição de guarda é opcional, de tal forma que um operando sem condição de guarda é executado sempre. Um *operador de interação* define a semântica de um fragmento combinado e determina como usar os operandos da interação contidos nesse fragmento combinado. Alguns desses operadores de interação definidos na UML 2.0 são os seguintes: *alt*, *opt*, e *loop*. Descrevemos seus significados a seguir.

O operador *alt* modela construções procedimentais do tipo se-então-senão. O corpo (operando) do operador de controle é dividido em várias sub-regiões por linhas horizontais tracejadas. Cada sub-região tem uma condição de guarda (Booch, 2005). Veja um exemplo na Figura 5-11.

O operador *opt* modela construção procedural do tipo se... então. O corpo (operando) do operador de controle é executado se uma condição de guarda for verdadeira (Booch, 2005). Veja um exemplo na Figura 5-12.

Figura 5-11: Utilização do operador de interação alt (Bezerra, 2007).

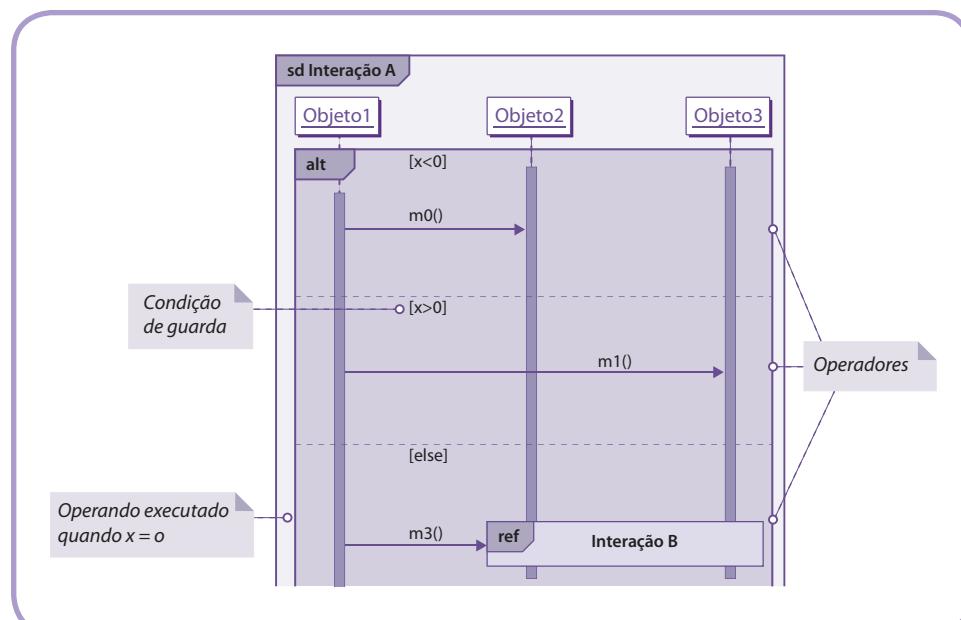
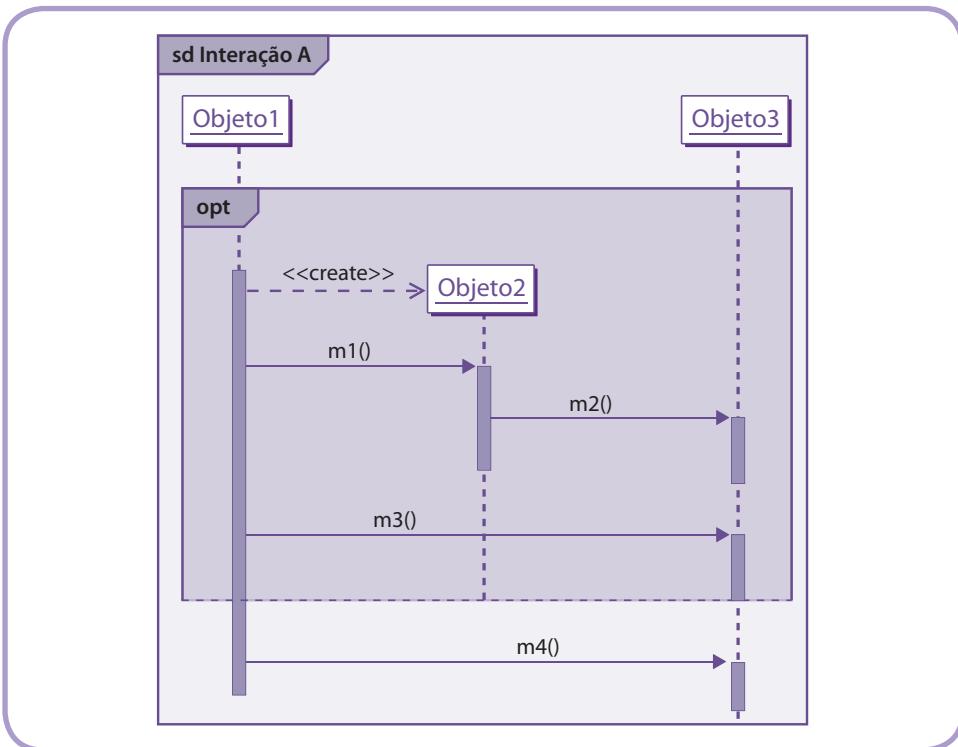


Figura 5-12: Utilização do operador de interação opt (Bezerra, 2007).



O operador *loop* serve para representar que uma interação (operando) deve ser realizada zero ou mais vezes. Após o nome do operador, é possível (opcional) representar os limites mínimo e máximo para definir a quantidade de iterações. A sintaxe do operador *loop* é a seguinte: *loop* (<minint>, <maxint>).

