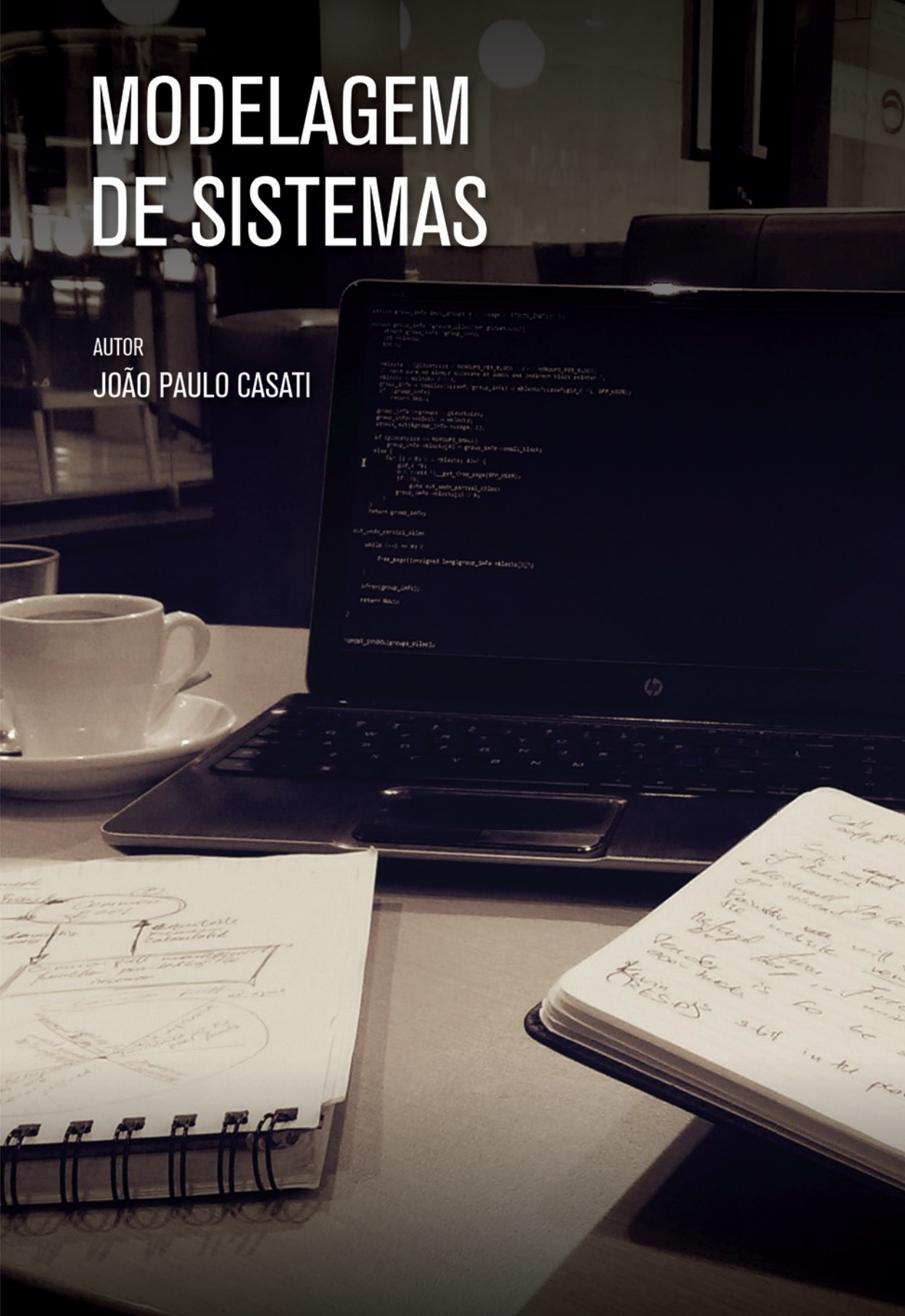
MODELAGEM DE SISTEMAS

AUTOR
JOÃO PAULO CASATI



MODELAGEM DE SISTEMAS

AUTOR
JOÃO PAULO BROGNONI CASATI

1ª EDIÇÃO
SESES
RIO DE JANEIRO 2016

Conselho editorial REGIANE BURGER; ROBERTO PAES; GLADIS LINHARES; KAREN BORTOLOTI; HELCIMARA AFFONSO DE SOUZA

Autor do original JOÃO PAULO CASATI

Projeto editorial ROBERTO PAES

Coordenação de produção GLADIS LINHARES

Coordenação de produção EaD KAREN FERNANDA BORTOLOTI

Projeto gráfico PAULO VITOR BASTOS

Diagramação BFS MEDIA

Revisão linguística AMANDA CARLA DUARTE AGUIAR

Imagem de capa CCO PUBLIC DOMAIN / FAQ

Dados Internacionais de Catalogação na Publicação (CIP)

C336M CASATI, JOÃO PAULO
Modelagem de sistemas / João Paulo Casati
Rio de Janeiro : SESES, 2016.
96 p. : il.

ISBN: 978-85-5548-211-3

1. Modelagem de software. 2. Orientação a objetos. 3. Sistemas de informação. 4. UML. I. SESES. II. Estácio.

CDD 004

Diretoria de Ensino — Fábrica de Conhecimento
Rua do Bispo, 83, bloco F, Campus João Uchôa
Rio Comprido — Rio de Janeiro — RJ — CEP 20261-063

Sumário

Prefácio

Prezados(as) alunos(as),

Sejam bem-vindos à disciplina de modelagem de sistemas. Esta disciplina oferece uma visão sistêmica muito importante para futuros desenvolvedores e analistas de sistemas, pois o seu entendimento facilita toda uma vida de tomada de decisões quanto à abordagem a ser utilizada na resolução de problemas com o uso de *software*.

Neste capítulo são abordados os conceitos iniciais de engenharia de *software* e seus ciclos de desenvolvimento. Posteriormente, conceitos de modelagem são apresentados, indicando a importância da criação de modelos em desenvolvimento de *software*.

Logo após, entra-se no mundo orientado a objetos. Um tema empolgante e de grande importância para o desenvolvimento de *software* contemporâneo. O estudo do paradigma orientado a objetos (também conhecido pela sigla OO) traz grandes benefícios ao raciocínio lógico visando criação de programas para solução de problemas.

Desejo bons estudos e muita atenção nos conceitos de orientação a objetos que vão sendo explicados e revistos ao longo de todo o livro.

Bons estudos!

1

Introdução à Engenharia de *Software* e Modelagem

Este capítulo visa introduzir os conceitos de engenharia de *software*, apresentando um panorama geral de processos de desenvolvimento e o ciclo de vida do desenvolvimento de *software*. Uma introdução aos conceitos de modelagem de sistemas também é apresentada, assim como a explicação do que é orientação a objetos e os conceitos iniciais da linguagem de modelagem chamada UML.
A orientação a objetos é um paradigma muito utilizado em desenvolvimento de *software* e traz também um nível de raciocínio e abstração interessantes para alunos de graduação.

OBJETIVOS

Ao final deste capítulo, o aluno conhecerá:

- Conceitos básicos de engenharia de *software*;
- Os principais ciclos de vida do processo de desenvolvimento de *software*;
- Conceitos básicos de modelagem de sistemas;
- Uma visão crítica do porquê se utilizar de modelos em desenvolvimento de *software;*
- Conceitos iniciais de orientação a objetos;
- Conceitos iniciais sobre a UML.

1.1 O que é a engenharia de *software*?

O *software* está presente, nos dias de hoje, em quase tudo que existe. Como definido por Pressman (2011), os campos de aplicação de *software* são:

- ***Software* de sistema:** programas de computador com a finalidade de atender às necessidades de outros programas;
- ***Software* de aplicação:** programas que possuem o objetivo de solucionar necessidades específicas de negócio (atividade fim);
- ***Software* científico ou de engenharia:** algoritmos que visam processamento numérico pesado (custoso computacionalmente), utilizado em diversas áreas de pesquisa e simulações;
- ***Software* embutido:** programas que residem em um produto ou sistema, utilizados para controlar as atividades do equipamento, como o *software* que controla o funcionamento de um ar condicionado digital.
- ***Software* para linha de produtos:** também conhecido como *softwares* generalistas ou de prateleira, são programas que podem suprir as necessidades de diversos clientes diferentes, por exemplo, *softwares* de criação de planilhas eletrônicas, editores de textos, entre outros;
- **Aplicações para a *web*:** também conhecidas como "*WebApps*", são programas que são executados em um servidor conectado à rede e que pode ser acessado e utilizado por diferentes usuários, desde que possuam acesso;
- ***Software* de inteligência artificial:** algoritmos não numéricos utilizados para a resolução de problemas complexos, com aplicações em redes neurais artificiais, robótica, jogos, entre outras.

Com a crescente dependência de tecnologias de *software* por parte de diversas organizações (empresas, órgãos governamentais, entre outras), o processo de fabricação (desenvolvimento) precisou ser amadurecido, padronizado e melhorado por meio de técnicas de engenharia de software.

A necessidade de se desenvolver *softwares* com maior qualidade deu-se pela procura em se reduzir custos com manutenção e erros, visto que quanto mais cedo os erros são descobertos no processo de desenvolvimento, menos trabalhosa é sua correção.

A engenharia de *software* consiste de um conjunto de procedimentos e técnicas aplicadas ao processo de *software* com o objetivo de obter maior qualidade nos processos de *software.* É importante frisar que este termo não abrange apenas a etapa de desenvolvimento do *software*, mas também sua manutenção.

CONCEITO

O autor Sommerville (2007) define o termo "engenharia de *software*" como:
"Uma disciplina de engenharia relacionada com todos os aspectos da produção de *software*, desde os estágios iniciais de especificação do sistema até sua manutenção, depois que este entrar em operação".

1.2 O processo de *software*

O processo de *software* é o conjunto de atividades que são executadas com o objetivo de criar um produto, neste caso um *software*. De uma forma genérica, Pressman (2011) define cinco atividades para o processo de *software* de acordo com a disciplina de engenharia de *software*, são elas:

- **Comunicação:** visa a compreensão dos objetivos dos interessados, definindo os requisitos para o desenvolvimento do produto;
- **Planejamento:** tem como objetivo a criação de um mapa (também conhecido como plano de projeto de *software*) que contém a descrição das tarefas, possíveis riscos, cronograma de execução, os recursos demandados e o resultado esperado;
- **Modelagem:** criação de modelos (esboços) para representar o produto que deseja desenvolver, fazendo assim com que as necessidades do produto final possam ser melhor entendidas e o projeto de *software* possa ser melhor definido antes do início do desenvolvimento, reduzindo assim os custos de mudanças (quanto mais cedo se identificar uma mudança, menor o custo de reparo);
- **Construção:** desenvolvimento do código do *software,* assim como o teste dos mesmos;
- **Emprego:** entrega do produto ao cliente, seja este totalmente finalizado ou parcial.

As disciplinas da engenharia de *software*, abrangendo agora todo o processo, também podem ser definidas por:

- **Gerência de projeto:** planejamento das funções a serem executadas;
- **Levantamento de requisitos:** levantar conhecimento sobre o negócio do cliente, identificando suas necessidades;
- **Análise:** detalhamento dos requisitos e definição lógica dos procedimentos;

• **Projeto:** definição física dos procedimentos, inclusive abrangendo os recursos tecnológicos a serem utilizados;

• **Implementação:** construção do sistema (códigos);

• **Teste:** validação do sistema, verificando se os requisitos levantados foram devidamente desenvolvidos, se atendem às expectativas do cliente e se o sistema funciona sem erros;

• **Implantação:** disponibilizar o produto ao cliente (usuário), inclusive treinamentos e carga de dados;

• **Manutenção:** efetuar os ajustes necessários em caso de erros no programa, no levantamento de requisitos ou até mesmo no desenvolvimento de novas funcionalidades, conforme necessidade do cliente; e a

• **Qualidade:** efetuar medidas para a verificação e busca pela excelência do sistema.

A abordagem utilizada neste livro considera as cinco atividades definidas por Pressman (2011) por se tratar de uma abordagem mais simples e de fácil entendimento, porém, pode-se facilmente adaptar à outras definições mais detalhadas.

A padronização no processo de *software* pode melhorar o desempenho e reduzir custos do desenvolvimento de *software*. Com isto, alguns modelos são comumente adotados com o objetivo de otimizar o processo de desenvolvimento de *software* (SOMMERVILLE, 2007).

Os modelos de processos, também conhecidos como ciclos de vida de *software* mais utilizados são:

- O modelo cascata;
- O modelo evolucionário; e
- O modelo incremental.

Todos estes modelos podem acomodar as cinco atividades definidas anteriormente (comunicação, planejamento, modelagem, construção e emprego) e são mais detalhadamente descritas a seguir.

1.2.1 O modelo cascata

Também conhecido como ciclo de vida clássico, o modelo cascata possui suas atividades sendo executadas de forma linear. A figura 1.1 apresenta a sequência de execução das atividades utilizando o modelo cascata.

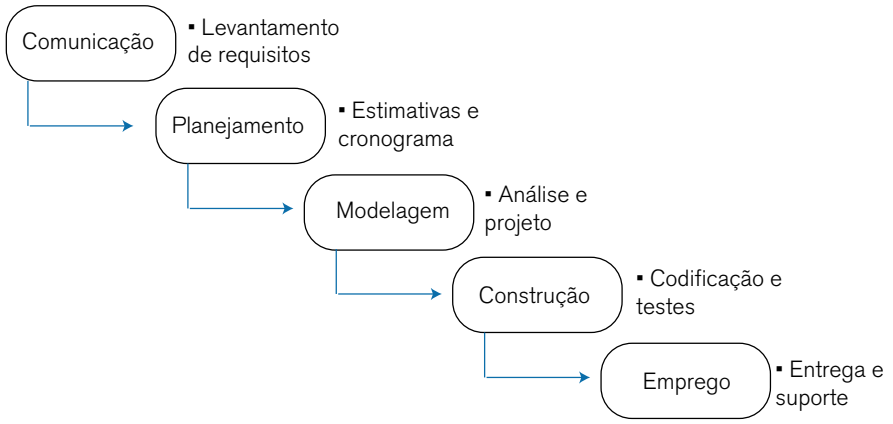


Figura 1.1 – Modelo cascata, adaptado de Pressman (2011).

Como é possível observar na figura 1.1, as etapas são executadas de forma linear, sendo assim, a etapa seguinte é dependente da anterior. Em outras palavras, uma etapa só pode ser iniciada quando a etapa anterior é finalizada.

Um problema recorrente do uso deste modelo é: se uma das etapas não for devidamente cumprida, a etapa seguinte tende a ser desenvolvida erroneamente.

EXEMPLO

Ao utilizar-se do modelo cascata, quando um requisito é entendido de forma errada na etapa de comunicação e este erro persiste até a etapa de emprego, é preciso retornar à etapa de comunicação e passar novamente por todas as outras etapas até que o emprego seja executado novamente. Sendo assim, ao utilizar-se deste modelo, é imprescindível que cada etapa seja exaustivamente checada antes de se passar para a próxima, a fim de se evitar custos adicionais e atrasos no prazo de entrega.

1.2.2 O modelo evolucionário

Com a necessidade de alterações e mudanças nas definições do projeto conforme o desenvolvimento avança, o modelo evolucionário visa amenizar o problema de interdependência das atividades do modelo cascata (PRESSMAN, 2011).

Esta abordagem tem como objetivo desenvolver o *software* de maneira que este evolua com o passar do tempo. Com isto, a atividade de emprego (entrega do produto ao cliente) pode ser feita sem o *software* estar finalizado, mas com uma parcela deste para que as entregas sejam mais rápidas e o cliente possa acompanhar o desenvolvimento do produto.

Este modelo pode ser dividido em duas diferentes abordagens:

- Prototipação; e
- Modelo espiral.

A prototipação é utilizada como ferramenta para a descoberta de requisitos de forma interativa. As atividades são executadas seguindo a ordem exibida na figura 1.2 e, quando da atividade de emprego do *software*, pondera-se a necessidade de reavaliação dos requisitos, fazendo assim com que todas as etapas sejam revistas e ao final, um novo protótipo seja entregue ao cliente.

A vantagem desta abordagem para a clássica (cascata) é que o produto final não é totalmente desenvolvido para a avaliação dos requisitos pelo cliente e sim um protótipo, de mais fácil manutenção e alteração e mais rapidamente entregue.

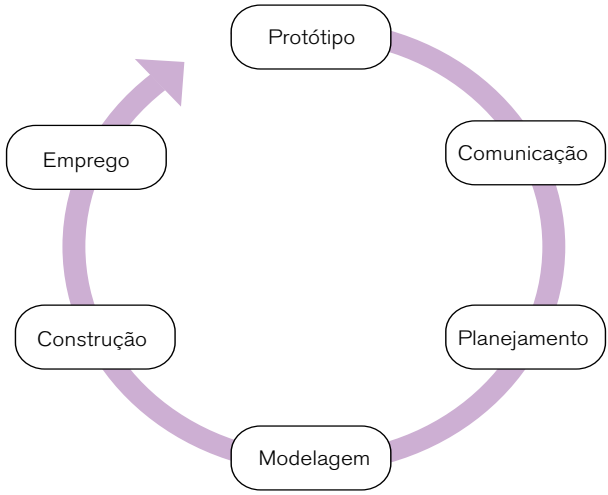


Figura 1.2 – Modelo de prototipação, adaptado de Pressman (2011).

Para a utilização da abordagem de prototipação, é necessário que todos os envolvidos estejam de acordo que os protótipos a serem desenvolvidos são mecanismos para a definição mais precisa dos requisitos, evitando assim problemas de entendimento (PRESSMAN, 2011).

A segunda abordagem do modelo evolucionário é o modelo espiral, proposto por Boehm (1988) em seu artigo científico.

Uma das diferenças do modelo espiral para os outros modelos é que seu ciclo não necessariamente termina quando o *software* é entregue, ele pode ser adaptado para perdurar por toda a vida do *software* (PRESSMAN, 2011).

Segundo Sommerville (2007), o modelo espiral é orientado a riscos. Ele busca definir os riscos a cada iteração (volta no ciclo) e minimizá-los durante o processo.

Analisando a figura 1.3 pode-se ter uma ideia melhor de como o modelo espiral funciona.

