Análise e Modelagem de Sistemas com a UML

Com Dicas e Exercícios Resolvidos

Luiz Antônio de Moraes Pereira

1ª Edição

Rio de Janeiro

Edição do Autor

2011

lpereira@uninet.com.br

CIP-BRASIL. CATALOGAÇÃO-NA-FONTE
SINDICATO NACIONAL DOS EDITORES DE LIVROS, RJ

P492a
Pereira, Luiz Antônio de Moraes, 1957-
Análise e modelagem de sistemas com a UML : com dicas e exercícios resolvidos / Luiz Antônio de Moraes Pereira. – 1.ed. – Rio de Janeiro : Luiz Antônio M. Pereira, 2011.
il.
Inclui bibliografia
Apêndice
ISBN 978-85-911695-0-4
1. Software - Desenvolvimento. 2. UML (Computação). 3. Análise de sistemas. I. Título.
11-0168. CDD: 005.1
CDU: 004.41
10.01.11 12.01.11 023817

Aos meus filhos Felipe e Henrique.

Luiz Antônio graduou-se em Engenharia de Fortificação e Construção pelo Instituto Militar de Engenharia – IME – no Rio de Janeiro em 1980.

Obteve o grau de Mestre em Informática, com enfoque em Computação Gráfica, pela Pontifícia Universidade Católica do Rio de Janeiro – PUC-Rio – em 1987.

Trabalhou de 1981 a 1991 no segmento de Tecnologia da Informação em diversas empresas no Rio de Janeiro e em São Paulo.

Trabalhou de 1991 a 1993 com Computação Gráfica na TV Cultura, São Paulo.

Desde 1993 trabalha no Banco Central do Brasil. Atualmente está lotado na Coordenadoria de Informática da Gerência Regional do Rio de Janeiro.

Ao longo de sua carreira atuou como professor/instrutor em diversas organizações e desde 2001 atua como Professor da PUC-Rio, ministrando cursos em análise, projeto e desenvolvimento de sistemas, em banco de dados e em gerência de projetos de software na Coordenação Central de Extensão – CCE.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

CA	Curso Alternativo
CASE	*Computer-Aided Software Engineering* – Engenharia de Software Ajudada por Computador
CMMI	*Capability Maturity Model Integration* – Integração do Modelo de Nível de Maturidade
CT	Curso Típico
DA	Diagrama de Atividades
DER	Diagrama de Entidades e Relacionamentos
DFD	Diagrama de Fluxos de Dados
DME	Diagrama de Máquina de Estados
DS	Diagrama de Sequência
ECA	Evento, Condição e Ação
MER	Modelo de Entidades e Relacionamentos ou Modelo Entidades-Relacionamentos
OCL	*Object Constraint Language* – Linguagem (de Especificação) de Restrições de Objetos
OMG	*Object Management Group* – Grupo de Gerência de Objetos
OO	Orientação a Objetos
OOAD	*Object-Oriented Analysis and Design* – Análise e Projeto Orientados a Objetos
SGBD	Sistema de Gerenciamento de Banco de Dados

UML	*Unified Modeling Language* – Linguagem Unificada de Modelagem
XMI[DI]	*XML Metadata Interchange, Diagram Interchange* – Intercâmbio de Diagramas com XML
XMI	*Metadata Interchange* – Intercâmbio de Metadados com XML
XML	*Extensible Markup Language* – Linguagem Extensível de Marcação

PREFÁCIO

A Linguagem Unificada de Modelagem (em inglês *Unified Modeling Language* – UML) foi concebida com o intuito de estabelecer um padrão único a ser usado para a especificação das características dos sistemas computacionais projetados para atender às necessidades dos usuários desses sistemas.

A linguagem começou a ser definida em novembro de 1997 e rapidamente se tornou um padrão *de facto*, causando grande impacto na maneira como os sistemas de software vêm sendo desenvolvidos, sendo usada para a comunicação entre membros da equipe de desenvolvimento, na discussão e especificação de soluções, e entre eles e os usuários, como recurso de especificação do que é para ser feito. A UML também vem sendo usada como especificação com vistas à geração automática de código com base no modelo, de forma modular e com altos índices de reúso.

A importância que a UML assumiu no contexto de desenvolvimento de sistemas, fez com que o curso de Análise, Projeto e Gerência de Sistemas – APGS – e o seu sucessor, o de Análise e Projeto de Sistemas – APS –, da PUC-Rio, dedicasse uma parte significativa do programa ao estudo teórico e prático desse assunto, mesclado, como não podia deixar de ser, com as questões relativas às disciplinas de análise e projeto de sistemas de computação.