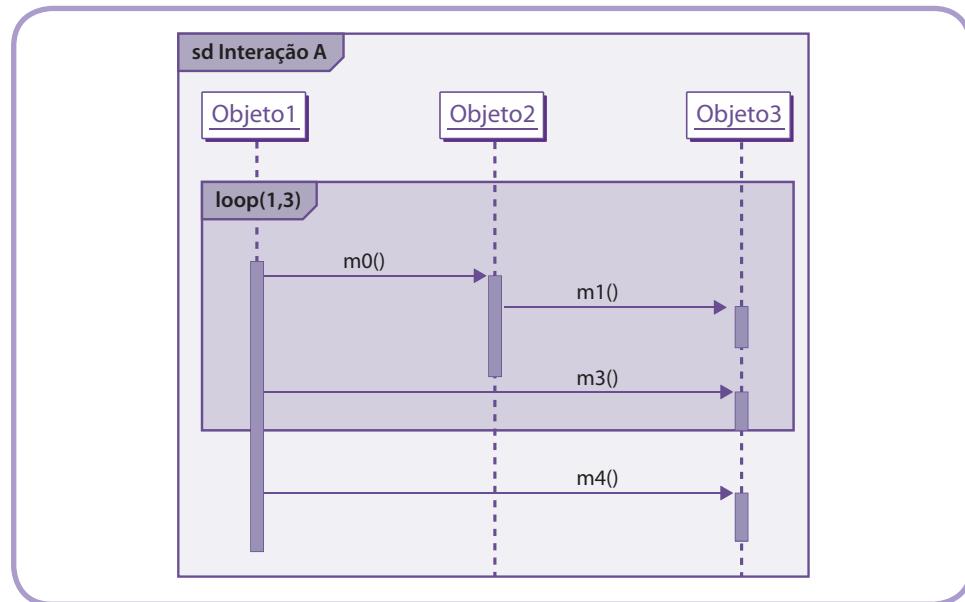
O corpo (operando) do *loop* é executado repetidamente enquanto a condição de guarda for verdadeira, antes de cada iteração (Booch, 2005).

A Figura 5-13 fornece um exemplo de diagrama de sequência no qual essa sintaxe é utilizada. Nessa sintaxe, temos:

1. O limite mínimo <minint> é um número inteiro não negativo.
2. O limite máximo <maxint> pode ser representado de duas maneiras: (1) com um número inteiro maior ou igual a <minint>; (2) pelo símbolo '*', que significa uma quantidade de iterações sem limite superior.

Na sintaxe do operador *loop*, se somente o componente <minint> é definido, isso significa que <minint> é igual a <maxint>. Por outro lado, se somente é utilizado o operador (sem os limites mínimo e máximo), isso significa uma quantidade de iterações com limite inferior ou igual a zero e sem limite superior.

Figura 5-13: Utilização do operador de interação *loop* (Bezerra, 2007).



5.3 PROCEDIMENTO DE CONSTRUÇÃO DE UM DIAGRAMA DE INTERAÇÃO

A seguir, baseado em Bezerra (2007), descrevemos um procedimento para construir diagramas de interação. Antes de descrevermos esse procedimento, entretanto, é necessário que definamos o conceito de *evento de sistema*, o que fazemos no próximo parágrafo.

Um evento de sistema é alguma ocorrência no ambiente do sistema e para o qual este sistema deve realizar alguma ação quando da ocorrência do evento. No contexto de casos de uso, eventos de sistema correspondem às *ações do ator* no cenário de determinado caso de uso. Sendo assim, é relativamente fácil identificar eventos de sistemas em uma descrição de caso de uso: devemos procurar nessa descrição os eventos que correspondem a ações do ator. No caso particular em que o ator é um ser humano e existe uma interface gráfica para que o mesmo interaja com o sistema, os eventos do sistema são resultantes de ações desse ator sobre essa interface gráfica, que corresponde a objetos de fronteira.

Apresentamos agora os passos do procedimento para construção dos diagramas de interação.

1. Para cada caso de uso, selecione um conjunto de cenários relevantes. Certamente, o cenário correspondente ao fluxo principal do caso de uso deve ser incluído nessa seleção. Considere também fluxos alternativos e de exceção que tenham potencial em demandar responsabilidades de uma ou mais classes.

2. Para cada cenário selecionado no passo anterior, identifique os *eventos de sistema*:
 - a) Posicione o(s) ator(es), objeto de fronteira e objeto de controle no diagrama;
 - b) Para cada passo do cenário selecionado, defina as mensagens a serem enviadas de um objeto a outro;
 - c) Defina as cláusulas de condição e de iteração, se existirem, para as mensagens;
 - d) Adicione multiobjetos e objetos de entidade à medida que a sua participação for necessária no cenário selecionado.

Um ponto importante na produção de diagramas de interação é acerca de como representar as requisições que chegam a um objeto de fronteira. Essas requisições normalmente são rotuladas com a informação fornecida pelo ator (por exemplo, *nome da disciplina*, *matrícula*, etc.). Essas informações são posteriormente repassadas do objeto de fronteira para o objeto de controle através de parâmetros de mensagens.

É também importante notar que, de uma forma geral, as mensagens enviadas pelo objeto de fronteira como consequência do acontecimento de algum evento de sistema resultam na definição das operações no objeto controlador do caso de uso em questão.

As operações mencionadas anteriormente são exemplos de *operações de sistema*. Uma operação de sistema é qualquer operação que deve ser executada para “recepçionar” um evento de sistema. Uma operação de sistema normalmente é alocada a um objeto controlador. Obviamente, diversas outras operações são invocadas como consequência da chamada de uma operação de sistema. Ou seja, uma vez que uma operação de sistema é identificada, devemos também identificar as operações que devem ser invocadas subsequentemente para que a tarefa correspondente seja executada pelos objetos do sistema. A identificação correta dessas operações subsequentes e sua correta alocação aos objetos do sistema não são tarefas fáceis. Entretanto, pelo menos o modelador tem um ponto de partida para começar essa identificação: os eventos do sistema identificados anteriormente. Uma vez identificados esses eventos, a definição das operações de sistema correspondentes é um processo relativamente fácil e mecânico: dado um evento de sistema, definimos uma operação de sistema correspondente (ou seja, uma mensagem enviada ao objeto de controle) e definimos nessa operação *parâmetros* (ver Seção 6.3.1) que permitam que as informações necessárias à

execução da operação sejam passadas de um objeto a outro. A partir desse ponto, a identificação das operações (ou equivalentemente, das mensagens) subsequentes é uma tarefa complicada, cuja correta realização depende muito da experiência do modelador.