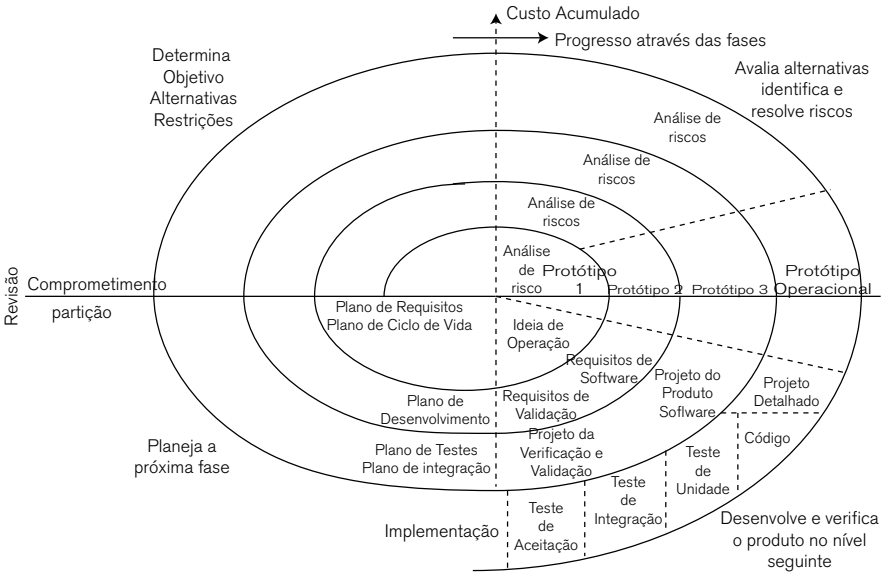


Figura 1.3 – Modelo espiral (NEPOMUCENO, 2015), adaptado de Boehm (1988).

A análise de riscos é feita na etapa de planejamento e a cada volta em torno do eixo do espiral pode ser utilizada para o desenvolvimento de um protótipo (PRESSMAN, 2011).

1.2.3 O modelo incremental

Muito utilizado nos dias de hoje devido às metodologias de desenvolvimento ágeis, este modelo, também conhecido como ciclo interativo, visa combinar as vantagens do modelo cascata com as vantagens do modelo espiral (SOMMERVILLE, 2007).

CONEXÃO

Uma das metodologias ágeis mais utilizadas para o desenvolvimento de *software* é o Scrum. Informações sobre esta metodologia podem ser encontradas no *site:* http://www.desenvolvimentoagil.com.br/scrum/

Os serviços a serem fornecidos pelo *software* a ser produzido são devidamente identificados e desenvolvidos independentemente. Sendo assim, logo na primeira entrega, o sistema já pode ser utilizado.

A figura 1.4 apresenta a forma como o modelo incremental funciona, onde cada etapa de emprego é a entrega do *software* incrementado para o cliente.

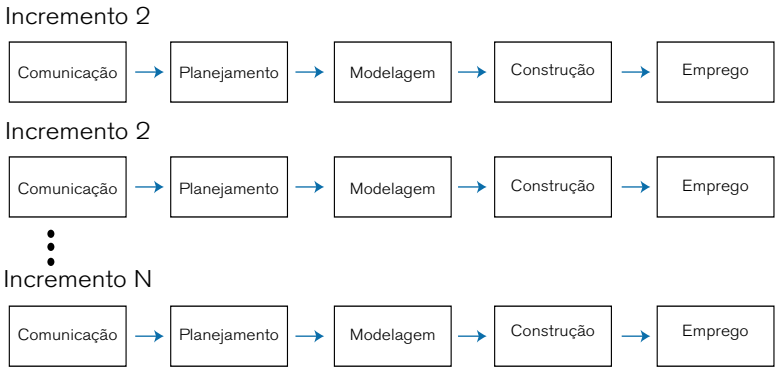


Figura 1.4 – Modelo incremental, adaptado de Pressman (2011).

Cada incremento entregue ao cliente é um produto perfeitamente operável. Geralmente se divide o sistema em diversas funcionalidades e em cada incremento, escolhe-se algumas destas funcionalidades para integrarem a próxima versão do sistema. Estas funcionalidades são completamente desenvolvidas e a versão do *software* gerada a partir do novo incremento é perfeitamente utilizável.

De acordo com Pressman (2011), um dos problemas do modelo incremental é definir as funcionalidades em tamanho adequado, visto que os incrementos devem ser relativamente pequenos, com o objetivo de serem entregues em pouco tempo.

1.3 Princípios de modelagem de sistemas

Para que haja uma melhor compreensão daquilo que se quer desenvolver, uma das medidas adotadas no processo de desenvolvimento de *software* é a criação de modelos (BOOCH; RUMBAUGH; JACOBSON, 2005).

Os modelos de sistemas, diferentemente dos modelos apresentados anteriormente (modelos de processo de *software*), buscam representar o sistema (*software*) em si com relação aos requisitos do mesmo. Em outras palavras, os modelos representam a estrutura do *software* a ser desenvolvido e suas funcionalidades.

São apresentados por Sommerville (2007) alguns tipos de modelos de sistema, são eles:

- **Modelo de fluxo de dados:** exibe o processamento dos dados em diferentes estágios do sistema;
- **Modelo de composição:** também conhecido como modelo de agregação, exibe como as entidades do sistema são compostas por outras entidades;
- **Modelo de arquitetura:** apresentam os subsistemas que compõem um sistema;
- **Modelo de classificação:** mostram como as entidades de um sistema possuem características em comum; e
- **Modelo de estímulo-resposta:** apresenta a reação do sistema a estímulos externos e internos, conhecido também como diagrama de transição de estados ou máquina de estados.

Os modelos de sistemas são criados com base nos requisitos de um sistema e podem ter diferentes níveis de abstração, dependendo da necessidade. Informações sobre o porquê da criação de modelos em desenvolvimento de *software* são apresentadas a seguir.

1.3.1 A importância da modelagem

Como dito anteriormente, os modelos são representações da realidade e segundo Booch, Rumbaugh e Jacobson (2005), esta representação é simplificada. Com esta simplificação da realidade, o modelo tende a ser mais facilmente entendido por pessoas do que a realidade (um modelo é mais facilmente compreensível que o código-fonte do sistema).

A possibilidade de se fazer alterações e redefinições do projeto de *software* na fase de modelagem pode trazer redução de custos, pois as alterações que são feitas na fase de construção do *software* são mais difíceis de serem executadas.

COMENTÁRIO

A criação de um modelo facilita o entendimento do *software*, facilita a definição dos requisitos e pode trazer redução de custos possibilitando alterações no *software* ainda na fase de modelagem.

Além da fase de desenvolvimento do *software*, os modelos de um sistema apoiam também a fase de manutenção, pois podem fornecer uma visão mais próxima do requisito e do funcionamento do sistema, facilitando o emprego de novas funcionalidades e identificação de áreas falhas para a correção de erros.

Outra grande vantagem de se modelar os sistemas é que, quando uma equipe de desenvolvimento de *software* é alterada, os novos membros precisam entender o sistema para que possam atuar e continuar o desenvolvimento, ou mesmo dar manutenção no código criado por outra pessoa. Modelos de sistemas auxiliam neste entendimento do sistema como um todo, assim como o entendimento específico de suas funcionalidades, agilizando, assim o processo de integração de novos membros.

Como princípios da modelagem, Booch, Rumbaugh e Jacobson (2005) definem:

- A definição de soluções de um problema é influenciada pela escolha de quais modelos serão criados;
- Os níveis de abstração e precisão de cada modelo podem variar;
- Modelos são mais efetivos se relacionados com a realidade; e
- Sistemas mais complexos são melhor representados por um conjunto de modelos com vários pontos de vista.

A escolha correta dos modelos a serem criados para o desenvolvimento de *software* pode priorizar os principais problemas, auxiliando em sua resolução. Escolhas erradas podem distrair a atenção para questões irrelevantes.

O ideal é que se possua um conjunto de modelos para a descrição dos problemas e, se possível, criados segundo diferentes pontos de vista e inter-relacionados quando necessário.

1.4 A modelagem orientada a objetos

Diferentemente da visão tradicional do desenvolvimento de *software*, onde a modularização é feita por meio de procedimentos e funções, a orientação a objetos é um paradigma contemporâneo. Na visão orientada a objetos os blocos de construção dos programas são classes e objetos (BOOCH; RUMBAUGH; JACOBSON, 2005).

O completo entendimento sobre a orientação a objetos e como aplicá-la no desenvolvimento de *software* demanda um nível de abstração elevado. O nível de abstração e a definição de quais serão as classes componentes do *software* dependem do problema a ser atacado, isto é, pode-se ter diferentes interpretações para cada problema. Esta abstração parte de princípios do que é uma classe e o que é um objeto, conceitos que serão apresentados e exemplificados no decorrer do livro.

COMENTÁRIO

Uma vez, quando cursava a graduação em sistemas de informação, ouvi de um professor de análise de sistemas, que o conceito de orientação a objetos parecia fácil, mas que o amadurecimento do conhecimento demora a acontecer e que em alguns anos, talvez uns 5 anos, nós alunos teríamos noção do que realmente é a orientação a objetos, até mesmo depois de utilizá-la em nosso dia-a-dia. Em termos ele estava correto, a utilização plena da orientação a objetos vai muito além daquilo que aprendemos em uma disciplina, só o tempo e a experiência nos farão amadurecer este conhecimento e aplicá-la plenamente em nossos *softwares*.

1.5 A linguagem UML

A linguagem de modelagem unificada, UML (*Unified Modeling Language*) foi elaborada por Grady Booch, James Rumbaugh e Ivar Jacobson e sua primeira versão foi lançada em janeiro de 1997 (ERIKSSON et al., 2004).

Algumas mudanças relevantes surgiram na versão 2.0 da UML. Estas mudanças aprimoraram a linguagem para certos paradigmas e visões, adicionando mais funcionalidades e deixando-a mais intuitiva (ERIKSSON et al., 2004). Neste livro, a versão abordada é a mais recente, a versão 2.0, abordada no livro de 2005 escrito pelos criadores da linguagem.

A UML não é apenas uma linguagem de modelagem de *software*, mas também pode ser empregada em diversas outras áreas de conhecimento. Algumas destas áreas são citadas por Booch, Rumbaugh e Jacobson (2005):

- Fluxo de trabalho no sistema legal;
- Estrutura de sistemas de saúde; e
- Projeto de *hardware*.

Apenas de poder trafegar por diversas áreas do conhecimento, o foco principal da UML é ser empregada em sistemas de *software* complexos. São alguns casos onde a UML pode ser aplicada, segundo Booch, Rumbaugh e Jacobson (2005):

- Sistemas de informação corporativos;
- Serviços bancários;
- Serviços financeiros;
- Setor de transportes;
- Setor de telecomunicações;
- Sistemas de eletrônica médica;
- *Softwares* científicos;
- Serviços distribuídos na internet;
- Entre muitas outras.

A UML abrange não só a documentação da arquitetura do sistema, mas também de seus detalhes, podendo expressar requisitos e modelar as atividades de planejamento (BOOCH; RUMBAUGH; JACOBSON, 2005).

AUTOR

Os autores Grady Booch, James Rumbaugh e Ivar Jacobson são os criadores originais da UML. Estes autores são reconhecidos mundialmente por suas contribuições significativas para a tecnologia de objetos.

De acordo com seus criadores, Booch, Rumbaugh e Jacobson (2005), a UML possui um total de 13 diagramas, são eles:

- Diagrama de classes;
- Diagrama de objetos;
- Diagrama de componentes;
- Diagrama de estruturas compostas;

- Diagrama de casos de uso;
- Diagrama de sequências;
- Diagrama de comunicações;
- Diagrama de gráficos de estados;
- Diagrama de atividades;
- Diagrama de implantação;
- Diagrama de pacote;
- Diagrama de temporização; e
- Diagrama de visão geral da interação.

Apenas para exemplificar o que seria uma classe representada em um diagrama de classes, a figura 1.5 mostra a classe "Pessoa", com seus atributos e métodos.

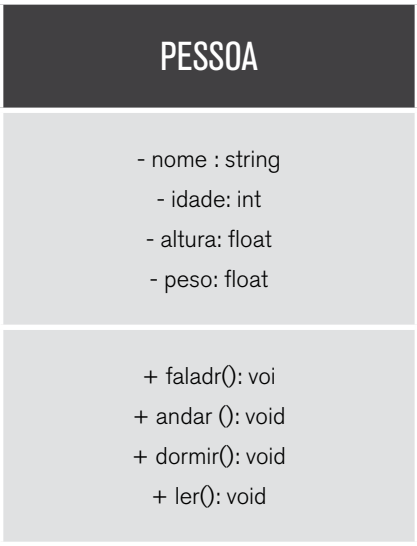


Figura 1.5 – Representação de uma classe na UML.

A representação apresentada na figura 1.5 mostra uma classe chamada "Pessoa". Como visto, o nome da classe aparece no primeiro espaço retangular.

Logo abaixo, no segundo espaço retangular, ficam os atributos da classe, que são as características que farão parte da descrição dos objetos que forem instanciados da classe "Pessoa". Neste caso, temos os atributos nome, idade, altura e peso. Portanto, cada objeto instanciado possui seu valor para estes atributos, em outras palavras, cada pessoa possuiria seu próprio nome, sua idade, sua altura e peso.

No terceiro espaço retangular, são exibidos os métodos. Estes métodos nada mais são que as ações que as "pessoas" (objetos instanciados da classe "Pessoa") podem executar. Neste caso, o objeto tem a possibilidade de falar, andar, dormir e ler.

As classes fazem parte da estrutura do sistema e podem ser relacionadas entre si, porém estes e outros conceitos de orientação a objetos são exaustivamente discutidos no restante do livro (vamos com calma, pois o conceito de orientação a objetos é extenso e complexo).

Na modelagem de um sistema não é necessário que se utilize de todos os diagramas disponíveis. O nível de detalhamento e uso dos diagramas depende do tipo de sistema e da abordagem desejada para a documentação do mesmo. É necessário que haja bom senso por parte de quem analisa e documenta o *software*.

ATIVIDADES

01. Cite duas desvantagens de se utilizar o modelo cascata em desenvolvimento de *software*.

02. Em um sistema de locação de carros, cite o nome de duas possíveis classes.

03. Qual dos modelos de processo de *software* é comumente utilizado quando se adota uma metodologia ágil?

REFLEXÃO

Neste capítulo foram apresentados os conceitos básicos de engenharia de *software* assim como os principais ciclos de vida de desenvolvimento de *software*, uma visão essencial para desenvolvedores que desejam aplicar técnicas reconhecidas e amplamente testadas em seus processos de desenvolvimento. Também foram abordados conceitos de modelagem de sistemas, apresentando a necessidade e as vantagens de se modelar software no processo de desenvolvimento, permitindo ao aluno adquirir uma visão crítica sobre diversos aspectos do desenvolvimento de *software*. A linguagem UML foi apresentada neste capítulo, esta será utilizada durante quase todo o livro, portanto, seus conceitos básicos são de fundamental importância para um bom aprendizado prático.

LEITURA

Na página 45 do livro "Engenharia de *Software:* uma abordagem profissional" de Roger S. Pressman (PRESSMAN, 2011) são apresentados alguns mitos relativos à *software*, abordando gerenciamento, clientes e profissionais da área. É importante ficar a par do que é mito e do que é realidade neste caso.

PRESSMAN, Roger S.. **Engenharia de Software**: Uma Abordagem Profissional. 7. ed. Porto Alegre: Amgh, 2011.

REFERÊNCIAS BIBLIOGRÁFICAS

BOEHM, B. W.. A spiral model of software development and enhancement. **Computer,** [s.l.], v. 21, n. 5, p.61-72, maio 1988. Institute of Electrical & Electronics Engineers (IEEE). DOI: 10.1109/2.59.

BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. **UML**: Guia do Usuário. 2. ed. Rio de Janeiro: Campus, 2005.

ERIKSSON, Hans-erik et al. *UML 2 Toolkit.* Indianapolis: Wiley, 2004.

PRESSMAN, Roger S.. **Engenharia de Software:** Uma Abordagem Profissional. 7. ed. Porto Alegre: Amgh, 2011.

SOMMERVILLE, Ian. **Engenharia de Software**. 8. ed. Pearson: São Paulo, 2007.

NEPOMUCENO, Dênys. **Modelos Incremental, Espiral e de Prototipação.** Disponível em: <http://engenhariadesoftwareuesb.blogspot.com.br/2012/12/blog-post.html>. Acesso em: 22 set. 2015.

2

Ferramentas Case e Casos de Uso

Este capítulo aborda uma questão subjetiva, porém de grande importância na modelagem de sistemas, a visão de sistema, que permite variações na modelagem e foco em diferentes pontos de vista.

O uso de ferramentas CASE também é abordado e você verá que o trabalho de modelagem e análise de sistemas pode ser assistido por computador, trazendo grandes vantagens e facilidades.

Diagramas de casos de uso são um dos mais utilizados modelos para a descrição de *software*. Com estes diagramas pode-se representar cenários de atuação no *software* por fatores externos. Este capítulo traz os conceitos de casos de uso e exemplos de diagramas para diversos cenários de sistemas, visando sua familiarização com este tipo de modelo.

OBJETIVOS

Ao final deste capítulo, o aluno conhecerá:

- As diferentes visões de sistema;
- Ferramentas CASE; e
- Casos de uso.

2.1 Visões de sistema

Um sistema de *software* complexo, para ser desenvolvido, demanda uma série de precauções por parte de seus analistas e desenvolvedores. A possibilidade de se "enxergar" este sistema por diferentes pontos de vista enriquece o detalhamento dos modelos.

De acordo com Bezerra (2007), cada visão enfatiza aspectos distintos, são elas:

- Visão de casos de uso;
- Visão de projeto;
- Visão de implementação; e
- Visão de processo.

A figura 2.1 exibe o esquema de como funcionam as diferentes visões de sistema.

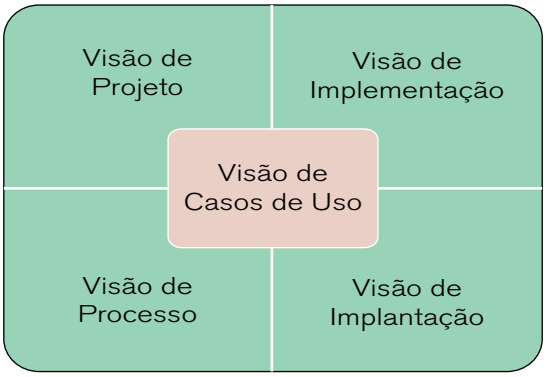


Figura 2.1 – Visões de sistema, adaptado de Bezerra (2007).