O que apresento neste texto é uma reunião das questões que venho discutindo em sala com os alunos da disciplina de Métodos de Análise de Sistemas – MAS – desses dois cursos. O assunto "análise de sistemas" é tratado na medida em que a necessidade vem surgindo, enquanto explico modelagem. Eu busquei relacionar e organizar os assuntos da maneira que fui percebendo ser a mais conveniente para a apresentação aos alunos (na maioria iniciantes em UML), não só pela ordem tipicamente empregada dos diagramas em um projeto e a relevância no contexto de desenvolvimento de sistemas, como também pelos interesses despertados por eles em sala de aula. Assim sendo, se você quer iniciar os estudos em UML, sugiro que leia o texto na ordem em que os assuntos são apresentados.

Talvez você considere o texto um tanto extenso, mas ele contém um resumo da UML, cuja especificação é longa porque precisa ser detalhada e porque a UML

é extensa e objetiva ser abrangente e completa. Com isso, parece óbvio que, em pouco mais de duzentas páginas, é bastante difícil tratar completamente o que a especificação da UML trata em mais de mil páginas; e isso sem passar as dicas, técnicas e melhores práticas de análise e modelagem de sistemas que procuro passar ao longo dos estudos. Eu vejo o texto como uma "partida rápida" para os que querem se familiarizar com a linguagem e aplicar os conhecimentos mais imediatamente em seus projetos. O que é tratado no texto também pode ser entendido como uma visão de alto nível, um panorama, para os que querem se aprofundar no estudo da UML.

Espero que esse texto seja útil de alguma forma em sua atividade de análise e modelagem de sistemas.

Luiz Antônio

CAPÍTULO 1

INTRODUÇÃO

Whatever begins, also ends.

Seneca

Quando nos propomos a realizar uma viagem a passeio, sabemos que é importante estabelecer, de antemão, roteiros, metas atingíveis, determinar custos, juntar os recursos necessários, avaliar alternativas e os riscos de atrasos no trajeto. É necessário definir um plano de atividades e ouvir a experiência de várias pessoas, dentre outras medidas. Desejamos, com isso, aumentar as chances da viagem ser um sucesso, certo? Boa parte desse esforço pode ser deixada a cargo de um funcionário de uma agência de turismo, com quem batemos um papo relativamente rápido, por telefone mesmo, expondo nossas expectativas.

Quando queremos construir nossa casa, temos que ser bem mais rigorosos quanto à especificação do que queremos. Nesse caso, um papo por telefone com um arquiteto já não basta para escalonar as etapas, levantar custos, definir prazos etc. As eventuais perdas, no caso de um insucesso na construção da casa, tendem a ser bem maiores do que no caso da viagem. Por essa razão, para que os detalhes do projeto fiquem bem entendidos por todos, os profissionais que prestarão os diversos serviços durante o projeto e a obra trocarão informações usando o jargão da área e por meio de modelos e especificações mais estruturadas e precisas do que um papo por telefone.

Para o projeto e construção de um novo automóvel ou navio, de um prédio, de uma ponte ou de uma rodovia (para a execução de qualquer obra de engenharia

de grande porte), é fundamental reunir um conjunto bastante extenso de desenhos, especificações, recomendações, além de estabelecer *a priori* processos de projeto e de construção. Deveremos contar com etapas bem definidas e pontos de controle intermediários para a verificação do andamento e para a medição da qualidade do que está sendo produzido e do processo de produção propriamente dito.

Curiosidade: Para a construção e montagem da usina hidrelétrica de Itaipu foram empregadas, desde a elaboração e as revisões de seu projeto, cerca de 1.200.000 folhas de desenho e listas de materiais. Essa documentação, superposta, seria suficiente para erguer uma pilha com altura equivalente a um prédio de 50 andares.
Fonte: Itaipu Binacional.

Em sociedades cada vez mais dependentes da informática, a complexidade e a criticidade dos sistemas têm aumentado, não havendo mais espaço para improvisações e para trabalho amador. O cuidado com o projeto e a documentação e o controle dos custos, prazos e qualidade vêm ganhando importância na construção de sistemas computacionais, que vem se tornando cada vez mais uma atividade de engenharia, uma obra de engenharia.

Muitos programas de computador infelizmente ainda têm sido construídos sem planejamento, sem especificação de como serão feitos e mesmo sem especificação do que será feito.

Neste texto apresentaremos os principais diagramas da Linguagem Unificada de Modelagem – *Unified Modeling Language* - UML – para o planejamento e a especificação dos aspectos conceituais de sistemas computacionais, de forma a podermos estruturar o resultado de nosso trabalho e encaixá-lo em uma atividade maior, que envolve todas as etapas da engenharia de construção desses sistemas.