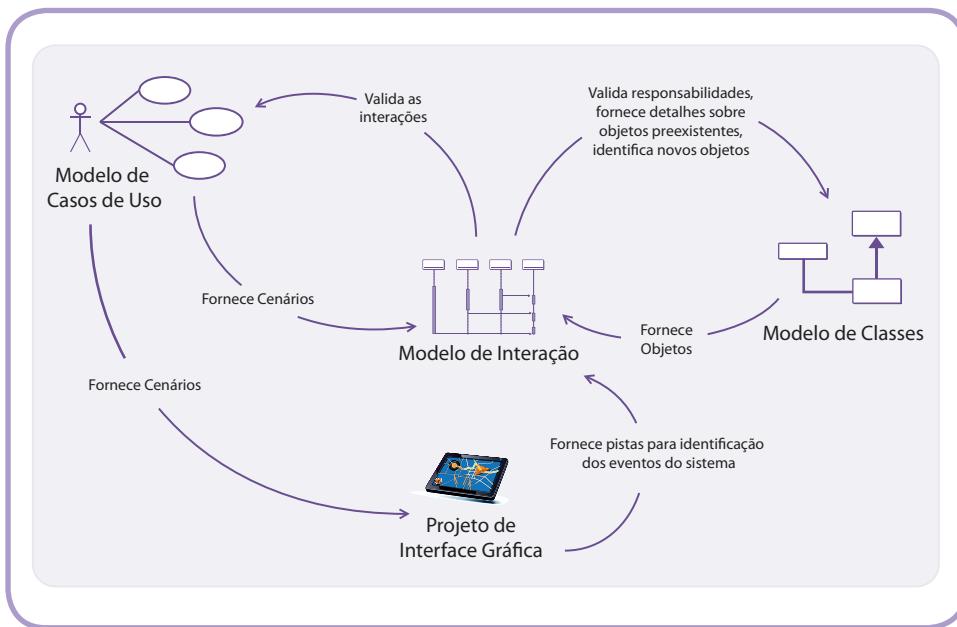
5.4 MODELO DE INTERAÇÕES NO PROCESSO DE DESENVOLVIMENTO

Bezerra (2007) levanta uma questão relevante acerca de qual é o momento adequado para começar a construir o modelo de interações. Segundo o autor, alguns textos sobre modelagem de sistemas orientados a objetos indicam o início da modelagem de interações já na atividade de análise. Outros defendem o início de sua construção somente na atividade de projeto. O fato é que a distinção entre essas duas fases não é tão nítida na modelagem orientada a objetos, e diagramas de interação podem ser utilizados em ambas as fases. Na análise, podemos começar a construir os diagramas de interação logo depois que uma primeira versão do diagrama de casos de uso estiver pronta. Inicialmente, o diagrama de interação pode ser utilizado para representar apenas os objetos participantes e com mensagens exibindo somente o nome da operação. Posteriormente (na fase de projeto), esse diagrama pode ser refinado com mais detalhes, incluindo criação e destruição de objetos, detalhes sobre o tipo e assinatura completa de cada mensagem.

Os objetivos da construção de diagramas de interação são (1) obter informações adicionais para completar e aprimorar outros modelos (modelo de casos de uso e modelo de classes) e (2) fornecer aos programadores uma visão detalhada dos objetos e mensagens envolvidos na realização dos casos de uso do sistema.

A Figura 5-14 enfatiza a interdependência entre os artefatos de software produzidos pelos três modelos mencionados há pouco. As setas indicam que as atividades de modelagem alimentam umas as outras com informações para que cada modelo evolua.

Figura 5-14: Interdependência entre os artefatos produzidos durante o desenvolvimento (Bezerra, 2007).



5.5 ESTUDO DE CASO

Continuando o desenvolvimento da modelagem do Sistema de Controle Acadêmico, proposto por Bezerra (2007), esta seção apresenta uma parte do modelo de interações para este sistema. A Figura 5-15 ilustra um diagrama de sequência para um dos eventos de sistema do fluxo principal do caso de uso Realizar Inscrição (a descrição deste caso de uso se encontra na Seção 3.5.3).

De acordo com a Figura 5-15, podemos perceber que no passo 2 desse caso de uso em questão, o sistema deve apresentar uma lista de disciplinas para que o aluno selecione aquelas nas quais deseja se inscrever no próximo semestre letivo.

"2. O sistema apresenta as disciplinas para as quais o aluno tem pré-requisito (conforme a RN03, excetuando-se as que já tinha cursado.)"

No entanto, essa lista de disciplinas não deve conter as que o aluno já cursou; também não deve conter as disciplinas para as quais o aluno não tem pré-requisitos. Para formar essa lista e repassar ao objeto de fronteira, o objeto controlador coordena a colaboração de diversos objetos de entidade.

Ainda no diagrama apresentado na Figura 5-15, podemos perceber o padrão de comunicação entre objetos descritos na Seção 4.2.2. O objeto de fronteira **FormulárioInscrição** se comunica com o ator **Aluno**.

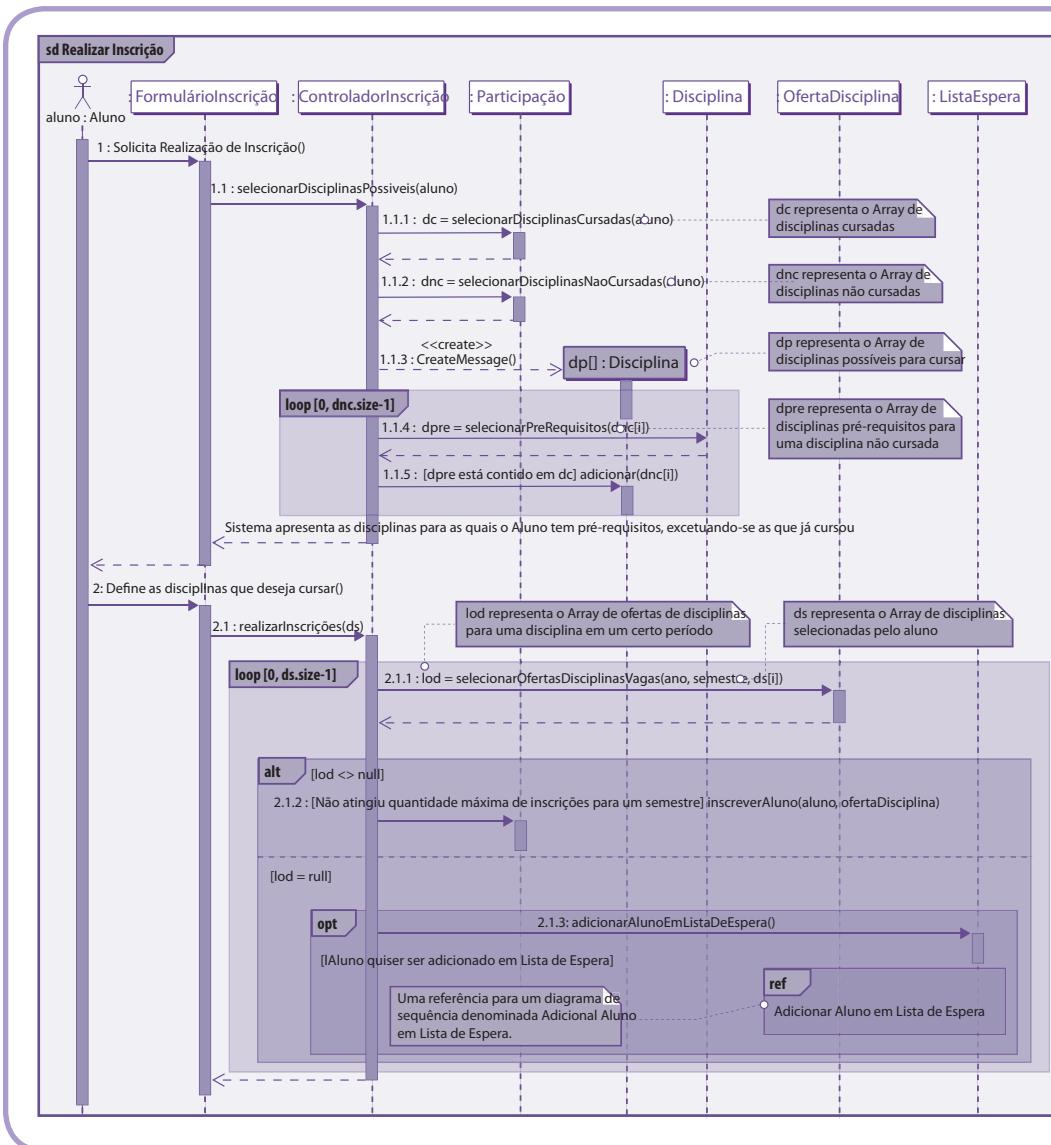
Esse mesmo objeto de fronteira também se comunica com o objeto de controle ControladorInscrição. Este, por sua vez, se comunica com os objetos de entidade envolvidos no cenário.