A visão de casos de uso utiliza o contexto externo para a modelagem do sistema, isto é, modela as interações do sistema com o ambiente e agentes externos. Segundo Bezerra (2007) os modelos criados a partir desta visão geralmente servem de direcionamento para os modelos das demais visões.

As demais visões dão suporte a áreas específicas do sistema. A visão de projeto dá suporte a funcionalidades externamente visíveis, a visão de implementação dá suporte ao gerenciamento de versões, módulos e subsistemas, a visão de processo evidencia características de desempenho e paralelismo e a visão

de implantação foca na distribuição física do sistema, sua instalação e acoplamento dos subsistemas (BEZERRA, 2007).

Nem todas as visões devem ser obrigatoriamente construídas, isto depende da complexidade e do foco do sistema, da documentação e dos modelos necessários.

2.2 Ferramentas CASE

Ferramentas de modelagem de *software* são muito utilizadas em processos de desenvolvimento por auxiliarem a criação dos modelos, além de algumas outras vantagens, estas ferramentas são chamadas de CASE (*Computer-Aided Software Engineering*, em português, Engenharia de *Software* Assistida por Computador). Pode-se citar como algumas das vantagens do uso de ferramentas CASE:

- Agilidade na criação de modelos;
- Facilidade de alteração nos modelos;
- Padronização na apresentação de modelos;
- Geração automática de código-fonte a partir dos modelos;
- Possibilidade de compartilhar os modelos salvos em arquivos; e
- Sincronizar modelos com o código-fonte da aplicação.

Algumas ferramentas CASE são pagas, outras gratuitas. Mesmo as gratuitas podem ter alguns recursos pagos. Seguem alguns exemplos de ferramentas CASE para UML:

- Jude;
- Rational Rose;
- Dia;
- ArgoUML;
- StarUML;
- Omondo Eclipse UML;
- Enterprise Architect;
- Astah; e
- Umbrello;

Um recurso muito interessante do uso de ferramentas CASE (algumas delas) é o sincronismo entre o modelo do *software* e o código-fonte. Este recurso permite que o modelo esteja sempre atualizado, mesmo após alterações serem feitas diretamente no código, mantendo a documentação sempre válida e atual. Em alguns casos, alterações feitas no modelo podem repercutir em alterações automáticas no código-fonte e sua estrutura, mas esta abordagem é pouco utilizada por programadores e analistas de sistemas.

Os modelos apresentados neste livro são criados utilizando a ferramenta Astah. O uso básico desta ferramenta é descrito a seguir.

2.2.1 A ferramenta Astah

A ferramenta CASE Astah existe na versão *professional* (paga) e a *community* (gratuita). Estudantes podem fazer a requisição da licença da versão *professional* de forma gratuita.

CONEXÃO

No *site* oficial da ferramenta Astah pode-se obter maiores informações e fazer o *download* do *software.*
http://astah.net/

Esta ferramenta possui uma série de recursos. Para a elaboração dos modelos necessários para o acompanhamento da disciplina, a sua versão *community* é suficiente. Por possuir uma *interface* gráfica de fácil utilização e permitir a criação de uma série de modelos da UML, esta ferramenta é utilizada na elaboração deste livro.

A *interface* do Astah é apresentada na figura 2.2.

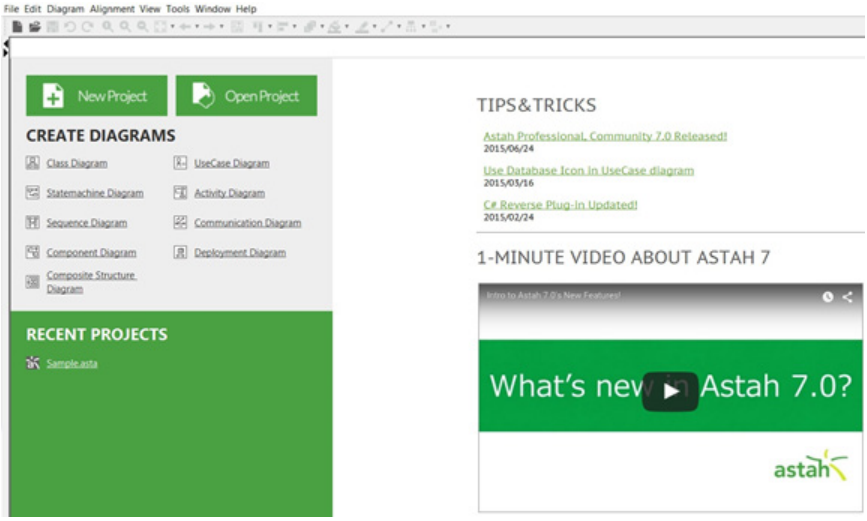


Figura 2.2 – interface inicial da ferramenta Astah.

Diversos diagramas podem ser criados utilizando a ferramenta Astah, sejam eles da UML ou não, como pode-se observar na figura 2.3. Porém, alguns destes somente na versão *professional*, marcados com [PAID].

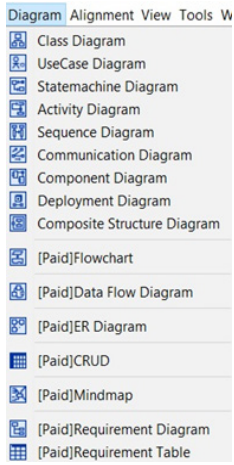


Figura 2.3 – Diagramas do Astah.

Os diagramas possíveis de serem criados com a versão *community* do Astah, traduzindo para o português, são:

- Diagrama de classes;
- Diagrama de casos de uso;
- Diagrama de gráfico de estados;
- Diagrama de atividades;
- Diagrama de sequência;
- Diagrama de comunicação;
- Diagrama de componentes;
- Diagrama de implantação; e
- Diagrama de estruturas compostas.

A seguir, os primeiros diagramas serão apresentados: os diagramas de casos de uso.

2.3 Casos de uso

Como o próprio nome faz-se lembrar, os casos de uso são descrições de utilização do sistema. Em outras palavras, os modelos de casos de uso visam descrever como o sistema é operado por agentes externos.

De acordo com Sommerville (2007), a técnica é baseada em cenários e são de fundamental importância para a modelagem de requisitos de sistemas orientados a objetos utilizando a UML.

Um exemplo de caso de uso é apresentado na figura 2.4. Pode-se observar que a representação de um caso de uso em um modelo de sistema é feita por uma elipse e, internamente, o nome do caso de uso.

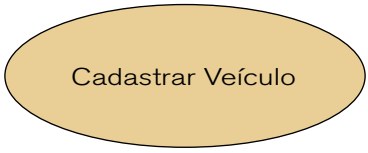


Figura 2.4 – Exemplo básico de caso de uso.

No caso do exemplo apresentado na Figura 2.4, a função de cadastramento de veículo está sendo representada pelo caso de uso em questão. Portanto, pode-se definir que o caso de uso representa o requisito do sistema.

Na utilização de casos de uso para modelagem de sistemas, deve-se ater em representar claramente a funcionalidade no nome do caso de uso, a fim de que o modelo seja entendido com facilidade.

CONEXÃO

O *link* a seguir leva a uma página da Universidade Federal de Campina Grande a qual possui um resumo sobre os diagramas de casos de uso.
http://www.dsc.ufcg.edu.br/~sampaio/cursos/2007.1/Graduacao/SI-II/Uml/diagramas/usecases/usecases.htm

Para que se possa interagir diretamente com o caso de uso, é necessário que um agente externo seja representado também. Este agente externo é chamado de ator.

Podemos dizer que o ator é quem realiza o caso de uso. Sua representação gráfica é demonstrada na figura 2.5.

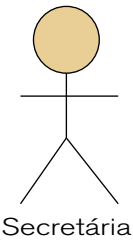


Figura 2.5 – Exemplo de representação de ator.

O ator não precisa necessariamente ser uma pessoa, ele pode representar setores, outros sistemas, organizações, entre outros.

Assim como outros modelos, os casos de uso podem ser de diversas complexidades, dependendo do nível de abstração que se pretende atingir com o modelo.

A interação entre casos de uso e atores são explicados e exemplificados a seguir.

2.3.1 Diagrama de casos de uso

A forma mais básica do diagrama de caso de uso é a interação entre um ator e um caso de uso. Neste caso, a interação representa que o ator está realizando o caso de uso. É demonstrado na figura 2.6 a interação entre um ator e um caso de uso.

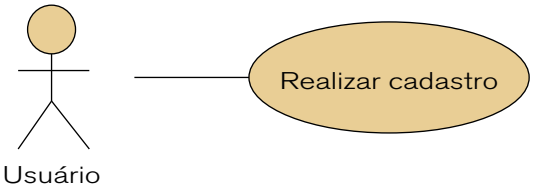


Figura 2.6 – Interação entre ator e caso de uso.

No caso apresentado na figura 2.6, o ator "usuário" é quem realiza cadastro, portanto ele é o executor do caso de uso "realizar cadastro". Para representar esta interação utiliza-se uma linha que liga o ator ao caso de uso.

Existem também interações entre casos de uso. Estas interações representam um caso de uso que é responsável por realizar outro caso de uso (e não mais um ator realiza um caso de uso).

As relações entre casos de uso podem ser de dois tipos:

- <<include>> ou
- <<extend>>.

O exemplo de diagrama de caso de uso exibido na Figura 2.7 apresenta os dois tipos de relação entre casos de uso.

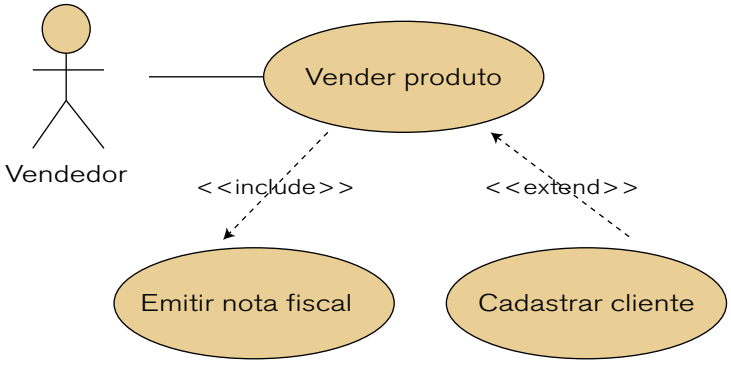


Figura 2.7 – Exemplo de relação entre casos de uso.

No exemplo da Figura 2.7 um vendedor realiza a venda de um produto. Porém, quando o caso de uso "vender produto" é efetuado, outros casos de uso podem ser realizados automaticamente. Neste caso, pode-se fazer uma analogia em que o ator dos casos de uso "emitir nota fiscal" e "cadastrar cliente" é o caso de uso "vender produto".

No caso de "emitir nota fiscal", por se tratar de uma relação de <<include>>, este caso de uso será obrigatoriamente realizado. Em outras palavras, ao vender um produto, a nota fiscal é emitida obrigatoriamente no sistema.

Já no caso de uso "cadastrar cliente", por ser uma relação do tipo <<extend>>, não é obrigatório que o cliente seja cadastrado a cada venda de produto efetuada.

Pode-se dizer, analisando o exemplo da Figura 2.7, que este sistema trabalha da seguinte forma:

- O vendedor é o responsável por efetuar a venda de produtos;
- Ao se realizar uma venda, a nota fiscal é obrigatoriamente emitida;
- Caso o cliente ainda não seja cadastrado, ao efetuar uma venda, pode-se cadastrar o cliente.

Os atores podem ser representados como uma hierarquia, também chamada de herança ou de generalização. Conceitos mais aprofundados de herança são apresentados no próximo capítulo. A generalização de atores é representada na figura 2.8.

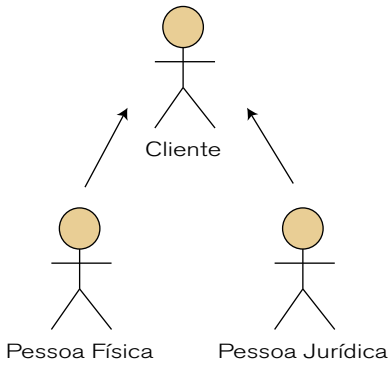
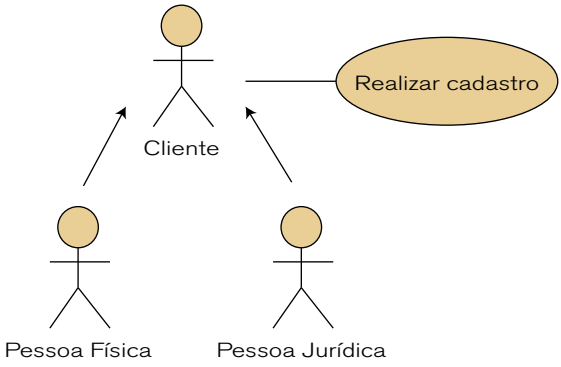


Figura 2.8 – Generalização de atores.

Em generalização podemos agrupar tipos de atores, como no exemplo da figura 2.8, onde os tipos de clientes são agrupados em um ator chamado "cliente". Isto facilita na hora de criar os casos de uso que são realizados por mais de

um ator. Neste exemplo, subentende-se que existam casos de uso que podem ser realizados tanto pelo ator "pessoa física" quanto pelo ator "pessoa jurídica". Para facilitar e simplificar o modelo, pode-se generalizar estes atores criando o ator "cliente", que executará o caso de uso. Um exemplo de como isto pode ser feito é apresentado na Figura.



No exemplo da figura pode-se entender que tanto o cliente pessoa física quanto o cliente pessoa jurídica realizam o caso de uso "criar conta bancária", Neste caso, ao invés de se representar duas realizações do mesmo caso de uso, pode-se representar utilizando apenas uma realização.

Outro exemplo de generalização pode ser citado quando os tipos de ator executam tarefas diferentes, como apresentado na figura 2.9.

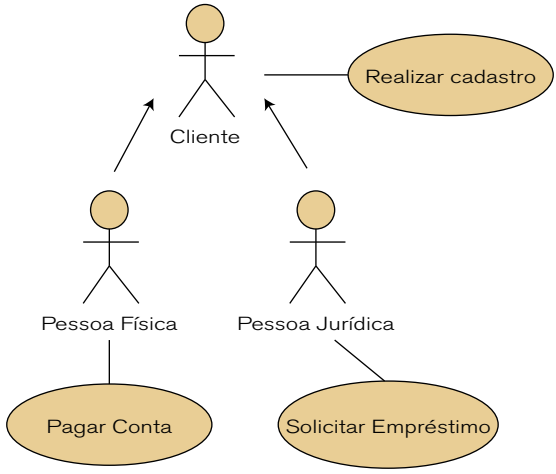


Figura 2.9 – Generalização de ator com ações específicas.

A Figura 2.9 traz um exemplo onde qualquer tipo de cliente pode criar uma conta bancária, porém apenas o cliente "pessoa jurídica" pode solicitar empréstimos e o cliente "pessoa física" é o único que pode pagar contas.

COMENTÁRIO

A generalização, também conhecida como herança, é um conceito de orientação a objetos e é utilizada em diversos diagramas da UML.

Além da generalização de atores, pode-se adotar também a generalização de casos de uso, seguindo o mesmo conceito e aplicações. Na Figura 2.10 é apresentado um exemplo de generalização de casos de uso.

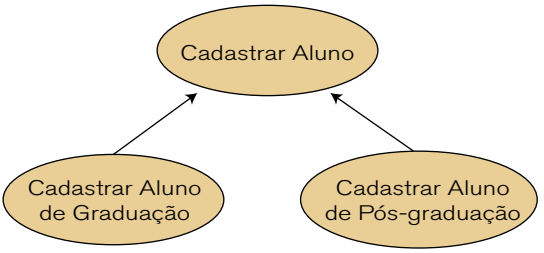


Figura 2.10 – Generalização de casos de uso.

A figura 2.10 representa uma generalização de casos de uso onde, a ação de cadastrar um aluno generaliza as outras duas ações de forma que, em casos específicos, pode-se realizar um dos casos de uso especiais ("cadastrar aluno de graduação" e "cadastrar aluno de pós-graduação") e em casos gerais pode-se realizar o caso de uso "cadastrar aluno".

Para exemplificar melhor o uso da generalização de casos de uso, a figura 2.11 exibe um diagrama onde pode-se observar a realização na forma mais geral e também mais especializada.

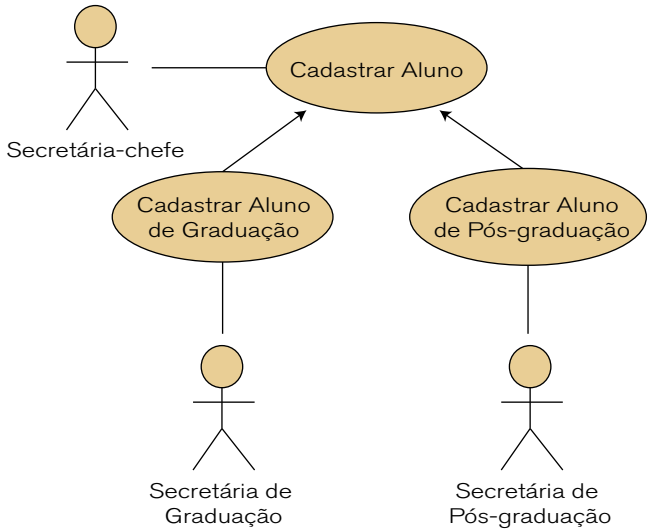


Figura 2.11 – Utilizando a generalização de casos de uso.

? PERGUNTA

Por que, no modelo demonstrado na figura 2.11, não se pode generalizar as secretárias de graduação e pós-graduação, como filhas de secretária-chefe?

Observando o exemplo apresentado na figura 2.11, têm-se três atores e três casos de uso. O ator "secretária-chefe" realiza o caso de uso "cadastrar aluno", isto significa que este ator pode cadastrar qualquer tipo de aluno, seja ele de graduação ou de pós-graduação, pois este caso de uso está generalizando os casos "cadastrar aluno de pós-graduação" e "cadastrar aluno de graduação".

Nota-se que o ator "secretária de graduação", diferentemente do ator "secretária-chefe", não realiza o caso de uso generalizado, e sim o especializado, chamado "cadastrar aluno de graduação". Sendo assim, este ator é impossibilitado

de realizar o caso de uso “cadastrar aluno de pós-graduação”. O mesmo acontece com o ator “secretária de pós-graduação”, que é impossibilitado de realizar o caso de uso “cadastrar aluno de graduação”.

De forma resumida, quando se utiliza generalização em casos de uso, pode-se deixar os modelos mais concisos e enxutos, além de possibilitar descrição mais abrangente e detalhada do requisito.

Outro termo utilizado em casos de uso são os assuntos. A figura 2.12 exibe um exemplo de diagrama de casos de uso utilizando assunto.

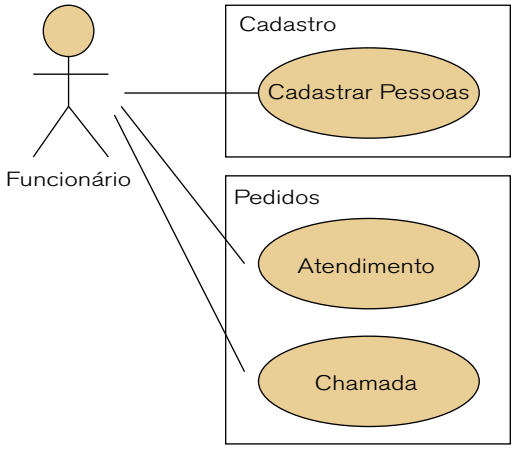


Figura 2.12 – Assunto em casos de uso.

Os assuntos do exemplo apresentado na Figura 2.12 são:

- Cadastros; e
- Pedidos.

Os assuntos são basicamente uma forma de segregar o contexto dos casos de uso, podendo também ser usado para representar diversos tipos de agrupamentos (BOOCH; RUMBAUGH; JACOBSON, 2005).

É de suma importância a exemplificação de alguns cenários para que o entendimento quanto aos casos de uso seja pleno, portanto, na próxima sessão, um estudo de caso é apresentado.

2.3.2 Exemplos de casos de uso

No exemplo aqui demonstrado tem-se uma funcionalidade de um sistema de solicitações de serviços, onde um cliente pode solicitar serviços a técnicos de uma empresa e o técnico, por sua vez, atua nesta solicitação até encerrá-la. Na figura 2.13 é apresentado o cenário onde o cliente solicita o serviço.

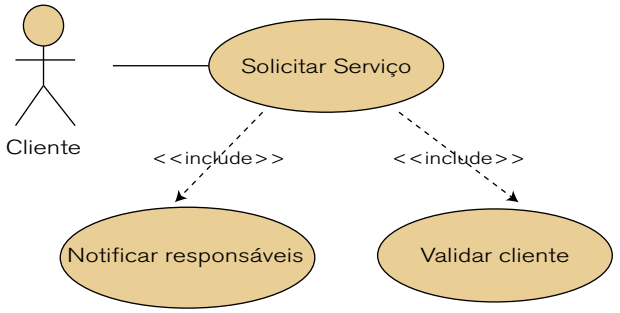


Figura 2.13 – Exemplo de caso de uso onde o cliente solicita um serviço.

O cenário apresentado na figura 2.13 demonstra que, no sistema em questão, no momento em que um cliente registra uma solicitação de serviço, o sistema automaticamente valida o cliente (verifica se o mesmo é cadastrado) e notifica os responsáveis.

Por se tratar de um *<<include>>*, a notificação dos responsáveis e a validação do cliente são ações obrigatórias na realização do caso de uso "solicitar serviço".

? PERGUNTA

Se o requisito descrito pela figura 2.13 fosse tal que o cliente pudesse optar por enviar ou não uma notificação aos responsáveis, quais seriam as alterações necessárias no diagrama para que representasse esta mudança no requisito?

Em outro cenário, a figura 2.14 demonstra um diagrama de casos de uso onde o funcionário técnico da empresa atua na solicitação feita anteriormente.

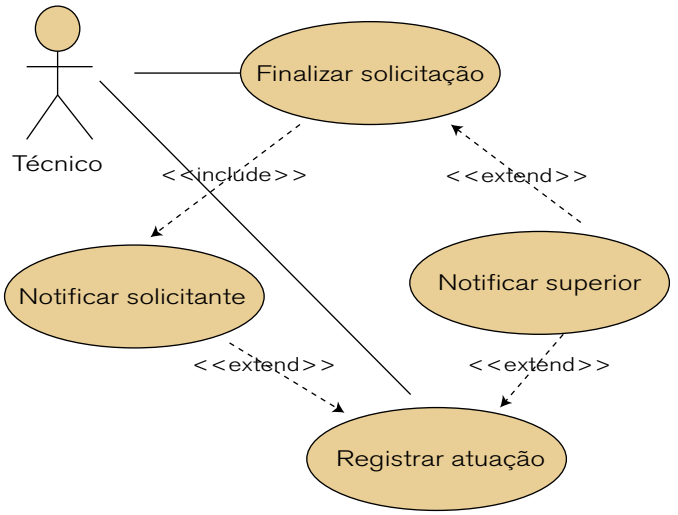


Figura 2.14 – Exemplo de caso de uso onde o técnico atua na solicitação de serviço.

O ator "técnico" pode realizar dois casos de uso:

- Registrar atuação; e
- Finalizar solicitação.

O caso de uso "registrar atuação" significa que o técnico estaria registrando um trabalho efetuado para a resolução da solicitação do cliente. Ao realizar este caso de uso, não é obrigatório que se realize também "notificar solicitante" e "notificar superior", pois estes relacionamentos são do tipo *<<extend>>*.

Na realização do caso de uso "finalizar solicitação", é necessário (obrigatório) que se notifique o solicitante. Portanto, o relacionamento entre "finalizar solicitação" e "notificar solicitante" deve ser do tipo *<<include>>*. Já o relacionamento entre "finalizar solicitação" e "notificar superior" é do tipo *<<extend>>*, o que significa que não é obrigatória a notificação do superior quando um técnico finaliza a solicitação.

ATIVIDADE

01. Crie um diagrama de casos de uso que represente a funcionalidade de cadastro de um material didático por um professor em um sistema de gestão de materiais de disciplinas.

Os requisitos do sistema são:

- O professor cadastra material didático;
- O sistema checa se o professor é vinculado à disciplina em questão;
- O sistema checa se a extensão do arquivo do material é condizente e segura;
- O professor pode escolher proteger o material cadastrado com uma senha.

REFLEXÃO

Neste capítulo foram apresentados alguns conceitos de visões de sistemas, onde pôde-se observar que dependendo da necessidade e do foco que se quer alcançar com a modelagem, utiliza-se um certo conjunto de modelos. Outro tema importante abordado aqui foi a utilização de ferramentas CASE, estas são de grande valia para analistas de sistemas, ajudando na tarefa de modelagem e até geração automática de código-fonte. A ferramenta Astah, que é utilizada para a criação dos modelos apresentados, mostra-se poderosa e pode ser de grande ajuda para estudos e utilização real.

Os casos de uso foram apresentados e sua importância destacada, assim como diversos exemplos. Estes exemplos são de grande utilidade pois além de ajudar a fixar os conceitos do diagrama de casos de uso, tem-se uma visão de como modelar sistemas simples, partindo de requisitos corriqueiros, para então se aprofundar em complexidade.

LEITURA

O livro de Ana Cristina Melo (MELO, 2006) apresenta uma série de exercícios resolvidos utilizando a modelagem UML. De forma didática, as resoluções são bem comentadas exemplificadas.

MELO, Ana Cristina. **Exercitando modelagem em UML**: 51 exercícios resolvidos. Rio de Janeiro: Brasport, 2006.

REFERÊNCIAS BIBLIOGRÁFICAS

BEZERRA, Eduardo. **Princípios de Análise e Projeto de Sistemas com UML**: Um Guia Prático Para Modelagem De Sistemas Orientados A Objetos Através Da Linguagem De Modelagem Unificada. 2. ed. Rio de Janeiro: Elsevier, 2007.

BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. **UML**: Guia do Usuário. 2. ed. Rio de Janeiro: Campus, 2005.

MELO, Ana Cristina. **Exercitando modelagem em UML:** 51 exercícios resolvidos. Rio de Janeiro: Brasport, 2006.

SOMMERVILLE, Ian. **Engenharia de Software.** 8. ed. Pearson: São Paulo, 2007.

3

Diagrama de Classes

Este capítulo aborda o diagrama responsável por modelar a estrutura de um sistema. Em muitos casos, a modelagem utilizando este diagrama é feita para que, após discutida e definida, sejam gerados automaticamente os códigos-fonte da lógica de negócios do sistema a ser desenvolvido. Os conceitos de orientação a objetos apresentados neste capítulo são de grande utilidade para o entendimento do paradigma e para futuras aplicações em desenvolvimento e modelagem de sistemas.

No final do capítulo existe um estudo de caso para que os conceitos apresentados no decorrer deste possam ser melhor fixados e posteriormente estudados.

OBJETIVOS

Este capítulo apresenta a modelagem utilizando o diagrama de classes conceitual da UML, além de importantes conceitos de orientação a objetos muito úteis para modelagem e desenvolvimento de software.

3.1 Classes e objetos

Os conceitos de classes e objetos são fundamentais para o desenvolvimento de *software* e de modelos de *software* neste paradigma. De acordo com Booch, Rumbaugh e Jacobson (2005) as classes são a estrutura mais importante na construção de sistemas orientados a objetos.

A representação de classe na UML (figura 3.1) é um retângulo dividido em três partes:

- Nome da classe;
- Atributos; e
- Métodos.

Figura 3.1 – Exemplo de classe com atributos e métodos tipados.

O nome da classe costuma iniciar com letra maiúscula, para os métodos e atributos, a letra inicial é minúscula. No exemplo da Figura 3.1 observa-se que a primeira divisão do retângulo contém o nome da classe. Na segunda divisão (a do meio), contém os atributos da classe enquanto a terceira divisão possui os métodos.

O nome da classe deve ser algo que represente uma entidade no sistema a ser modelado. Em um sistema bem simples de cadastro de clientes, pode-se destacar, por exemplo, a entidade "Cliente".

De acordo com Fowler (2005), para se entender os atributos de uma classe, pode-se fazer uma analogia à campos (variáveis pertencentes à uma estrutura)

de uma linguagem de programação. Em outras palavras, os atributos são as qualidades que um objeto possuirá quando instanciado seguindo o modelo de uma classe.

Os métodos de uma classe são as ações que um objeto instanciado pode executar, recebendo ou não parâmetros, retornando ou não valores.

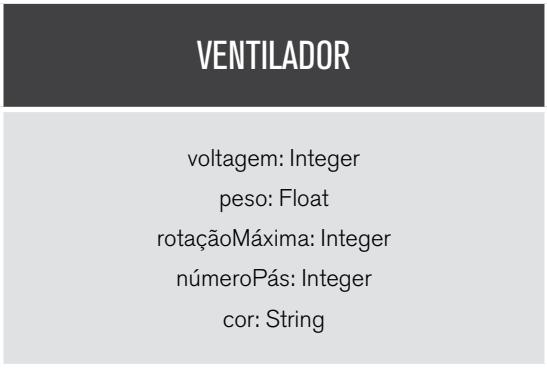
Na classe apresentada na figura 3.1, os valores de nome, atributos e métodos são fictícios e meramente ilustrativos. Como pode-se perceber, o nome da classe é "Nome", seus atributos são "atributo1" e "atributo2" e seus métodos são "método1" e "método2". Os "tipos" que aparecem na classe servem para "tipar" os campos, isto é, definir se são campos de texto, números, caracteres, objetos de outras classes e etc.

Os tipos dos métodos são basicamente o tipo do valor que será retornado. Quando não há retorno, define-se o tipo "void". No "método2" existe uma variável sendo enviada como parâmetro para a execução do método (está entre parênteses), enquanto o "método1" não precisa de parâmetros para ser executado.

! ATENÇÃO

A notação utilizada para os tipos de variáveis e atributos (String, Integer, Float, entre outros) é advinda da linguagem de programação Java, que implementa a orientação a objetos.

Um exemplo real de classe é apresentado na figura 3.2.



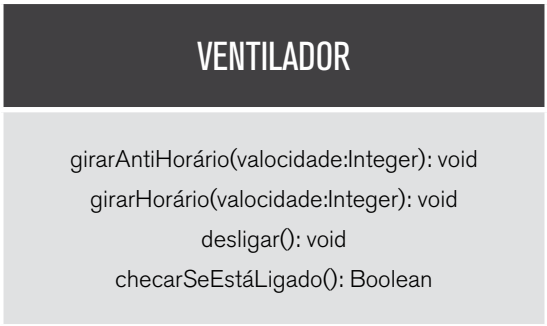


Figura 3.2 – A classe ventilador.

A classe "Ventilador" (figura 3.2) possui os seguintes atributos:

• **Voltagem:** é um número inteiro e armazena em qual voltagem que o ventilador funciona;

• **Peso:** é um número com ponto flutuante (real) e armazena o peso do ventilador;

• **Rotação máxima:** é um número inteiro que indica qual a rotação máxima que o ventilador pode alcançar (voltas por minuto);

• **Número de pás:** é um número inteiro que indica quantas pás o ventilador possui; e

• **Cor:** é do tipo texto e armazena a cor que o ventilador possui.

Os métodos da classe ventilador são:

• **Girar anti-horário:** o ventilador começa a girar em sentido anti-horário e ao executar este método, nenhum valor é retornado, porém no momento da execução do método, é necessário que se transmita a velocidade desejada;

• **Girar horário:** o ventilador começa a girar em sentido horário de acordo com a velocidade recebida por parâmetro e ao ser executado, este método não retorna valor;

• **Desligar:** o ventilador é desligado e para de girar, este método não recebe parâmetro nem retorna valor;

• **Checar se está ligado:** para ser executado, este método não necessita de parâmetro, porém o valor booleano (verdadeiro ou falso) é retornado, indicando verdadeiro para quando o ventilador está ligado e falso para quando está desligado.

? CURIOSIDADE

A escrita que aparece na Figura 3.2, onde as palavras são juntas, porém, ao iniciar uma nova palavra utiliza-se a primeira letra maiúscula chama-se CamelCase, pelas frases ficarem com ondulações parecidas com corcovas de camelo. Esta escrita é adotada como padrão em muitas linguagens de programação, como Java.

A classe, como dito anteriormente, é um modelo para a criação (instanciação) de objetos. A tabela 3.1 exibe uma série de objetos que poderiam ser instanciados da classe Ventilador com seus respectivos atributos.

Ventilador 1	voltagem	220
	peso	2,13
	rotação máxima	700
	número de pás	4
	cor	rosa
Ventilador 2	voltagem	220
	peso	1,54
	rotação máxima	650
	número de pás	5
	cor	preto

Ventilador 3	voltagem	110
	peso	1,95
	rotação máxima	800
	número de pás	3
	cor	branco

Tabela 3.1 – Objetos da classe ventilador.

Nota-se nos ventiladores da Tabela 3.1 que seus valores obedecem ao tipo definido na classe, isto é, onde na classe é definido que o atributo é um número inteiro, no objeto o valor a ser atribuído deve obedecer ao modelo, portanto, deve ser um número inteiro também, acontecendo isto para todos os atributos de um objeto instanciado.

EXEMPLO

Um exemplo clássico de classe é a classe "Carro". Seus atributos podem ser "cor", "modelo", "renavam", "placa", "ano de fabricação", entre outros. Como métodos, pode-se citar "acelerar", "frear", "trocar a marcha", "ligar o rádio", "desligar o rádio", entre outros.

Para se modelar um sistema utilizando o diagrama de classes é preciso identificar suas classes, isto é explicado com mais detalhes no estudo de caso deste capítulo. Porém, não basta identificar as classes, é necessário que se identifique também sua função e interação dentro do sistema a ser modelado. Estes conceitos de interação entre classes, importância vital para um bom modelo de *software*, são apresentados a seguir.

3.2 Relacionamentos

Os blocos de construção de um sistema são basicamente as classes, porém classes por si só não o descrevem suficientemente. Para que o entendimento do modelo seja satisfatório e cumpra seu objetivo de representar o funcionamento do sistema de *software*, é necessário que as classes interajam entre si.

As interações entre classes são chamadas de relacionamentos. Estes relacionamentos podem ser de diversos tipos, dependendo da função que a classe exerce no sistema a ser modelado.

Os tipos de relacionamento entre classes em um diagrama de classes são:

- Herança;
- Associação;
- Agregação;
- Composição; e
- Dependência.

Estes quatro tipos de relacionamentos são detalhados nos próximos itens.

3.2.1 Herança

Para se identificar um relacionamento de herança (também conhecido como generalização ou especialização) entre duas ou mais classes, costuma-se utilizar da pergunta: "é um(a)?". Ao fazer esta pergunta entre duas classes, se a resposta for sim, pode-se então considerar o emprego de um relacionamento de herança.

Na herança da vida real as características dos pais são passadas aos filhos. Os filhos, herdando as características dos pais, desenvolvem as suas próprias e também seu próprio comportamento.

Em sistemas orientados a objetos, tem-se também a classe pai e a classe filha. Em algumas linguagens de programação, um filho pode ter mais de um pai, como é o caso de *C++ e Python*, isto é chamado de herança múltipla. A herança simples é quando uma classe só pode ter uma classe pai, como no caso da linguagem Java.

Para exemplificar este tipo de relacionamento, são utilizadas classes de animais, imaginando-se que um sistema necessite das classes de animais para seu

funcionamento, portanto, têm-se as classes de alguns tipos de animais em seu modelo, como exibido na figura 3.3.

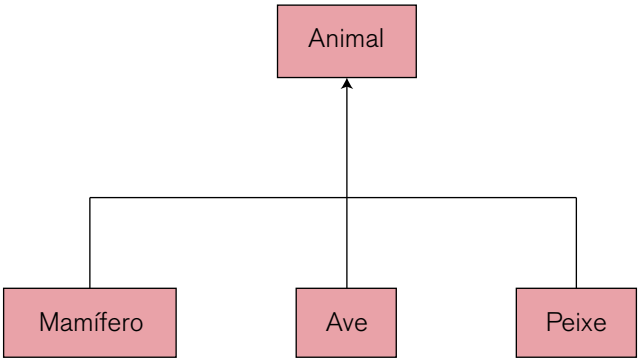


Figura 3.3 – Classes da herança de animais.

O exemplo da figura 3.3 considera apenas os nomes das classes pois o intuito é de identificação da herança, na figura 3.4 o modelo está completo.