Esse diagrama representa ainda a troca de mensagens resultante da realização do passo 4 do caso de uso. Nesse passo, o controlador coordena o processo de inscrição: para uma disciplina desejada pelo aluno, o controlador percorre a coleção de ofertas existentes com o objetivo de encontrar uma oferta em que haja vagas para inscrever o aluno. Finalmente, quando essa oferta de disciplina é encontrada, o controlador solicita que o objeto Aluno se inscreva na mesma.

"4. Para cada disciplina selecionada, o sistema designa o aluno para uma turma que apresente uma oferta para tal disciplina."

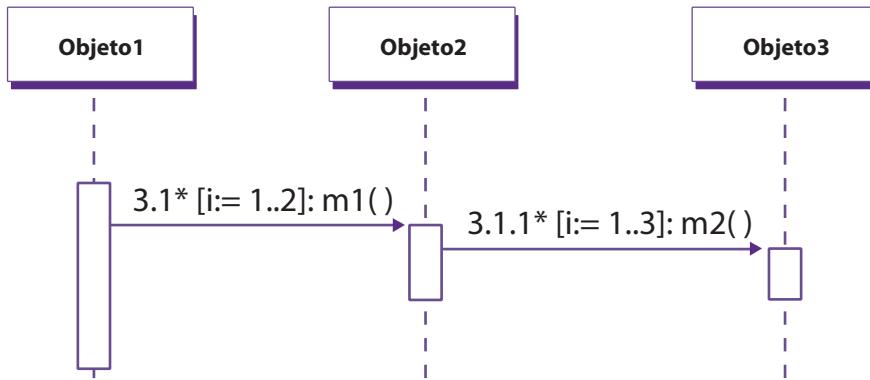
O evento de sistema corresponde ao fluxo de exceção no qual o sistema insere o aluno em uma lista de espera, considerando que não existe oferta disponível para alguma disciplina em que esse aluno quis se inscrever fica representado pela referência adicionada a este diagrama.

Figura 5-15: Diagrama de sequência para o caso de uso Realizar Inscrição.



1. Descreva a posição do diagrama de interação no processo de desenvolvimento incremental e iterativo. Quando eles são utilizados? Para quê são utilizados?

2. Considere o fragmento de diagrama de sequência a seguir. Determine a ordem na qual as mensagens m1 e m2 serão passadas.



Resposta: m1, m2, m2, m2, m2, m1, m2, m2, m2

Atividades



MODELAGEM DE CLASSES DE PROJETO

Este capítulo descreve, de acordo com Bezerra (2007), algumas das transformações pelas quais passam as classes e suas propriedades (atributos, operações e associações) com o objetivo de transformar o modelo de classes de análise no modelo de classes de projeto. Neste capítulo, também estudamos outros elementos do diagrama de classes que não são considerados no modelo de análise e que são necessários à construção do modelo de projeto. Esses elementos permitem adicionar mais rigor ao diagrama de classes do sistema. Por exemplo, podemos definir o tipo de cada atributo, a assinatura de cada operação, a navegabilidade de cada associação, etc.

Angue risus at
et velit at tellus.
massa portitor
sectetur magna.

Fala Professor

6.1 TRANSFORMAÇÃO DE CLASSES DE ANÁLISE EM CLASSES DE PROJETO

Esta seção descreve as transformações e os refinamentos que classes de fronteira, de controle e de entidade sofrem na passagem para o modelo de classes de projeto.

6.1.1 Especificação de Classes de Fronteira

As classes de fronteira fazem a interface da aplicação com qualquer entidade externa, como os seus usuários, outras aplicações ou mesmo dispositivos com os quais a aplicação precise interagir. Todos os atores devem se comunicar apenas com essas classes. Elas isolam o núcleo da aplicação do mundo exterior, evitando que mudanças na interface com o mundo exterior afetem outras classes da aplicação. Exemplos de classes de fronteira são: GUI (*Graphical User Interface* – Interface Gráfica do Usuário) e interfaces com outras aplicações.

Para encontrar classes de fronteira, deve-se observar os atores e os casos de uso com os quais interage – em geral, existirá uma classe de fronteira para cada par ator-caso de uso. Por exemplo, em uma aplicação para máquinas de bancos 24h, analisando o caso de uso Sacar Dinheiro podem ser encontradas pelo menos uma classe de fronteira: FormularioSaque.

6.1.2 Especificação de Classes de Controle

As classes de controle coordenam a realização do caso de uso. Casos de uso com fluxos simples podem ser realizados sem classes de controle, com as classes de fronteira trocando informação diretamente com as classes de entidade. Todavia, casos de uso com fluxos mais complexos geralmente precisam de uma ou mais classes de controle para coordenar a ação de outros objetos da aplicação. Elas formam uma camada entre as classes de fronteira e as classes de entidade, permitindo separar o uso da entidade (específico da aplicação) do comportamento inerente à entidade. Classes de controle são dependentes do caso de uso, mas independente do ambiente externo.