Para a identificação da herança de animais, a pergunta proposta "é um(a)?" é utilizada. Portanto, quando se dirige a pergunta de "MamíferoTerrestre" para "Animal", a resposta é sim, portanto, "MamíferoTerrestre" é filho de "Animal" (todo mamífero terrestre é um animal). Já quando dirigimos a pergunta de "Animal" para "Ave", a resposta é não, portanto, "Animal" não é filho de "Ave".

No modelo apresentado na figura 3.3, "MamíferoTerrestre", "Ave" e "Peixe" são filhos de "Animal", pois todos são obrigatoriamente animais. Além disto, a herança pode conter vários níveis. Poderíamos modelar mais classes como filhas de "MamíferoTerrestre", filhas de "Ave" e filhas de "Peixe", e assim por diante.

Outra propriedade da herança entre classes é que uma classe filha pode sobrescrever os métodos da classe pai, se desejar que ele seja executado de forma específica (o nome e o tipo dos parâmetros devem ser idênticos aos do método a ser sobrescrito). Na figura 3.4, um exemplo de sobrescrição de método é apresentado na classe "Peixe", onde o método "respirar" é sobrescrito, visto que os animais do tipo peixe executariam este método (respirariam) de uma forma diferente que seu pai (animal).

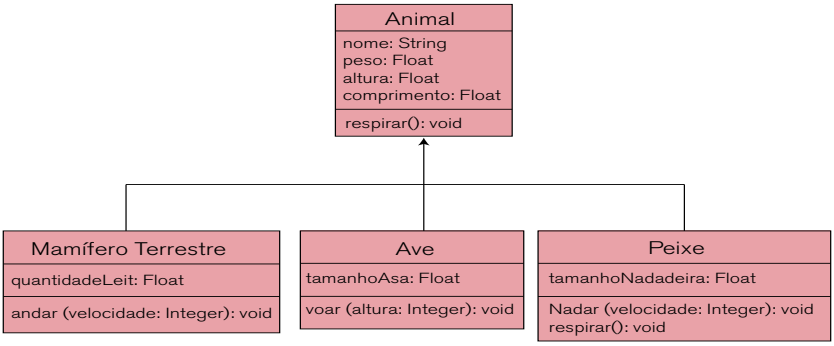


Figura 3.4 – Herança entre animais com as classes completas.

Cada classe herdeira possui todos os atributos e métodos que são herdados da classe pai, somados aos seus específicos. A classe pai possui apenas os seus específicos. Se uma das classes filho, por exemplo "Ave", tivesse seu próprio filho, este herdaria tudo desde "Animal", somando com tudo da classe "Ave", somando com seus atributos e métodos próprios. Para melhor exemplificar a herança, a tabela 3.2 exibe objetos instanciados dos filhos da classe "Animal".

Ave 1	nome	Piu-Piu
	peso	0,55
	altura	0,35
	comprimento	0,25
	tamanho Asa	0,28

Ave 2	nome	Blue
	peso	1,65
	altura	0,30
	comprimento	0,35
	tamanho Asa	0,40
Mamífero terrestre1	nome	Tom
	peso	12,8
	altura	0,55
	comprimento	0,70
	quantidade Leite	4
Mamífero terrestre2	nome	Rex
	peso	25,45
	altura	0,80
	comprimento	1,15
	quantidade Leite	7

Peixe1	nome	Adam
	peso	0,07
	altura	0,03
	comprimento	0,08
	tamanho Nadadeira	0,02
Peixe2	nome	Zeus
	peso	0,15
	altura	0,07
	comprimento	0,12
	tamanho Nadadeira	0,06

Tabela 3.2 – Objetos das classes filhas de Animal.

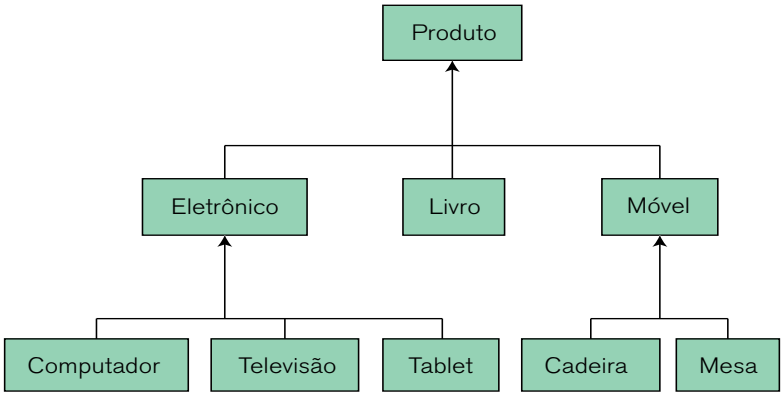
EXERCÍCIO RESOLVIDO

Treine o relacionamento de herança utilizando a pergunta "é um(a)?" entre as seguintes classes de uma loja de departamentos (utilize apenas o nome das classes):

- Televisão;
- Eletrônico;
- Cadeira;
- Computador;
- Produto;

- Livro;
- Móvel;
- Mesa; e
- Tablet.

Resolução:



Uma aplicação real de herança em um modelo de sistema é demonstrada no estudo de caso deste capítulo. A seguir, outros tipos de relacionamento entre classes são apresentados.

3.2.2 Associação

O relacionamento de associação é a base da interação entre as classes de um projeto de sistema. A associação entre classes pode ser:

- Simples;
- De agregação;
- De composição; e
- De dependência.

Este item trata da associação simples, as outras modalidades são apresentadas nos itens seguintes.

As associações são relações entre classes que possuem um rótulo e uma cardinalidade (também conhecida como multiplicidade). O rótulo é um nome descritivo dado ao relacionamento para que o entendimento da relação entre

as classes seja mais objetiva. A cardinalidade é o modo com que as classes são relacionadas no sistema, o que é de grande importância para a implementação do mesmo. Uma cardinalidade mal especificada pode trazer muitas consequências para o desenvolvimento e operação do *software.*

Na UML, para se representar uma associação simples é utilizada uma linha reta e contínua que conecta as duas classes relacionadas, como exibido na figura 3.5.

Figura 3.5 – Exemplo de associação simples entre classes.

As classes "Cliente" e "Produto" estão relacionadas por uma associação simples no exemplo da Figura 3.5. A cardinalidade é representada pelos símbolos "1" e "*" e o rótulo pelo texto "compra".

O rótulo indica o porquê de as classes estarem relacionadas e a seta que o segue indica a direção do relacionamento. Neste caso (figura 3.5), pode-se entender que na estrutura do sistema a ser modelado, o cliente compra produto, portanto deve-se saber qual cliente comprou um certo produto e quais produtos foram comprados por um cliente.

Para que se chegue ao entendimento da cardinalidade e como ela atua no relacionamento, é preciso que se saiba que os tipos de cardinalidade existentes são:

- 1;
- *;
- 0..*;
- 1..*; e
- 1..1.

O asterisco (*) representa a palavra muitos enquanto os números 0 e 1 representam quantidades definidas. Na ocasião onde se tem dois símbolos separados por dois pontos (0..*, 1..* e 1..1) o primeiro significa a quantidade mínima e o segundo a quantidade máxima. Em ocasiões onde só se tem um símbolo, apenas a quantidade máxima está especificada. Entende-se então que, ao utilizar apenas o asterisco como cardinalidade (como no exemplo da figura 3.6), qualquer quantidade pode ser associada.

Existem casos em que se utiliza de constantes numéricas (diferentes de zero e um), isto significa que os objetos de uma classe se relacionam em quantidades exatas (ou faixas de valores definidos) com os objetos de outra classe. Como exemplo, podemos ter a cardinalidade 5 ou 3..8, representando respectivamente um relacionamento com exatamente cinco objetos da outra classe e outro que pode variar de 3 a 8 objetos. O uso destas constantes não é muito comum, mas pode ser necessário especificar relacionamentos desta maneira dependendo da funcionalidade e estrutura do *software* que está sendo modelado.

No caso da figura 3.5, um cliente pode comprar muitos produtos mas um produto só pode ser comprado por no máximo um cliente (as quantidades mínimas não estão especificadas).

Outro exemplo de associação simples é apresentado na figura 3.6, onde são utilizadas outras cardinalidades.

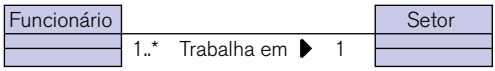


Figura 3.6 – Exemplo de associação simples entre funcionário e setor.

As cardinalidades do relacionamento da figura 3.6 podem ser entendidas como:

- Um funcionário trabalha em, no máximo, um setor; e
- Um setor pode ter um ou mais funcionários ligados à ele;

Caso a cardinalidade de setor ao invés de 1 fosse 1..1, significaria que um funcionário deve estar obrigatoriamente associado a um setor.

No estudo de caso deste capítulo há mais exemplos de associação simples. A seguir são apresentados dois tipos de associação chamados de todo-parte: a agregação e a composição.

3.2.3 Agregação

Este relacionamento é uma associação diferente da associação simples apresentada no item anterior. A agregação considera as classes relacionadas como sendo uma o "todo" e a outra a "parte". Por tal motivo também é conhecida, juntamente com a composição, como relacionamento todo-parte.

O símbolo que identifica a agregação é apresentado na figura 3.7. Um losango vazado é utilizado para identificar qual é a classe "todo" e qual é a classe "parte". Como pode-se observar, o losango indica a classe "todo".

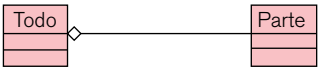


Figura 3.7 – Exemplo básico de agregação.

Na figura 3.7 o relacionamento apresentado indica que um objeto da classe "Todo" possui um conjunto de objetos da classe "Parte".

Na agregação pode-se incluir também, assim como na associação, a cardinalidade e o rótulo do relacionamento.

Um exemplo real é apresentado na Figura 3.8, onde pode-se observar um pedido de compra com os itens que serão comprados.

Figura 3.8 – Exemplo real de agregação.

Cada pedido do exemplo da figura 3.8 possui itens. No caso da agregação, os itens podem existir no sistema mesmo antes de fazerem parte de um pedido. Em outras palavras, a existência de "Item" independe da existência de seu "Pedido".

3.2.4 Composição

A representação gráfica da composição é feita com um losango cheio (pintado). Um exemplo desta representação é apresentado na figura 3.9.

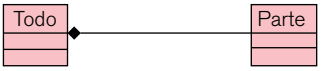


Figura 3.9 – Exemplo de composição.

A utilização da composição é muito similar à agregação, porém, com uma diferença principal: a existência da "parte" não faz sentido se o "todo" não existir.

CONCEITO

Na agregação, o objeto parte pode pertencer a mais de um todo, enquanto na composição, o objeto parte pode pertencer somente a um todo. Na composição, as partes são destruídas como consequência na destruição do seu respectivo todo (PODESWA, 2005).

Para melhor entendimento de como a composição funciona em modelagem de sistemas, a figura 3.10 exibe um exemplo real de utilização.

Figura 3.10 – Exemplo real do uso de composição.

Um módulo de galeria de fotos simplificado é exibido na figura 3.10. Segundo a modelagem apresentada, uma galeria é composta (por isso o termo composição) de fotos. Pode-se chegar a duas conclusões com os conceitos apresentados anteriormente:

- a existência da foto não faz sentido se não for para compor uma galeria; e
- uma galeria, quando destruída no sistema, as fotos também são.

Os relacionamentos todo-parte podem ser muito úteis em modelagem de sistemas, podendo representar diversas situações reais de agregação ou composição.

3.2.5 Dependência

A dependência de classes é utilizada quando, para que ações sejam executadas por um objeto, é necessário que exista um objeto de outra classe.

Um exemplo prático deste uso é apresentado na figura 3.11, onde um caixa de banco possui o método de depositar (no caso, dinheiro). Este método recebe dois parâmetros: o valor a ser depositado e a conta em que o valor será creditado. Para que a conta possa ser especificada, é necessário que um objeto da classe "Conta" seja passado como parâmetro para o método, portanto, a classe "Caixa" tem uma relação de dependência com a classe "Conta".

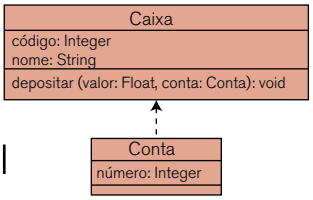


Figura 3.11 – Exemplo de relacionamento de dependência entre classes.

Como pode ser observado na figura 3.11, o relacionamento de dependência é representado por uma linha tracejada e uma seta. A seta indica a classe dependente, neste caso, a classe "Caixa" depende da classe "Conta".

Um estudo de caso de modelagem utilizando diagrama de classes é apresentado no item seguinte, com o objetivo de fixar os conceitos de orientação a objetos apresentados e também os conceitos da modelagem utilizando a UML.

3.3 Estudo de caso

Este estudo de caso tem como objetivo a fixação dos conceitos apresentados neste capítulo. A abordagem que será utilizada é a de modelagem de um sistema de uma escola. O sistema da escola é apresentado de forma simplificada para atuar de forma didática.

O modelo do estudo de caso apresentado na figura 3.12 é discutido neste item. Este modelo possui apenas os nomes das classes e seus atributos, informação suficiente para o entendimento e aprendizado do modelo.

? CURIOSIDADE

No momento do desenvolvimento do sistema orientado a objetos, os relacionamentos de muitos para muitos, também conhecidos como N para N, possuem uma implementação onde cada um dos objetos possui uma lista do objeto com o qual está se relacionando. Ao utilizar bancos de dados relacionais para o desenvolvimento, a solução para este relacionamento é de forma diferente, criando-se uma tabela entre as outras duas relacionadas para que se possa fazer a navegação entre ambas.

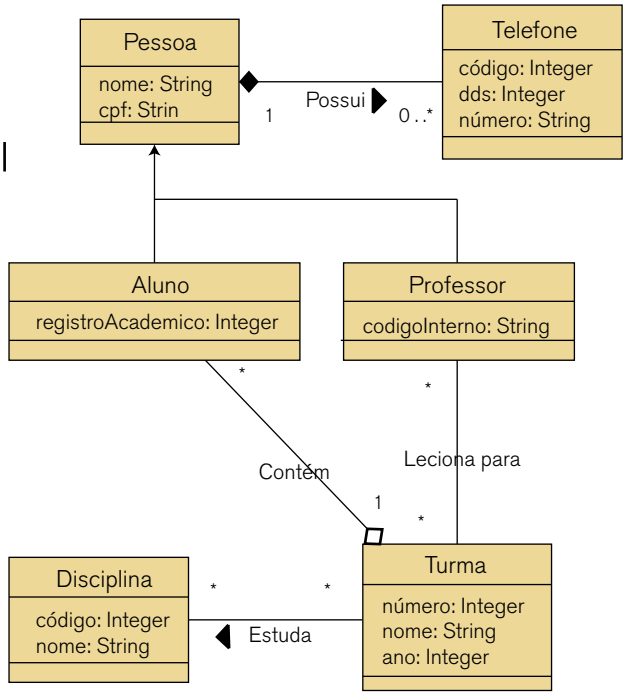


Figura 3.12 – Estudo de caso de diagrama de classes.

No estudo de caso apresentado (figura 3.12) há uma herança entre "Pessoa" e "Aluno" e "Pessoa" e "Professor". Sendo assim, a composição entre "Pessoa" e "Telefone" é válida para ambos os filhos ("Professor" e "Aluno"). Em outras palavras, alunos e professores terão a possibilidade de ter telefones incluídos em seu cadastro. Neste caso, a existência de um telefone só faz sentido se seu "dono" também existir, para que não haja telefones cadastrados no sistema sem uma relação com algumas das pessoas também cadastradas.

Ambas as classes "Professor" e "Aluno" têm relacionamento com a classe "Turma", porém de maneiras diferentes. O professor leciona para diferentes turmas, enquanto o aluno é parte de apenas uma turma.

As disciplinas que são estudadas por uma turma podem ser várias, por isto a cardinalidade (*). Uma mesma disciplina pode ser estudada por mais de uma turma, portanto, o outro lado da associação simples também possui cardinalidade (*). Neste caso, pode-se optar por detalhar o modelo criando uma classe associativa.

CONEXÃO

Uma explicação sucinta e didática sobre classes associativas é apresentada no seguinte *link*: http://www.inf.ufpr.br/silvia/ESNovo/UML/pdf/ModeloConceitualAl.pdf

Este estudo de caso simples pode servir de guia para o desenvolvimento de novos modelos de *software* orientados a objeto, porém, trata-se de um modelo conceitual. Em modelos conceituais não são necessários identificar os tipos das variáveis e as dependências, estes e outros conceitos são abordados em diagramas de classes de projeto, no capítulo 5.

ATIVIDADE

01. Crie um diagrama de classes que modele um sistema bancário simplificado, onde devem ser abordados os clientes, as contas, e os gerentes de conta. Utilize apenas os nomes das classes.

REFLEXÃO

A apresentação de conceitos importantes de orientação a objetos, guiados pelo diagrama de classe, foram apresentados neste capítulo. A construção de um sistema informatizado passa pela definição de suas estruturas básicas, sendo assim, o diagrama de classes dá o suporte necessário para que esta estrutura seja montada e modelada de acordo com o paradigma orientado a objetos. Classes, objetos e os diversos relacionamentos apresentados dão suporte à criação de modelos de sistemas simples a complexos.

LEITURA

O segundo capítulo do livro "UML for the IT Business Analyst" (UML para o analista de negócios de TI) de Howard Podeswa (2005) apresenta de forma sucinta, objetiva e didática conceitos de orientação a objetos direcionados à analistas de TI.

PODESWA, Howard. UML for the IT Business Analyst: A Practical Guide to Object-Oriented Requirements gathering. 1. ed. Boston: Thomsom, 2005.

REFERÊNCIAS BIBLIOGRÁFICAS

BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. **UML**: Guia do Usuário. 2. ed. Rio de Janeiro: Campus, 2005.

FOWLER, Martin. **UML Essencial:** Um breve guia para a linguagem-padrão de modelagem de objetos. 3. ed. Porto Alegre: Bookman, 2005.

PODESWA, Howard. **UML for the IT Business Analyst:** A Practical Guide to Object-Oriented Requirements gathering. 1. ed. Boston: Thomsom, 2005.

4

Descrição de Casos de Uso e Outros Diagramas da UML

Este capítulo apresenta um aprofundamento nos casos de uso já apresentados anteriormente com um conceito muito importante chamado de descrição de casos de uso. As descrições de casos de uso geralmente contém mais informações e de forma mais detalhada que os diagramas apresentados no capítulo 2, portanto, é comum que se utilize das duas técnicas em conjunto. Alguns outros diagramas também são apresentados neste capítulo, cada um com seu objetivo e foco principal, permitindo que se possa modelar *software* com mais qualidade e apresentar modelos concisos e condizentes com o objetivo final.

OBJETIVOS

Ao final deste capítulo, o aluno conhecerá:

- Conceitos de descrição de casos de uso;
- Os diagramas de interação da UML;
- O diagrama de estados; e
- O diagrama de atividade;

4.1 Descrição de casos de uso

Os casos de uso, como vistos no capítulo 2, são os principais componentes do diagrama de casos de uso. Este diagrama é limitado quanto aos procedimentos de execução e detalhamento dos casos de uso, por este motivo, tem-se a descrição destes.

Esta descrição é feita de forma textual e detalha o funcionamento de um caso de uso. De acordo com Larman (2007), manter o foco nos diagramas de caso de uso e esquecer sua forma textual é um erro comum para quem está iniciando em análise de sistemas. Deste modo, pode-se concluir que as narrativas textuais dos casos de uso são mais importantes e contém mais informação que os diagramas.

Para cada caso de uso é feita uma descrição independente. Cada descrição é dividida em duas partes:

- Cabeçalho; e
- Descrição.

Conforme exibido na tabela 4.1, o cabeçalho possui os seguintes itens:

- Nome do caso de uso;
- **Objetivo:** descrição sucinta do objetivo do caso de uso no contexto do sistema;
- **Pré-condição:** as condições que devem existir para que o caso de uso que está sendo descrito possa ser realizado;
- **Pós-condição:** são as regras cumpridas pelo caso de uso que está sendo descrito e que podem ser utilizadas como pré-condição de procedimentos futuros.

Nome:	
Objetivo:	C
Pré-condição:	A
	B
	E
	Ç
Pós-condição	A
	L
	H
	O

Tabela 4.1 – Formato do cabeçalho da descrição de casos de uso.

São dois os tipos de descrição de casos de uso, cada um com um objetivo específico:

- Descrição não expandida; e
- Descrição expandida.

A descrição não expandida é utilizada para descrever requisitos (ou casos de uso) que são menos complexos, que possuem poucas regras e que não utilizem mecanismos de outros casos de uso. Outro cenário que demanda a utilização deste tipo de descrição é quando o caso de uso é de conhecimento de todos os envolvidos.

De forma prática, a descrição não expandida trata-se de um texto curto que apresenta de forma sucinta o que o caso de uso deve fazer (seus objetivos). Um exemplo desta descrição é exibido na tabela 4.2.

COMENTÁRIO

Utilize a descrição não expandida para casos de uso corriqueiros e de fácil entendimento. Desta forma diminui-se o cansaço com leitura excessiva, além de evitar documentação desnecessária do sistema.

CABEÇALHO	
Nome: Cadastrar cliente. Objetivo: Registrar os dados de um novo cliente. Pré-condição: Não há. Pós-condição: Efetuar venda.	
O cadastro do cliente trata-se da inclusão de seus dados em formulário e armazenamento das informações no sistema.	DESCRIÇÃO

Tabela 4.2 – Exemplo de descrição de caso de uso não expandida.

A descrição expandida é utilizada em casos de uso mais complexos, por se tratar de uma descrição detalhada. Um passo-a-passo da realização do caso de uso é exibida nesta descrição, que possui seu fluxo normal e seus fluxos alternativos.

O fluxo normal de uma descrição de casos de uso é a maneira comum com que este é realizado. Seus fluxos alternativos são acionados caso uma particularidade aconteça durante sua realização.

As particularidades ou exceções, para que sejam pertinentes, devem ser de ordem lógica. Uma indisponibilidade no sistema, uma indisponibilidade de bancos de dados, por exemplo, não são contempladas nesta descrição (exemplos de exceções são exibidos na tabela 4.3).

Uma descrição expandida deve contemplar de forma detalhada o fluxo normal e também os fluxos alternativos. Um exemplo de descrição de caso de uso expandida é apresentado na tabela 4.3.

Nome: Efetuar venda. Objetivo: Registrar os dados da venda de produtos a um cliente. Pré-condição: Ter acesso ao sistema. Pós-condição: Gerar boleto de cobrança. Produtos sairão do estoque.	CABEÇALHO
Fluxo normal: 1. Sistema apresenta interface de venda 2. Vendedor informa produtos a serem vendidos 3. Sistema obtém dados de produtos 4. Vendedor escolhe o cliente 5. Sistema obtém dados do cliente 6. Vendedor conclui a venda 7. Sistema registra venda 8. Sistema inclui "Gerar cobrança" 9. Sistema inclui "Atualizar estoque" 10. Sistema emite comprovante de venda 11. Sistema encerra caso de uso Fluxo alternativo: 4. Vendedor escolhe cliente 4.1 Cliente não cadastrado 4.1.1 Sistema inclui "Cadastrar cliente" 4.1.2 Sistema retorna para item 5 6. Vendedor conclui venda 6.1 Vendedor cancela venda 6.1.1 Sistema retorna para item 1	DESCRIÇÃO

Tabela 4.3 – Exemplo de descrição de caso de uso expandida.

A inclusão de outros casos de uso pode ocorrer na descrição (como nos itens 8 e 9 do fluxo normal da tabela 4.3), porém as descrições destes que estão sendo incluídos devem ser feitas de forma separada. É importante frisar que cada descrição contempla apenas um caso de uso.

Na descrição expandida é o sistema quem realiza a primeira e a última tarefa, em outras palavras, quem inicia e finaliza o caso de uso é o sistema.

? CURIOSIDADE

A descrição do caso de uso pode conter um esboço da tela do sistema que será utilizada para sua realização, desta forma, adiciona-se mais valor à descrição.

É comum que durante a descrição de um caso de uso se identifique a necessidade de criação de novos destes. Quando isto ocorre, é necessário que se reveja o modelo do *software* para que este contemple a nova necessidade.

4.2 Diagramas de Interação

Os diagramas de interação modelam a interação entre objetos e seus relacionamentos. Isto inclui as mensagens que podem ser trocadas entre eles (BOOCH; RUMBAUGH; JACOBSON, 2005).

Cada diagrama de interação contempla apenas um caso de uso, provendo uma visualização da interação entre seus objetos.

CONEXÃO

O *site* da *Microsoft Developer Network* traz informações sobre modelagem de *software*, apesar de ser direcionado ao uso do *software* Visual Studio, conceitos são apresentados de forma didática e aplicada.
https://msdn.microsoft.com/pt-br/library/dd409445.aspx

São dois os tipos de diagramas de interação:

- Diagrama de sequência; e
- Diagrama de comunicação.

De acordo com Booch, Rumbaugh e Jacobson (2005) os diagramas de interação são constituídos basicamente por:

- Papéis ou objetivos;
- Comunicações ou vínculos; e
- Mensagens.

Os diagramas de interação são explicados e exemplificados a seguir.

4.2.1 Diagrama de sequência

Como visto anteriormente, o diagrama de classes representa os dados do sistema e o diagrama de casos de uso, o uso do sistema. O diagrama de sequência tem uma junção destes dois mundos, representando o uso do sistema com relação aos dados que são acessados e como é feita a interação entre eles. Este diagrama pode ser dividido em dois tipos:

- Diagrama de sequência; e
- Diagrama de sequência de sistema.

O diagrama de sequência de sistema é um caso particular de diagrama de sequência e é explicado logo em seguida ao diagrama de sequência.

A ordenação temporal é o foco do diagrama de sequência (BOOCH; RUMBAUGH; JACOBSON, 2005).

Um exemplo simples de diagrama de sequência é exibido na figura 4.1. Pode-se observar retângulos organizados na horizontal, na parte superior do diagrama, estes são os objetos que participam da interação. Abaixo de cada objeto, existe uma linha vertical tracejada chamada de "linha da vida". O eixo vertical representa o tempo, portanto, esta linha representa a duração da vida de um objeto.

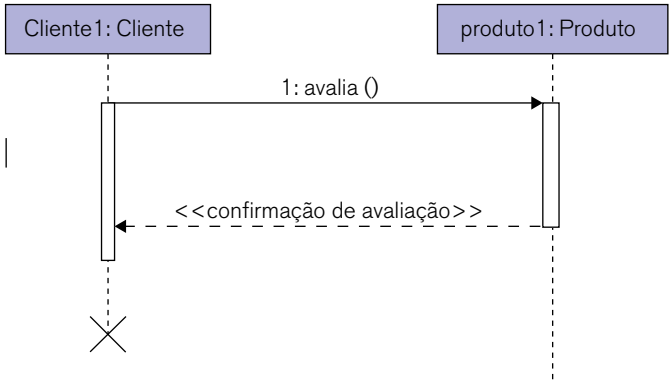


Figura 4.1 – Exemplo simples de diagrama de sequência.

Os dois objetos que participam deste diagrama são:

- **Cliente 1:** objeto da classe Cliente; e
- **Produto 1:** objeto da classe Produto.

A seta que parte do cliente para o produto indica a mensagem enviada entre os dois objetos, enquanto a seta tracejada que parte de produto para cliente representa a resposta da interação.

Os retângulos observados nas linhas de vida representam a duração das interações. O símbolo de destruição do objeto (fim da linha da vida) é representado por um X.

CONEXÃO

O documento em PDF do *link* apresentado (disponibilizado pela *wiki* da PUC-RIO traz, em slides, informações aprofundadas sobre o diagrama de sequência, com detalhes interessantes). http://www.les.inf.puc-rio.br/wiki/images/e/ef/Aula02-diagrama_sequencia.pdf

Os objetos podem trocar mensagens de criação e destruição de objetos entre si e uma mensagem pode ser enviada de um objeto para ele próprio. As mensagens são métodos, onde podem ser passados parâmetros para sua execução.

O exemplo apresentado na figura 4.1, apesar de ser muito simples, dá uma visão de artefatos que podem ser utilizados neste tipo de diagrama.

Um exemplo mais complexo é apresentado na figura 4.2, onde se tem incluídos alguns conceitos como a criação e destruição de objetos e a utilização de atores (mais informações sobre os atores são apresentadas no capítulo 2 deste livro).

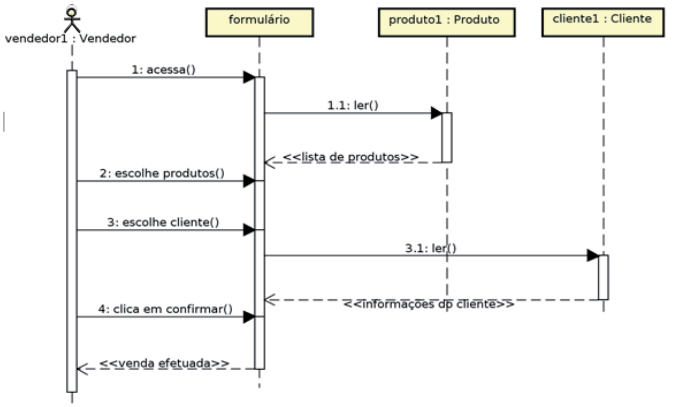


Figura 4.2 – Exemplo de diagrama de sequência.

Na figura 4.2 pode-se observar a sequência de execução do sistema na ação de um vendedor para registrar uma venda. A leitura do diagrama é feita de cima para baixo e da esquerda para a direita:

1. Vendedor acessa a interface do sistema;
2. Sistema lê a lista de produtos e os exibe ao vendedor;
3. Vendedor escolhe os produtos a serem vendidos;
4. Vendedor escolhe o cliente comprador (neste caso, o sistema ainda não leu as informações de cliente);
5. Sistema busca informações do cliente selecionado e as exibe ao vendedor;
6. Vendedor confirma a venda;
7. Sistema envia mensagem de venda efetuada ao vendedor.

O diagrama de sequência de sistema representa a interação entre o usuário e o sistema (entradas e saídas do sistema) de forma semelhante ao diagrama de sequência, porém, sem detalhar os objetos participantes da realização do caso de uso.

Um exemplo de diagrama de sequência de sistema é apresentado na figura 4.3, onde o vendedor efetua uma venda utilizando o sistema.

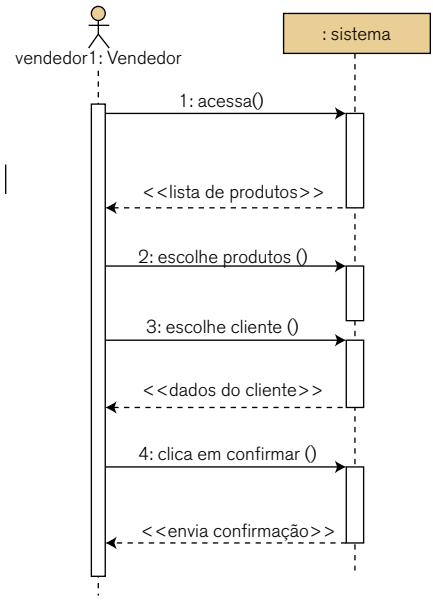


Figura 4.3 – Diagrama de sequência de sistema.

Nota-se na figura 4.3 que o diagrama exibe apenas as informações que entram e saem do sistema, sem detalhamento de quais objetos estão participando, portanto, trata-se de um caso particular de diagrama de sequência onde apenas a interação do sistema com o mundo externo é modelada.

4.2.2 Diagrama de comunicação

Antigamente chamado de diagrama de colaboração, este diagrama foca na organização dos objetos participantes de uma interação (BOOCH; RUMBAUGH; JACOBSON, 2005).

Enquanto o diagrama de sequência apresenta uma visão da interação ao longo do tempo, o diagrama de comunicação apresenta uma visão de interação dos objetos no contexto do sistema.

? CURIOSIDADE

Engenharia de produção em desenvolvimento de *software* é fazer a codificação (produto) a partir de um modelo (projeto). A engenharia reversa é criar o modelo (projeto) a partir do código-fonte (produto). Existem *softwares* que auxiliam a criação de modelos a partir da análise de código-fonte.

A figura 4.4 traz um exemplo de diagrama de comunicação onde um objeto do tipo Computador envia uma mensagem para a impressão de um arquivo a um objeto Servidor de Impressão. Caso a impressora esteja ocupada, o Servidor de Impressão envia uma mensagem para armazenamento do arquivo na fila de impressão e caso a Impressora (objeto) esteja desocupada, o arquivo é enviado para ela para que seja impresso.

CONEXÃO

O arquivo disponibilizado no *link* a seguir, desenvolvido por Aristófanes Corrêa Silva, possui uma descrição detalhada sobre os diagramas de comunicação (SILVA, 2015).
http://www.deinf.ufma.br/~acmo/MOO_Com.pdf

Apesar de manter o sequenciamento da ordem do envio das mensagens, o diagrama de colaboração não tem o enfoque temporal que o diagrama de sequência possui.

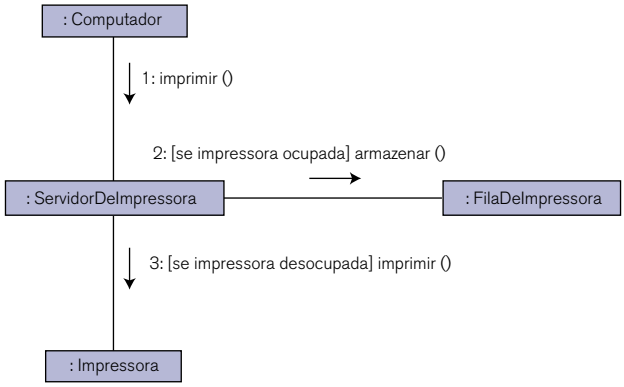


Figura 4.4 – Diagrama de comunicação (ERIKSSON et al., 2004).

Pode-se observar com clareza a interação entre objetos do exemplo da figura 4.4, assim como alguns elementos da notação do diagrama de comunicação.

A ligação entre os objetos é feita pela linha sólida interligando dois objetos. As mensagens são representadas pelas setas e podem ter respostas (assim como no diagrama de sequência). Na realidade, o diagrama de comunicação tem a notação muito parecida com o diagrama de sequência, porém, sua formatação é um pouco diferente para enfatizar outros aspectos do sistema.

EXEMPLO

Caso um objeto possua muitas entradas, ou seja, receba muitas mensagens de outros objetos, esta representação pode ficar confusa no diagrama de comunicação, pois todas as mensagens são representadas na mesma ligação. A vantagem de se utilizar o diagrama de comunicação neste caso é ter uma visão abrangente de todas as mensagens enviadas a um único objeto sem ter que percorrer toda a linha do tempo do diagrama de sequência.

4.2.3 Diagrama de estados

Também conhecido como máquina de estados, este diagrama é utilizado para a apresentação dos estados que um objeto ou um caso de uso pode assumir, assim

como os eventos que fazem com que o estado seja alterado, sendo uma técnica muito utilizada para tratar as restrições do sistema que são impostas pelos requisitos.

O exemplo exibido na figura 4.5 traz um diagrama onde o objeto em questão pode assumir três estados. O início do diagrama é representado pela esfera preta, sendo o "estado 1" o estado inicial do objeto, antes de qualquer ocorrência. O objeto assume o "estado 2" quando o "evento 1" acontece. O estado final do objeto, representado pela esfera preta com um círculo externo, é assumido quando o "evento 2" acontece. Há também a possibilidade de um objeto retornar ao mesmo estado (auto-transição). Portanto, este diagrama é composto basicamente por:

- Início;
- Estado: nome do estado e atividade;
- Transição: mudança de estado; e
- Fim.

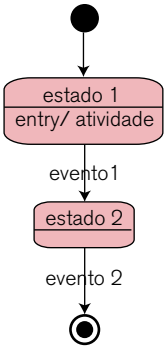


Figura 4.5 – Exemplo simples de diagrama de estados.

Um estado, representado por um retângulo, possui duas partes:

- Nome do estado;
- **Atividade:** o que está sendo executado enquanto o objeto permanece no estado;

A descrição das transições é dividida em três partes:

- **Evento:** a ocorrência para que aquele estado seja alterado;
- **Guarda:** condição lógica que deve ser satisfeita para que o estado seja alterado;
- **Ação:** procedimento do sistema responsável por efetivar a alteração de estado.

LEITURA

O capítulo 22 do livro UML: Guia do Usuário traz uma especificação de máquinas de estados complexas, com conceitos específicos e avançados sobre o uso deste diagrama, como sub estados sequenciais, estados concorrentes, entre outros conceitos (BOOCH; RUMBAUGH; JACOBSON, 2005).

Um exemplo prático e simples de diagrama de estados é exibido na figura 4.6, onde um objeto produto pode assumir três estados em um sistema.