As classes de controle deixam as classes de entidade mais reusáveis, isolando-as de comportamento específico da aplicação. Assim, geralmente será encontrada pelo menos uma classe de controle para cada caso de uso. Para o caso de uso Sacar Dinheiro, por exemplo, foi identificada a classe ControladorSaque.

Ainda sobre o tipo comum de objeto controlador, *controlador de caso de uso*, ao qual já fizemos menção na Seção 4.2.2, Bezerra (2007) afirma que esse objeto é responsável pela coordenação da realização de um caso de uso em particular. Mais especificamente, as seguintes responsabilidades são esperadas de um típico controlador de caso de uso:

- deve coordenar a realização de um caso de uso do sistema;
- deve servir como canal de comunicação entre objetos de fronteira e objetos de entidade.
- Deve comunicar-se com outros controladores, quando necessário;
- Deve mapear ações do usuário (ou atores, de uma forma geral) para atualizações ou mensagens a serem enviadas a objetos de entidade;
- Deve estar apto a manipular exceções provenientes das classes de entidades.

6.1.3 Especificações de Classes de Entidade

As classes de entidade representam os conceitos principais da aplicação, as fontes de informação que a aplicação manipula. Elas geralmente são persistentes e sua principal função é armazenar e gerenciar informação. No estudo de caso, proposto por Bezerra (2007) e mencionado neste material, exemplos de classes de entidade são Aluno, Professor,

OfertaDisciplina etc. Classes de entidade também são chamadas de classes do negócio.

Um aspecto importante durante o refinamento do modelo de classes de projeto e relativo às classes de entidade é a forma utilizada para representar os relacionamentos (associações, agregações e composições) entre seus objetos. Essa representação é função da navegabilidade e da multiplicidade definidas para a associação.

Outro aspecto relevante durante a especificação de uma classe de entidade diz respeito ao modo como podemos identificar cada um de seus objetos *unicamente*.

6.2 ESPECIFICAÇÃO DE ATRIBUTOS

6.2.1 Notação da UML para Atributos

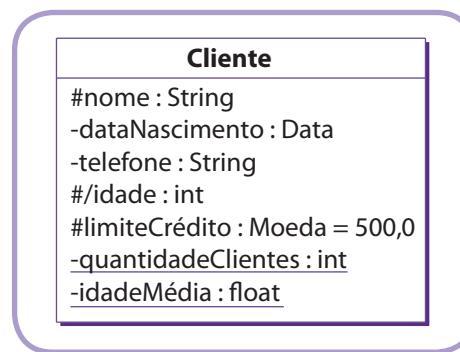
No modelo de classes de análise, os atributos, quando representados, o são somente pelo nome. No entanto, a sintaxe completa da UML para definição de um atributo é bem mais detalhada. Essa sintaxe, cujo uso é apropriado para a fase de especificação do modelo de classes, é a seguinte:

```
[/]visibilidade nome: tipo = valor_inicial
```

Vamos descrever, de acordo com Bezerra (2007), cada um desses elementos. (A Figura 6-1 mostra a utilização desses elementos para atributos da classe Cliente).

A *visibilidade* de um atributo diz respeito ao seu nível de acesso. Ou seja, a visibilidade permite definir quais atributos de um objeto são acessíveis por outros objetos. A visibilidade de um atributo pode ser *pública, protegida, privativa ou de pacote*. A primeira é a visibilidade por omissão (se não for específica da visibilidade alguma, essa é assumida). O símbolo e o significado de cada visibilidade são apresentados na Tabela 6-1.

Figura 6-1: Especificação dos atributos da classe Cliente (Bezerra, 2007).



A propriedade de *visibilidade* serve para implementar o *encapsulamento* da estrutura interna da classe. Somente as propriedades que são realmente necessárias ao exterior da classe devem ser definidas com visibilidade pública. Todo o resto é “escondido” dentro da classe através das visibilidades protegida e privativa. Um atributo definido com a visibilidade protegida diz respeito ao conceito de herança entre classes.

Tabela 6-1: Possíveis visibilidades de um elemento (atributo ou operação).

Visibilidade	Símbolo	Significado
Pública	+	Qualquer objeto externo pode obter acesso ao elemento, desde que tenha uma referência para a classe em que o atributo está definido.
Protegida	#	O elemento protegido é visível para subclasses da classe em que este foi definido.
Privativa	-	O elemento privativo é invisível externamente para a classe em que este está definido.
Pacote	~	O elemento é visível a qualquer classe que pertence ao mesmo pacote no qual está definida a classe.

O elemento *nome* da sintaxe do atributo corresponde ao nome do atributo. Na verdade, somente esse elemento é obrigatório na sintaxe de declaração de um atributo.

O elemento *tipo* especifica o tipo do atributo. Esse elemento é dependente da linguagem de programação na qual a classe deve ser implementada. O tipo definido para um atributo pode ser definido pela utilização de um *tipo primitivo* da linguagem de programação a ser utilizada na implementação. Por exemplo, se estamos utilizando a linguagem Java, temos, dentre outros, os seguintes tipos primitivos: int, float e boolean. É também comum a utilização de *tipos abstratos de dados* (tradução para *abstract data type*, TAD) para declarar atributos. Um TAD é um tipo definido pelo usuário ou um tipo disponível na linguagem de programação que contém uma estrutura interna. Por exemplo, na Figura 6-1, os atributos nome e telefone são definidos como do tipo String, que é um tipo abstrato de dados. Além de String, temos (em Java): List, Set e array. Embora frequentemente corresponda a uma classe, um TAD é representado no comportamento de atributos em vez de ser

representado como uma classe separada no diagrama de classes. Isso porque o diagrama ficaria muito carregado visualmente se essas classes TAD fossem representadas por retângulos separados (Bezerra, 2007).

A maioria das linguagens de programação já fornece tipos abstratos de dados. No entanto, os desenvolvedores podem definir seus próprios tipos. Por exemplo, a Figura 6-1 apresenta o atributo `limiteCrédito` como do tipo `Moeda`, um TAD para manipular valores monetários. Outros exemplos de TAD são listados a seguir.

- Data (para processamento de datas em geral).
- Horário (para processamento de valores de tempo).
- Endereço (para armazenamento de endereços em geral).
- CódigoPostal, CPF e CNPJ (para armazenamento e validação desses valores).