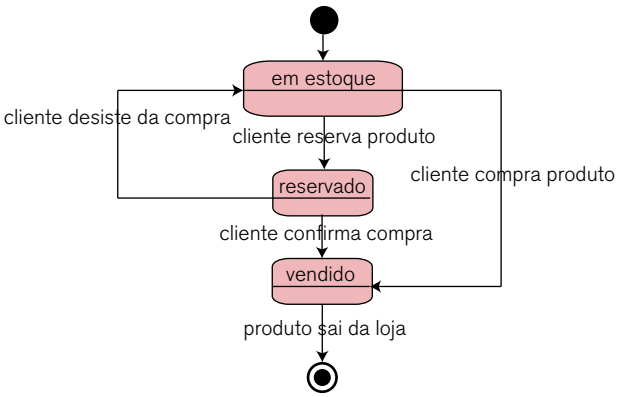


Figura 4.6 – Exemplo prático de diagrama de estados.

4.2.4 Diagrama de atividade

Os diagramas de atividade são meramente gráficos de fluxo e podem remeter a fluxogramas, porém, uma diferença essencial os distinguem: os diagramas de atividade podem modelar processos paralelos.

Para melhor entendimento deste diagrama, é feita uma discussão sobre o exemplo apresentado na figura 4.7.

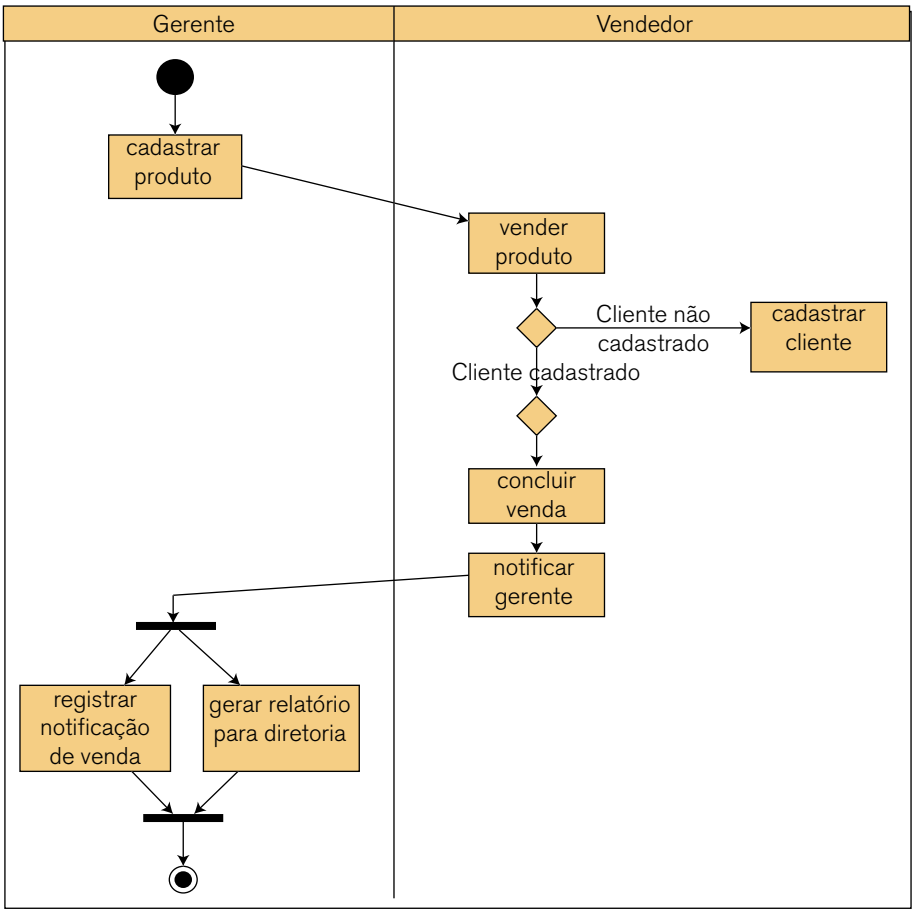


Figura 4.7 – Exemplo de diagrama de atividade.

Observando a parte de cima da Figura 4.7, temos duas divisões chamadas de raias. Estas raias definem os responsáveis pela execução das ações descritas mais abaixo. Neste caso, no retângulo da esquerda estão as ações que são executadas pelo gerente e à direita as ações executadas pelo vendedor.

As ações são os blocos em maior número no diagrama apresentado (Figura 4.7), os quais são executados em sequência partindo do início (esfera preta) ao fim (esfera preta com círculo externo). A sequência é dada pelas setas indicando qual seria a próxima ação a ser executada.

As ramificações sequenciais, também denominadas nós de decisão, são representadas por um losango e são utilizadas para definir qual ação será tomada dependendo de uma condição. No caso do exemplo da Figura 4.7, a condição é o cliente estar ou não cadastrado no sistema. O losango pode também representar intercalação, com o objetivo de unir saídas de ações, assim como é feito antes da conclusão da venda.

As bifurcações concorrentes e uniões concorrentes são representadas por barras onde, a próxima ação só é executada após todas as ações da bifurcação serem executadas. Neste caso (Figura 4.7), a atividade só é finalizada quando ambas ações descritas na bifurcação forem completamente executadas (o registro da notificação da venda e a emissão do relatório para a diretoria).

ATIVIDADES

Responda às seguintes questões:

01. Sobre casos de uso, geralmente qual detém maior quantidade de informação, o diagrama ou a descrição?
02. Cite uma vantagem de se utilizar diagrama de sequência ao invés do diagrama de comunicação.
03. Como eram conhecidos os diagramas de comunicação?
04. Qual a principal diferença entre um diagrama de atividade e um fluxograma?

REFLEXÃO

Este capítulo abordou uma grande quantidade de informação sobre modelagem de *software*. Primeiramente foi apresentada a descrição de casos de uso e pôde-se observar que o nível de detalhamento aumenta consideravelmente quando se opta por fazer a descrição estendida. Refletindo sobre algo que foi dito no início do livro, não é comum que se utilize de todos os diagramas ao modelar *software*, porém uma visão abrangente e conhecimento sobre os modelos pode trazer qualidade à modelagem, pois a chance de serem selecionados diagramas que representem melhor o que se quer modelar é maior.

LEITURA

O quinto capítulo do livro UML 2 Toolkit aborda modelagem dinâmica, que trata também dos diagramas apresentados neste capítulo. Alguns exemplos e aprofundamento podem ser encontrados nesta leitura altamente recomendada.

ERIKSSON, Hans-erik et al. **UML 2 Toolkit.** Indianapolis: Wiley, 2004.

REFERÊNCIAS BIBLIOGRÁFICAS

BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. **UML:** Guia do Usuário. 2. ed. Rio de Janeiro: Campus, 2005.

ERIKSSON, Hans-erik et al. **UML 2 Toolkit.** Indianapolis: Wiley, 2004.

LARMAN, Craig. **Utilizando UML e Padrões:** Uma introdução à análise e ao projeto orientados a objetos e ao desenvolvimento iterativo. 3. ed. Porto Alegre: Bookman, 2007.

SILVA, Aristófanes Corrêa. **Unified Modeling Language (UML)**. Disponível em: <http://www.deinf.ufma.br/~acmo/MOO_Com.pdf>. Acesso em: 16 out. 2015.

5

Diagrama de Classe de Projeto e Diagrama de Implementação

Chegamos ao último capítulo do livro de modelagem de sistemas. Aqui, é abordado o diagrama de classes de projeto, que é uma derivação do diagrama de classes já apresentado no capítulo 3 (inclusive sua apresentação teve alguns aspectos de projeto embutidos). Este diagrama, diferentemente do diagrama de classes conceitual já visto, enfoca na implementação do sistema e não em sua lógica de negócios. Porém, o entendimento do diagrama de classes conceitual é de suma importância para que se possa transferir seus conceitos para o diagrama de classes de projeto.

O aspecto físico da modelagem de sistemas também é abordado neste capítulo, onde são enfatizados itens físicos da implementação e implantação do *software* a ser desenvolvido.

Espero que tenha absorvido os conceitos apresentados até aqui e que este último capítulo lhe forneça uma visão de programação do *software* aliada à modelagem.

OBJETIVOS

Ao final deste capítulo, o aluno conhecerá:

- Diagrama de classes de projeto;
- Diagrama de componentes; e
- Diagrama de implantação.

5.1 Diagrama de classe de projeto

No capítulo 3 são apresentados os diagramas de classe, porém de uma forma conceitual (modelo de domínio). Os diagramas de classe de projeto são derivados do modelo de domínio, sendo definidos como um diagrama de classes na visão de implementação.

As classes de projeto abordam:

- As classes;
- As associações;
- Os atributos;
- As interfaces;
- Os métodos;
- O tipo dos atributos;
- A visibilidade dos atributos;
- A navegabilidade entre objetos; e
- A dependência.

Estes itens não são necessários no diagrama de classes (modelo de domínio), apesar de terem sido apresentados juntamente com estes no capítulo 3.

CONCEITO

O diagrama de classes (modelo de domínio) apresenta os objetos do sistema dentro de uma visão lógica, representando as funcionalidades do sistema independente de tecnologia. As classes de projeto representam um modelo dependente de tecnologia, com a adição de itens que são pertinentes à implementação do *software*.

Pelo fato dos diagramas de classe de projeto serem dependentes de tecnologia, é necessário adotar uma para que se possa desenvolver os modelos. Neste capítulo, subentende-se que o projeto é desenvolvido utilizando a linguagem de programação Java.

Vale a pena relembrar os tipos de atributos que vamos utilizar nos diagramas apresentados:

- **String:** atributo do tipo texto;
- **Integer:** atributo do tipo número inteiro;

- **Float:** atributo do tipo número real;
- **Date:** utilizado para datas;

As atividades da fase de projeto, de acordo com Bezerra (2007), são as seguintes:

- Definição detalhada de aspectos dinâmicos do sistema;
- Aprimoramento de aspectos estáticos e estruturais do sistema; e
- Detalhamento e definição de outros aspectos da solução.

Algumas dessas etapas são cumpridas nos diagramas de interação, outras no diagrama de classe de projeto.

? CURIOSIDADE

O diagrama de classes de projeto possui informação suficiente para a criação do banco de dados do sistema, portanto, pode ser utilizado para este fim.

O diagrama de classes de projeto aborda um tipo específico de classe: as classes de fronteira. Estas classes são representações da interação do sistema (formulários, atores, outros sistemas, entre outros) e não devem conter lógica de negócio embutida. É comum que se tenha uma classe de fronteira para cada ator participante do caso de uso, porém podem ter casos em que serão necessárias mais de uma classe ou até modelos de subsistemas (BEZERRA, 2007).

As classes de entidade são basicamente as classes do modelo conceitual apresentado no capítulo 3. Estas classes modelam a lógica de negócio do sistema e geralmente são transportadas para o modelo de classes de projeto sem grandes alterações.

Um exemplo de diagrama de classe de projeto é apresentado na figura 5.1, onde se modela um caso de uso de registro de empréstimo de livro em uma biblioteca.

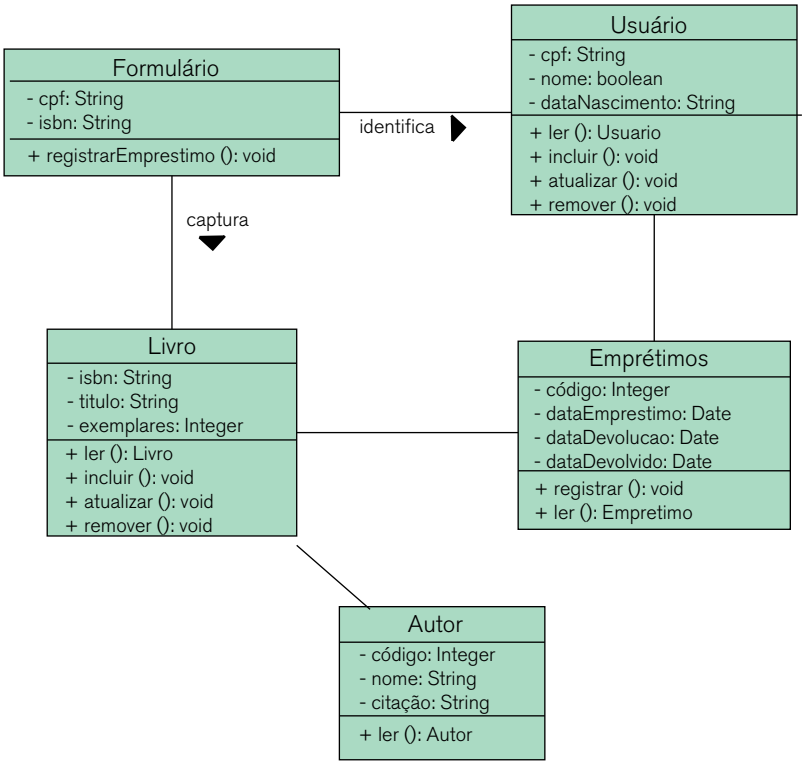


Figura 5.1 – Diagrama de classes de projeto de um sistema de registro de empréstimos de livros.

No exemplo da figura 5.1, a classe "Empréstimo" pode ser considerada uma entidade associativa, pois está entre o relacionamento de usuário com livro (usuário empresta livro), sendo que um usuário pode emprestar vários livros e um livro pode ser emprestado a vários usuários.

As setas entre as classes indicam a visibilidade entre os objetos. A seta que parte de "Autor" para "Livro" indica que autor enxerga livro, em outras palavras, a informação identificadora de "Livro" é adicionada ao objeto "Autor". Podemos definir também que na implementação, a classe "Autor" possui um objeto da classe "Livro" como atributo. No modelo conceitual, neste caso específico, a modelagem mostra que um autor pode escrever apenas um livro, mas um livro pode possuir vários autores.

No caso da classe associativa “Empréstimo”, pode-se afirmar que esta possui um objeto da classe “Livro” e um objeto da classe “Usuário” como atributos. Segundo a visibilidade de objetos, pode-se afirmar que “Empréstimo” enxerga “Usuário” e “Livro”.

? CURIOSIDADE

Do ponto de vista de implementação, pode-se utilizar listas de objetos para facilitar o desenvolvimento, como no exemplo da Figura 5.1, pode-se ter uma lista de objetos da classe “Autor” como atributo da classe “Livro”.

Para que um objeto possa enviar mensagens a outro, este destinatário deve ser visível ao remetente. A visibilidade pode ser conseguida de quatro maneiras:

1. **Por atributo:** quando uma classe detém um objeto de outra como seu atributo, assim como apresentado no exemplo da figura 5.1.
2. **Por parâmetro:** quando um objeto de uma classe é parâmetro de algum método de outra. Se uma classe A possui um método que recebe um objeto da classe B como parâmetro, então B é visível para A.
3. **Declaração local:** quando um objeto de uma classe é declarado como variável local em um método de outra. Se a classe A possui um método e, neste método, é declarado um objeto da classe B, então B é visível para A. Este método é muito utilizado com listas, inclusive resultados de pesquisas em bancos de dados.
4. **Visibilidade global:** quando uma variável, objeto de uma classe, é visível globalmente no sistema. Deste modo, é visível para todos os outros objetos.

A navegabilidade indica a associação unidirecional, ou seja, a navegação de um objeto de origem para o objeto-alvo. A navegação só é permitida se o objeto -alvo for visível, portanto, a rota de navegação é dada pela seta de visibilidade.

No exemplo da figura 5.1, podemos interpretar a navegabilidade entre “Empréstimo” e “Livro” como: um “Empréstimo” possui um “Livro”.

O conceito de dependência, explicado no item 3.2.5 deste livro, é utilizado em diagramas de classes de projeto. A dependência indica que um objeto tem conhecimento de outro por um curto prazo e é associado à visibilidade por parâmetro, por definição local ou por definição global.

A cardinalidade dos relacionamentos no diagrama de classes conceitual é utilizada para checar a navegabilidade entre as classes:

• Em relacionamentos onde a cardinalidade é de 1 para muitos (de um lado a cardinalidade máxima é 1 e do outro é *), o lado onde aparece o asterisco possui um objeto da classe em que está se relacionando (vide figura 5.2);

• Em relacionamentos de onde a cardinalidade é de 1 para 1, a navegabilidade acontece a gosto do analista, pois geralmente é irrelevante qual classe possuirá um objeto de seu relacionamento como atributo.

• Em relacionamentos onde a cardinalidade é de muitos para muitos, utiliza-se uma classe associativa que possui um objeto de cada classe relacionada (vide figura 5.3).

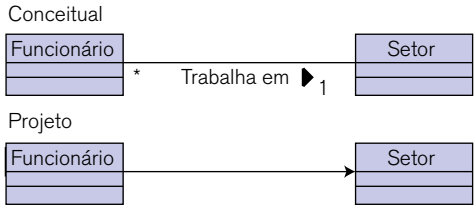


Figura 5.2 – Exemplo de relacionamento de um para muitos.

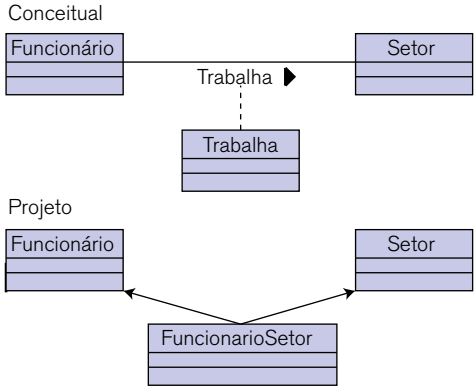


Figura 5.3 – Exemplo de classe associativa em diagrama de classes de projeto.

No conceito de herança, há três formas ou abordagens das classes envolvidas serem transportadas para o diagrama de classes de projeto:

• Criando todas as classes envolvidas;
• Criando apenas a classe pai;
• Criando apenas as classes filhas;

A figura 5.4 exibe um exemplo de herança no diagrama de classes conceitual para que em seguida sua utilização no diagrama de classes de projeto seja explicada para cada uma das três abordagens.

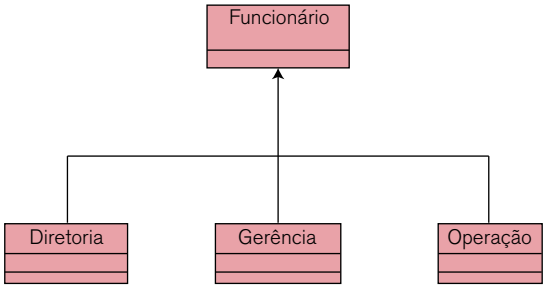


Figura 5.4 – Exemplo de herança em um diagrama de classes conceitual.