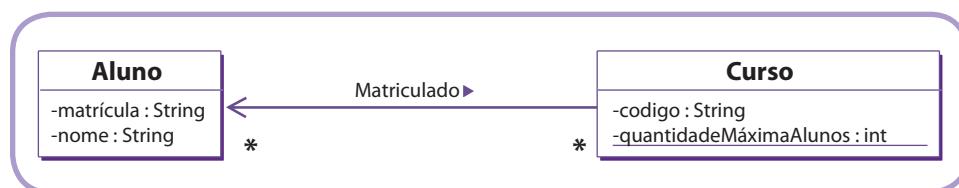
Um *valor inicial* também pode ser declarado para o atributo. Dessa forma, sempre que um objeto de uma classe é instanciado, o valor inicial declarado é automaticamente definido para o atributo. Assim como o tipo, o valor inicial de um atributo também é dependente da linguagem de programação.

Pode-se definir um *atributo derivado* em uma classe. Um atributo é derivado quando o seu valor pode ser obtido a partir do valor de outro(s) atributo(s). Por exemplo, o valor da idade de uma pessoa pode ser obtido a partir de sua data de nascimento. Um atributo derivado é representado com uma barra inclinada à esquerda. Atributos derivados normalmente são definidos por questões de desempenho. Por exemplo, considere a classe Cliente da Figura 6-1. Em vez de calcular a idade média de toda a coleção de clientes a cada vez que este valor é necessário, definimos o atributo derivado `idadeMédia` e um método seletor correspondente.

Outra característica de um atributo é o seu *escopo*. Cada objeto tem um valor para cada atributo definido em sua classe. Esse é o caso mais comum, e diz-se que o atributo tem *escopo de objeto*. No entanto, pode haver atributos que tenham *escopo de classe*, ou seja, que armazenam valor comum a todos os objetos da classe. Esse tipo de atributo é definido por um sublinhado na sintaxe da UML. Um atributo que tenha escopo de classe é também denominado *atributo estático*. Por exemplo, a classe `Cliente` da Figura 6-1 contém o atributo `quantidadeClientes` para armazenar a quantidade de instâncias criadas a partir dessa classe. Há apenas um valor desse atributo para todos os objetos da classe `Cliente`. O atributo `idadeMédia` também é estático.

Uma possível aplicação dos atributos estáticos é na implementação de regras de negócio. Por exemplo, considere a Figura 6-2, em que a classe *Curso* possui um atributo estático *quantidadeMáximaAlunos*. Este atributo pode ser utilizado para implementação da seguinte regra de negócio: *Em um curso podem estar matriculados até 30 alunos.*

Figura 6-2: Atributos estáticos podem ser utilizados para implementar regras de negócio (Bezerra, 2007).



6.3 ESPECIFICAÇÃO DE OPERAÇÕES

Na descrição do modelo de interações, declaramos que as operações de uma classe são identificadas em consequência da identificação de mensagens enviadas de um objeto a outro. No detalhamento dessas operações, devemos considerar diversos aspectos: seu nome, lista de parâmetros, tipo de cada parâmetro, tipo de retorno. Nesta seção, descrevemos a notação fornecida pela UML para realizar esse refinamento de operações. Descrevemos, segundo Bezerra (2007), alguns princípios e dicas recomendados durante essa atividade.

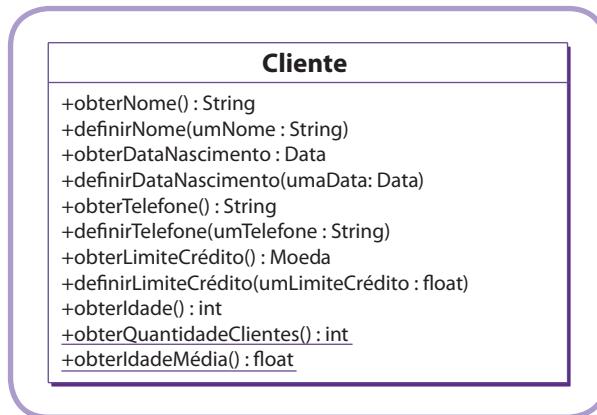
6.3.1 Notação da UML para Operações

As operações de uma classe correspondem a algum processamento realizado por essa classe. Em termos de implementação, uma operação é uma rotina associada a uma classe. Conforme visto no Capítulo 4, a construção do modelo de classes de análise permite a identificação de algumas operações. Durante a construção do modelo de interações, essas operações são validadas e várias outras operações são identificadas. Todas essas operações devem ser adicionadas ao diagrama de classes e devem ser documentadas com a definição de sua *assinatura*.

A sintaxe definida na UML para a assinatura de uma operação é a seguinte:

visibilidade nome(parâmetros): tipo-retorno {propriedades}

Figura 6-3: Especificação das operações da classe Cliente (Bezerra, 2007).



O elemento *nome* corresponde ao nome dado à operação. A *visibilidade* usa a mesma simbologia das visibilidades para atributos. Se uma operação é pública, ela pode ser ativada com a passagem de uma mensagem para o objeto. Se a operação é protegida, ela só é visível para a classe e para seus descendentes (ou *subclasses*). Finalmente, se a operação é privativa, somente objetos da própria classe podem executá-la. Os *parâmetros* de uma operação correspondem a informações que esta recebe quando é executada. Normalmente, essas informações são fornecidas pelo objeto remetente da mensagem que requisita a execução da operação no objeto receptor. Uma operação pode ter zero ou mais *parâmetros*. Estes são separados por vírgulas. Cada parâmetro da lista tem a seguinte sintaxe:

nome-parâmetro: tipo-parâmetro

O elemento *nome-parâmetro* corresponde ao nome do parâmetro. Cada parâmetro deve ter um nome único dentro da assinatura da operação. O elemento *tipo-parâmetro* corresponde ao tipo do parâmetro e é dependente da linguagem de programação. O elemento *tipo-retorno* representa o tipo de dados do valor retomado por uma operação. Esse tipo depende da linguagem de programação. Assim como atributos, as operações também possuem um *escopo*. Uma operação pode ter *escopo de classe* ou *escopo de instância*. Uma operação que tem escopo de classe processa atributos estáticos. Se essa operação tem escopo de classe, ela é também chamada de *operação estática*. Uma operação que tenha escopo de instância indica que ela processa atributos que têm escopo de instância.

6.4 ESPECIFICAÇÃO DE ASSOCIAÇÕES

Nesta seção, descrevemos alguns detalhes, de acordo com Bezerra (2007), a serem adicionados para refinar os relacionamentos de associação identificados do modelo de classes de análise.