A primeira abordagem é melhor utilizada quando todas as classes possuem relacionamentos específicos com outras classes e seus atributos são bem distribuídos, isto é, as classes possuem um número parecido de atributos. Neste caso, o diagrama de classes de projeto possuiria as quatro classes: "Funcionário", "Diretoria", "Gerência" e "Operação". Cada classe possuirá seus atributos e seus métodos, assim como seus relacionamentos específicos.

A segunda abordagem é a criação apenas da classe pai (neste caso, a classe "Funcionário"). Ao utilizar esta abordagem, apenas uma classe aparece no diagrama de classes de projeto, isto é, apenas uma tabela é criada no banco de dados. Esta classe deve conter todos os seus próprios atributos, métodos e relacionamentos, como também todos os atributos, métodos e relacionamentos de cada um dos seus filhos. Este caso é bem empregado quando se tem uma grande quantidade de atributos e relacionamentos na classe pai e poucos atributos e relacionamentos nas classes filhas.

A terceira abordagem é a criação apenas das classes filhas. Nesse caso (figura 5.4), as classes que apareceriam no diagrama de classes de projeto são: "Diretoria", "Gerência" e "Operação". No banco de dados deste sistema, estariam implementadas apenas três tabelas, uma para cada classe. Cada classe deve conter seus atributos, métodos e relacionamentos e também os atributos, métodos e relacionamentos da classe pai.

? CURIOSIDADE

É comum que se utilize a abordagem de criação de todas as classes (primeira abordagem) para a implementação no sistema, porém no banco de dados isto varia com mais frequência.

Outro conceito utilizado em diagramas de classes de projeto são as interfaces. Estas não podem ser confundidas com as interfaces gráficas do sistema, pois são ações que um objeto pode realizar dentro de um sistema específico. Na figura 5.5 tem-se um exemplo de interface em um diagrama de classes de projeto.

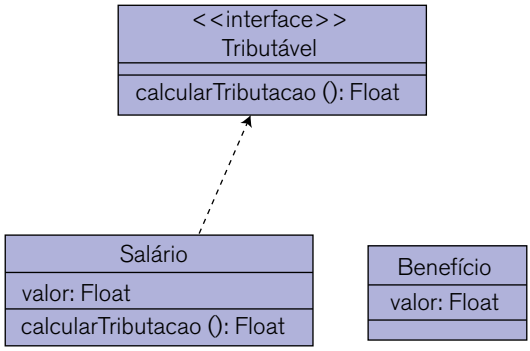


Figura 5.5 – Exemplo de interface em diagrama de classes de projeto.

Quando uma classe realiza a ação de uma interface, diz-se que esta classe implementa tal interface. Portanto, no exemplo da figura 5.5, pode-se dizer que a classe "Salário" implementa a interface "Tributável", enquanto a classe "Benefício" não.

Uma interface contém um conjunto de métodos (ações), porém, estes métodos não são implementados pela interface. O que acontece é que a interface possui apenas a assinatura destes métodos, em outras palavras, ela descreve o que fazer, mas não como fazê-lo.

A implementação dos métodos da interface é de responsabilidade das classes que a implementam e esta implementação deve ser feita utilizando a assinatura idêntica a que é apresentada na interface, inclusive tipos de parâmetros e de retorno. Neste caso, no exemplo da Figura 5.5, a implementação de como

é calculada a tributação (método “calcular Tributação”) é de responsabilidade da classe “Salário”.

A classe benefício não tem a obrigação de implementar o (método “calcular Tributação”) pois esta não implementa a interface “Tributável”.

Para a indicação de uma implementação entre uma classe e uma interface utiliza-se de uma seta tracejada com a ponta vazada, esta seta parte da classe para a interface (assim como demonstrado na Figura 5.5).

Pode-se dizer, utilizando o conceito de interface, que o salário é tributável e o benefício não é tributável.

5.2 Diagramas de implementação

Estes diagramas modelam aspectos físicos e descrevem *hardware* e *software* que cercam a implementação de um sistema. São abordados dois tipos de diagramas de implementação, pertencentes à UML:

- Diagrama de componentes; e
- Diagramas de implantação.

A seguir, estes dois diagramas são explicados e discutidos.

5.2.1 Diagrama de componentes

O diagrama de componentes é utilizado para modelar itens físicos que podem residir em um nó, como exemplo tem-se:

- Arquivos executáveis;
- Bibliotecas;
- Documentos;
- Tabelas; arquivos em geral.

O componente é a menor representação de um programa, portanto, é necessário que se estabeleça a modelagem física do sistema executável, que pode ser um conjunto de programas.

De acordo com Booch, Rumbaugh e Jacobson (2005), estes diagramas mostram as dependências entre os diversos componentes de um *software,* inclui-se desta forma:

- Componentes de código-fonte;
- Componentes de código binário; e
- Componentes executáveis;

Pode-se definir este diagrama como um grafo de componentes conectados por relacionamentos de dependência.

CONEXÃO

Grafos são definidos de forma teórica no *link* a seguir, disponibilizado pela Universidade Federal de Santa Catarina:
http://www.inf.ufsc.br/grafos/definicoes/definicao.html

A notação deste diagrama possui três símbolos:
- Componente (vide figura 5.6);
- Interface; e
- Dependência.

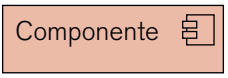


Figura 5.6 – Exemplo de componente.

Um exemplo didático deste diagrama é apresentado na Figura 5.7.

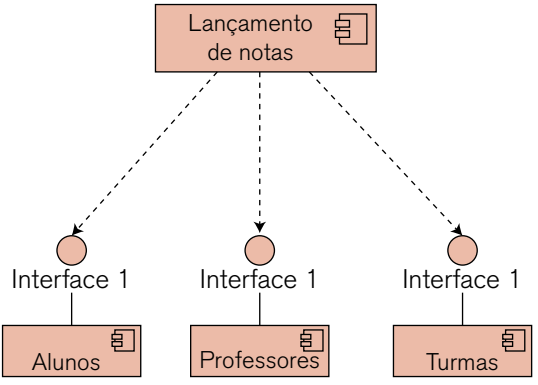


Figura 5.7 – Exemplo didático de diagrama de componentes.

No exemplo apresentado na figura 5.7, o componente “Lançamento de notas” é dependente das interfaces fornecidas pelos componentes “Alunos”, “Professores” e “Turmas”. Isto significa que para as notas serem lançadas no sistema, os alunos, professores e as turmas devem estar disponíveis.

Outro exemplo, agora mais prático e adaptado de Fowler (2005), é apresentado na figura 5.8. Neste exemplo pode-se observar um diagrama de componentes onde um caixa efetua uma venda em uma loja.

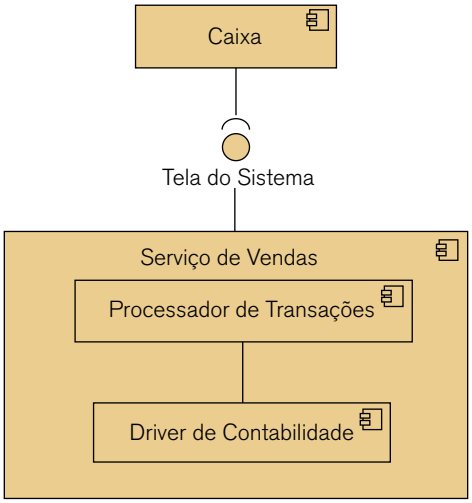


Figura 5.8 – Exemplo de diagrama de componentes (adaptado de FOWLER, 2005).

Ao observar a figura 5.8, um componente “Caixa”, para efetuar uma venda, é requerida uma interface fornecida pelo componente “Servidor de Vendas”. Os componentes citados “Processador de Transações” e “Driver de Contabilidade” são parte do componente maior, o “Servidor de Vendas”. O processador de transações, por sua vez, é dependente do driver de contabilidade, que fará os cálculos necessários para a transação da venda.

5.2.2 Diagramas de implantação

De acordo com os criadores da linguagem UML, os diagramas de implantação são empregados para a modelagem de uma visão estática do sistema, envolvendo inclusive a topologia de *hardware* na qual o sistema é executado. Este diagrama tem como objetivo visualizar, especificar e documentar sistemas embutidos, cliente/servidor e distribuídos (BOOCH; RUMBAUGH; JACOBSON, 2005).

CONCEITO

"Um sistema embutido é uma coleção complexa de *software* para o *hardware* que interage com o mundo físico. Os sistemas embutidos envolvem *software* que controla dispositivos como motores, atuadores e monitores e que, por sua vez, é controlado por estímulos externos como entrada de um sensor, movimentação e mudança de temperatura." (BOOCH; RUMBAUGH; JACOBSON, 2005)

CONCEITO

"O sistema cliente/servidor é uma arquitetura comum, cujo foco é a criação de uma clara separação de questões entre a interface para o usuário (que vive no cliente) e os dados persistentes no sistema (que vive no servidor)." (BOOCH; RUMBAUGH; JACOBSON, 2005)

CONCEITO

Sistema distribuído

"Um sistema distribuído é um conjunto de computadores independentes que se apresenta a seus usuários como um sistema único e coerente" (TENENBAUM; VAN STEEN, 2007)

Os diagramas de implantação mostram a configuração de nós de processamento em tempo de execução e seus componentes. Os componentes que não existem em tempo de execução não são abordados por estes diagramas.

Basicamente, o diagrama de implantação é um grafo de nós conectados por associações de comunicações. A representação de um nó neste diagrama é apresentada na figura 5.9.

Figura 5.9 – Exemplo de representação de nó em um diagrama de implantação.

Os nós geralmente são:

- Computadores;
- Impressoras;
- Leitores de cartão;
- Dispositivos de comunicação;
- Entre outros.

Na figura 5.10 é apresentado um exemplo de diagrama de implantação simplificado e didático. Neste exemplo, um cliente acessa um servidor de aplicação na internet por meio do protocolo HTTP (Hyper-Text Transfer Protocol), enquanto este servidor faz o acesso ao banco de dados para coletar e gravar informações.

O banco de dados do sistema modelado possui um sistema gerenciador (SGBD) e a unidade de persistência (os dados propriamente ditos). A persistência destes dados é dependente do sistema gerenciador, pois é ele quem controla as transações.

O sistema também possui uma impressora, que é acessada pelo cliente via rede local (LAN)

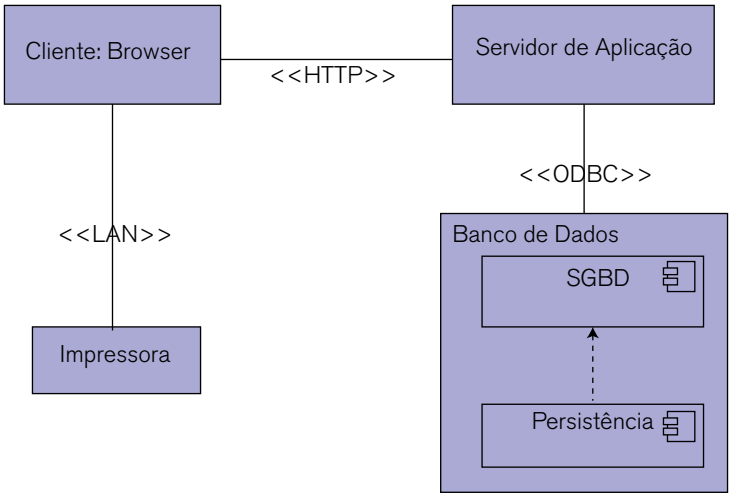


Figura 5.10 – Exemplo de diagrama de implantação.

ATIVIDADES

01. Qual o foco do diagrama de classes de projeto para modelagem de sistemas?

02. Pensando em um sistema fictício e puramente didático, faça um diagrama de classes de projeto com apenas duas classes: "Elefante" e "Urubu", onde uma delas deve implementar uma interface chamada "Voador", que define o método voar.

03. Dê três exemplos de itens que podem ser representados por nós em diagramas de implementação.

REFLEXÃO

Os modelos de classes de projeto são uma alternativa muito rica para modelagem de *software* visando a implementação do código-fonte, assim como os modelos de implementação trazem aspectos práticos da especificação física do funcionamento e implantação do *software* como produto.

É importante lembrar que os modelos podem ter diferentes níveis de abstração e detalhamento, portanto, devem ser definidos pelo analista conforme a necessidade.

LEITURA

No sétimo capítulo do livro "Engenharia de Software: uma abordagem profissional" de Roger S. Pressman (PRESSMAN, 2011) é abordada modelagem de software específica para aplicações *web.*

PRESSMAN, Roger S.. **Engenharia de Software:** Uma Abordagem Profissional. 7. ed. Porto Alegre: Amgh, 2011.

REFERÊNCIAS BIBLIOGRÁFICAS

BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. **UML:** Guia do Usuário. 2. ed. Rio de Janeiro: Campus, 2005.

FOWLER, Martin. **UML Essencial:** Um breve guia para a linguagem-padrão de modelagem de objetos. 3. ed. Porto Alegre: Bookman, 2005.

PRESSMAN, Roger S.. **Engenharia de Software:** Uma Abordagem Profissional. 7. ed. Porto Alegre: Amgh, 2011.

TENENBAUM, Andrew S.; VAN STEEN, Maarten. **Sistemas Distribuídos:** princípios e paradigmas. 2. ed. São Paulo: Pearson Prentice Hall, 2007.

GABARITO

Capítulo 1

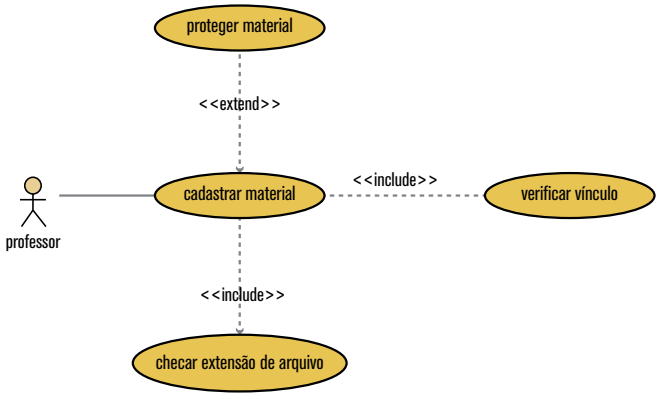
01. A primeira desvantagem que pode ser citada é que se ocorrer algum erro de especificação do *software*, é necessário que se retorna ao ponto de especificação para que este seja refeito, o que reflete em retrabalho também nas próximas etapas. Uma segunda desvantagem pode ser a entrega do produto pronto, pois o cliente pode identificar erros na execução, e então o projeto todo deve ser revisado.

02. Neste tipo de sistema pode-se identificar uma série de classes. Para citar apenas duas, podem ser escolhidas a classe "Cliente" e a classe "Carro".

03. Na adoção de metodologia ágil de desenvolvimento de *software*, é comum utilizar o processo incremental.

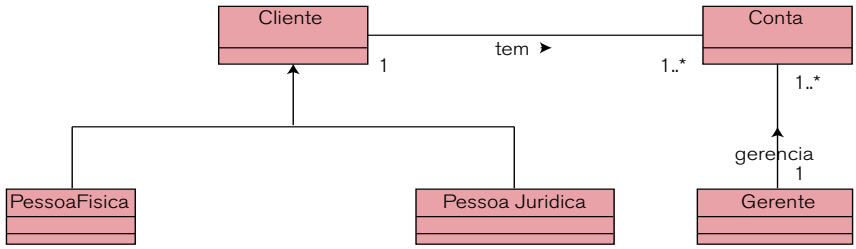
Capítulo 2

Existem diversas maneiras de se especificar *software* em diagramas de casos de uso, com diferentes níveis de detalhamento. Uma delas é apresentada a seguir:



Capítulo 3

O diagrama de classes representa a estrutura lógica do sistema, e desta forma, um sistema bem simplificado de uma instituição bancária abrangendo apenas cliente, conta e gerente pode ser como a seguir:

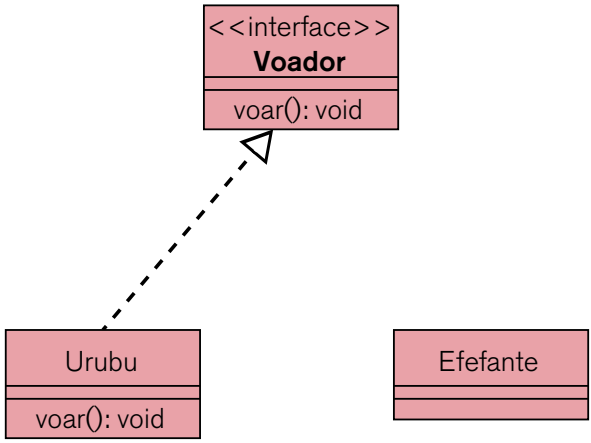


Neste modelo, um cliente pode ser de dois tipos: pessoa física ou pessoa jurídica e ambos podem ter contas bancárias. Uma conta bancária pode ser de apenas um cliente. Um gerente pode gerenciar diversas contas, porém, uma conta só pode ter um gerente.
Esta modelagem pode variar conforme interpretação e lógica de negócios, portanto, variações podem estar corretas.

Capítulo 4

01. A descrição de casos de uso detém mais informação, por descrever a realização do caso de uso passo a passo, provendo inclusive informações sobre os fluxos alternativos que podem ocorrer em uma realização.
02. O diagrama de sequência pode ser utilizado em casos que o processo precisa ser analisado de forma temporal, ou seja, identificar a sequência das mensagens trocadas entre objetos.
03. Antigamente, os diagramas de comunicação eram conhecidos como diagramas de colaboração (até a versão 1.5 da UML). A partir daí seu nome foi alterado para diagrama de comunicação.
04. A principal diferença entre um diagrama de atividade e um fluxograma é que o diagrama de atividade é capaz de implementar processos paralelos.

01. O foco do diagrama de classes de projeto é na implementação do sistema, definindo tipos de variáveis, métodos, entre outros termos. Isto o torna dependente de tecnologia.
02. Um modelo básico que satisfaz o que é pedido na questão é exibido abaixo:



03. Três dos itens que podem ser representados por nós nestes diagramas podem ser:
    a) computadores (como processadores);
    b) impressoras; e
    c) dispositivos de comunicação.