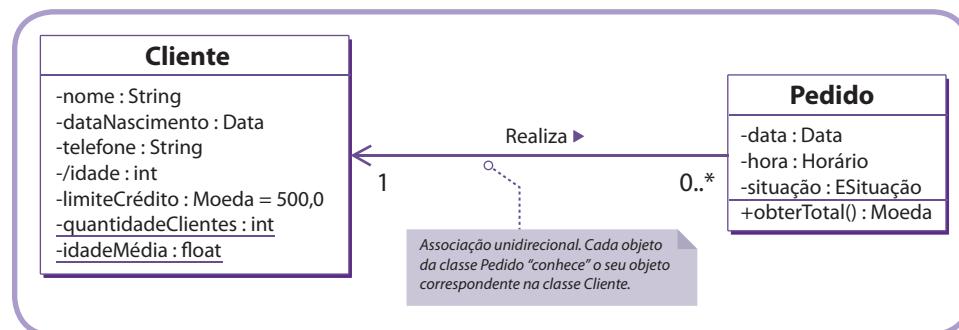
6.4.1 Navegabilidade de Associações

As associações (assim como agregações e composições) podem ser classificadas em *bidirecionais* e *unidirecionais*. Uma associação bidirecional indica que há um conhecimento mútuo entre os objetos associados. Ou seja, se um diagrama de classes exibe uma associação entre duas classes C1 e C2, então as duas assertivas a seguir são verdadeiras.

1. Cada objeto de C1 conhece todos os objetos de C2 aos quais ele está associado.
2. Cada objeto de C2 conhece todos os objetos de C1 aos quais ele está associado.

Graficamente, uma associação unidirecional é representada adicionando-se um sentido à associação. A classe para a qual o sentido aponta é aquela cujos objetos *não* possuem visibilidade dos objetos da outra classe. O diagrama da Figura 6-4 ilustra o uso de uma associação unidirecional. Esse diagrama indica que objetos da classe Pedido possuem, cada qual, uma referência para um objeto Cliente. Por outro lado, um objeto de Cliente não tem referências para os objetos associados em Pedido.

Figura 6-4: Exemplo de associação unidirecional (Bezerra, 2007).



Durante a construção do modelo de classes de análise, associações são costumeiramente consideradas “navegáveis” em ambos os sentidos, ou seja, as associações são bidirecionais. No modelo de classes de projeto, o modelador deve refinar a navegabilidade de todas as associações. Pode ser que algumas associações devam permanecer bidirecionais. No entanto, se não houver essa necessidade, recomenda-se transformar a associação em unidirecional. Isso porque uma associação bidirecional é mais difícil de implementar e de manter do que uma associação unidirecional correspondente. Para entender isso, basta ver que o *acoplamento* entre as classes envolvidas na associação é maior quando a navegabilidade é bidirecional. Portanto, um dos objetivos a alcançar durante o projeto é identificar quais navegaibilidades são realmente necessárias.

A definição do sentido da navegabilidade se dá em função dos diagramas de interação construídos para o sistema (ver Capítulo 5). Mais especificamente, devemos analisar os *fluxos das mensagens* entre objetos no diagrama de interações. Dados dois objetos associados, A e B, se há pelo menos uma mensagem de A para B em algum diagrama de interação, então a associação deve ser navegável da classe de A para a classe de B. Da mesma forma, se existir pelo menos uma mensagem de B para A, então a associação deve ser navegável de B para A.

A definição do sentido da navegabilidade é feita em função dos diagramas de interação construídos para o sistema. Devemos analisar os fluxos das mensagens entre objetos no diagrama de interações.

6.5 MODELO DE CLASSES DE PROJETO NO PROCESSO DE DESENVOLVIMENTO

Em um processo de desenvolvimento iterativo, o modelo de classes de projeto é construído em diversas iterações. Normalmente, esse modelo é construído em paralelo com o *modelo de interações* (ver Capítulo 5). Isso porque o modelo de interações fornece informações para completar o modelo de classes em seu estágio de análise a fim de levá-lo ao estágio de projeto. Em particular, uma mensagem que identificamos durante a construção dos diagramas de interações sempre corresponde a uma operação na classe do objeto remetente. Para cada mensagem identificada, a assinatura da operação correspondente deve ser completamente específica da no modelo de classes de projeto (Bezerra, 2007).

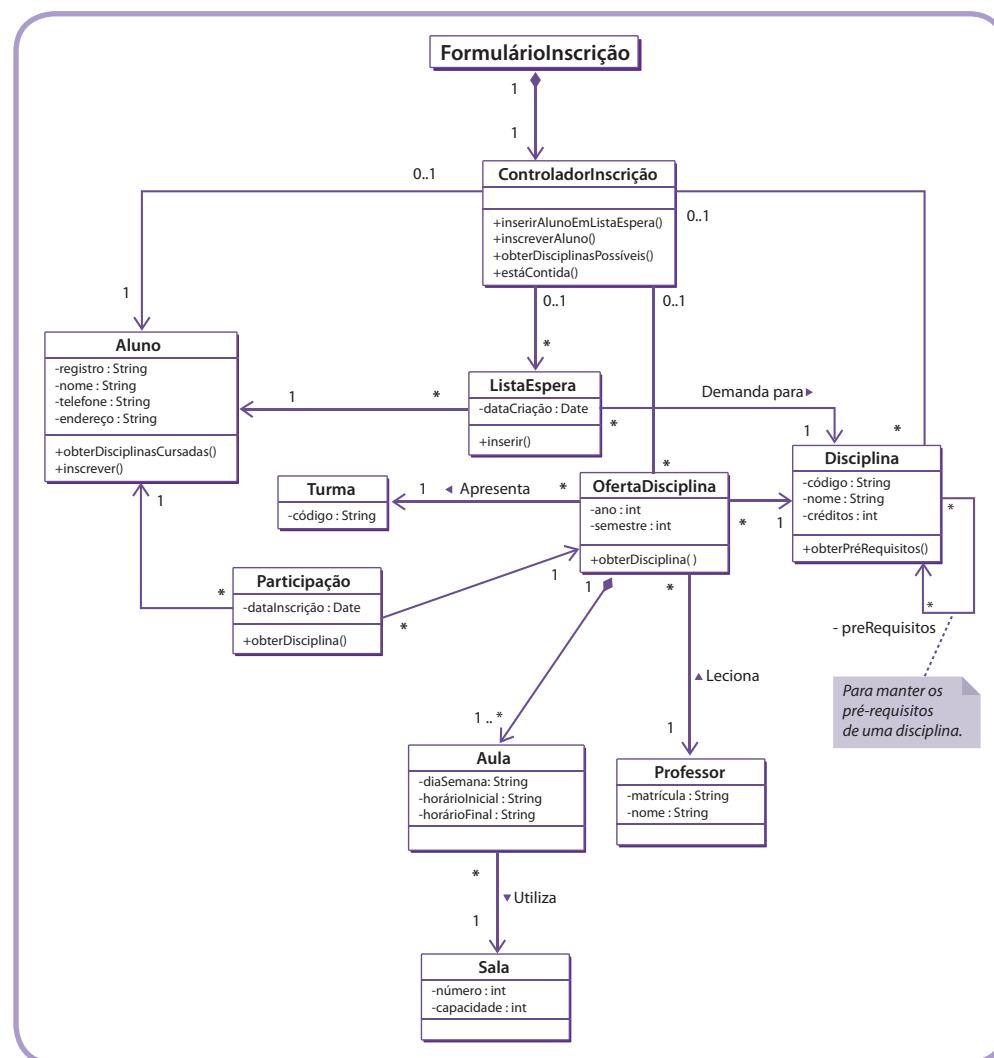
Uma dúvida frequente, segundo o mesmo autor, diz respeito ao nível de detalhe na especificação dos diagramas de classes de projeto. Há diversos fatores que influenciam a resposta para essa questão. Em primeiro lugar, devemos perceber que um modelo que não esteja em conformidade com o sistema não tem valia. Por outro lado, quanto mais detalhados os modelos que construímos possuem, mais difícil mantê-los em sincronia com o código-fonte, pois este último é muito instável. Se a equipe de desenvolvimento tiver à disposição ferramentas CASE que permitam realizar engenharias direta e reversa, a construção de modelos mais detalhados é possível. Isso porque as próprias ferramentas podem atualizar os modelos existentes. Entretanto, se este não for o caso, a equipe corre o risco de perder tanto tempo mantendo os modelos atualizados quanto leva para realmente implementar o sistema. Portanto, o bom senso também vale para decidir o nível de detalhamento que deve ser adotado na construção, não só dos diagramas de classes, mas de qualquer modelo de um SSOO.

6.6 ESTUDO DE CASO

Para exemplificar a especificação de classes no estudo de caso do Sistema de Controle Acadêmico, proposto por Bezerra (2007), considere o diagrama de classes correspondente à visão de classes participantes do caso de uso *Realizar Inscrição*. A Figura 6-5 apresenta uma versão inicial do refinamento desse diagrama, considerando vários assuntos tratados neste capítulo. Por simplificação, os parâmetros das operações não são exibidos.

Note que, por simplificação, apenas o refinamento correspondente a uma das visões de classes participantes é apresentado. Em uma situação real, todas as classes de análise devem ser consideradas.

Figura 6-5: Especificação da Visão de Classes Participantes (VCP) do caso de uso *Realizar Inscrição* (Bezerra, 2007).

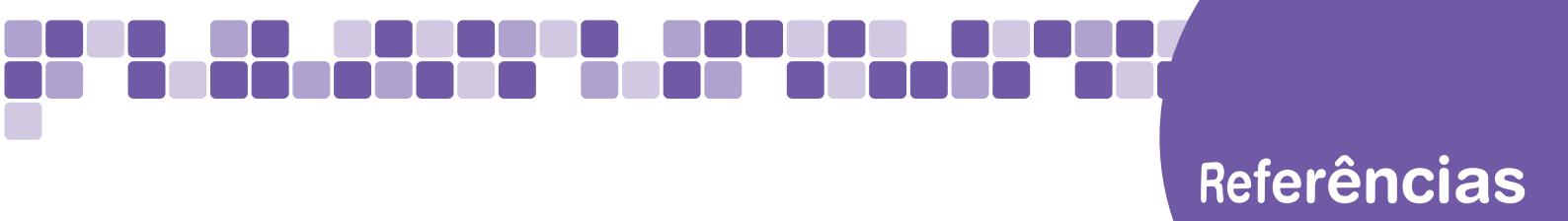


1. Em cada um dos itens abaixo, desenhe o diagrama de classes correspondente, indicando as multiplicidades. Especifique, ainda, possíveis heranças, generalizações, composições.

- a) Uma Pessoa, como programador, utiliza uma Linguagem de Programação.
- b) Um Objeto de Desenho pode ser um Texto, um Gráfico ou um Grupo de objetos.
- c) Modem, Teclado e Impressora são Dispositivos de Entrada e Saída.
- d) Um Banco de Dados contém Tabelas de Sistema e Tabelas de Usuário. Uma Tabela de Sistema mantém informações sobre uma ou várias Tabelas de Usuário. Uma Tabela contém Registros.
- e) Um Item pode ser um Item Atômico ou um Item Composto. Um Item Composto possui dois ou mais Itens.

Atividades





Referências

- BEZERRA Eduardo. **Princípios de Análise e Projeto de Sistemas com UML.** 2^a Edição. Campus, 2007.
- BOOCH Grady, JACOBSON Ivar, RUMBAUGH James. **UML - Guia do Usuário.** 2^a Edição. Campus, 2006.
- DENNIS Alan, WIXOM Barbara Haley. **Análise e Projeto de Sistemas.** 2^a Edição. Ltc, 2005.
- FALBO Ricardo de Almeida. **Notas de Aula: Engenharia de Software.** Disponível em <http://www.inf.ufes.br/~falbo>, 2005.
- JONES, Carper. **Applied Software Measurement**, 1997.
- LARMAN Craig. **Utilizando UML e Padrões.** 3^a Edição. BOOKMAN, 2007.
- MACIASZEK, L. A. **Requirements Analysis and System Design: Developing Information Systems with UML.** Addison Wesley, 2000.
- MCLAUGHLIN Brett; POLLICE Gary; WEST David. **Use a Cabeça! Análise e Projeto Orientado ao Objeto.** 1^a Edição. Alta Books, 2007.
- OMG Object Management Group. "**UML 2.0 Superstructure Final Adopted specification,**" 2003. Disponível em <http://www.omg.org/technology/documents/formal/uml.htm>. 2003.
- RUMBAUGH James. **Modelagem e Projetos Baseados em Objetos com UML 2.** 1^a Edição. Campus, 2006.
- RUP Rational Unified Process, Best Practices for Software Development Teams. White Paper disponível em <http://www-01.ibm.com/software/awdtools/rup/>, 2011.
- SERAFINI, José Inácio. **Análise de sistemas.** Vitória: IFES, 2008.
- WAZLAWICK Raul. **Análise e Projeto de Sistemas de Informação Orientados a Objetos.** 2^a Edição. Campus, 2010.