1.1 Bibliografia Adicional Recomendada

Uma pergunta que quase invariavelmente me fazem no início das aulas que ministro refere-se ao que ler para complementar o que falo em sala (que pode ser quase totalmente associado ao que tratarei nesta nossa apostila). Costumo recomendar três livros, dependendo do tipo de informação em que o aluno está interessado.

Se você está interessado em uma visão aplicada da UML, mais informal e "coloquial" da linguagem, eu recomendo o livro *UML Essencial* ([5]), de Martin Fowler. Nesse livro relativamente fino, Fowler coloca em evidência, de forma bem pedagógica, sua experiência com o uso da UML em ambientes reais. O livro cobre basicamente os mesmos diagramas de que trataremos neste texto.

Se você está interessado em um material mais completo e mais alinhado com a especificação formal da UML, eu recomendo o livro *UML – Guia do Usuário* ([1]), de Grady Booch, James Rumbaugh e Ivar Jacobson, autores que são vistos por muitos como as três personalidades mais importantes da UML.

Se você está interessado em uma visão da UML em ambientes de desenvolvimento com o uso do *Rational Unified Process* (Processo Unificado da empresa Rational) – RUP – eu recomendo o livro *Utilizando UML e Padrões* ([8]), de Craig Larman, outro grande conhecedor e divulgador de UML e orientação a objetos.

Esses três livros são bem interessantes e, de alguma forma, se complementam. No entanto, caso você queira um único material impresso além deste, eu recomendaria o livro do Fowler.

Não poderia deixar de comentar que a especificação da UML pode ser baixada da Internet gratuitamente. Essa especificação consiste de mais de mil páginas em arquivos `.pdf`, disponíveis no sítio do *Object Management Group* (Grupo de Gerência de Objeto) – OMG[1] –, grupo que gere o desenvolvimento da linguagem. Não espere, no entanto, entender muito facilmente o que está descrito lá, pois as informações são colocadas sem levar em conta o aspecto pedagógico. A consulta à especificação normalmente só é feita no caso de uma dúvida quanto a um detalhe ou outro da linguagem. Os dois principais documentos da especificação são o documento de infraestrutura ([10]) e o de superestrutura ([11]).

1.2 As Ferramentas CASE

Outras duas perguntas que os alunos me fazem dizem respeito à necessidade de uso de uma ferramenta de desenho dos modelos e, havendo, qual eles devem usar.

Os modelos podem, claro, ser desenvolvidos utilizando papel e lápis. As ferramentas de software, no entanto, permeiam a quase totalidade dos ambientes de trabalho nas mais diversas áreas da atividade humana, dando suporte organizado ao planejamento, à metodologia e à disciplina, que são componentes essenciais de qualquer ambiente produtivo. Modelagem de sistemas se insere nesse contexto.

Para a modelagem de sistemas utilizando a UML, usualmente empregamos ferramentas CASE (*Computed-Aided Software Engineering* – engenharia de software auxiliada por computador), apelidadas simplesmente de CASEs.

Nas boas ferramentas, todos os elementos de notação que normalmente precisamos estão disponíveis, inclusive nas versões gratuitas, as chamadas versões "comunidade", ou *community*, em inglês.

[1] Ver `http://www.omg.org`.

Simples editores gráficos, que permitem manipular "caixinhas e bolinhas" numa tela gráfica, já possuem vantagens em relação ao uso de papel e lápis, por conta de facilitar a experimentação necessária durante a modelagem. CASEs, no entanto, são mais do que isso. O que faz a diferença real no uso dos CASEs para modelagem de sistemas é o suporte organizado ao processo de modelagem que mencionamos, por conta da garantia que eles normalmente dão de que o modelo gerado está sintaticamente correto e de que os diversos diagramas que compõem o modelo estão consistentes entre si. Por exemplo, se dois determinados atores estão relacionados entre si por generalização/especialização em um diagrama, esse relacionamento é representado automaticamente pelo CASE em todos os demais diagrama do modelo onde eles aparecerem. Caso removamos esse relacionamento em um diagrama, todas as representações desse relacionamento nos demais diagramas serão automaticamente removidas. Além disso, CASEs não permitem que atribuamos um identificador já existente a um novo elemento do modelo em um mesmo espaço de nomes . CASEs também não devem permitir que estabeleçamos associações entre dois atores, o que é vedado pela UML.

Há vários CASEs de qualidade no mercado. Apenas a título de exemplo, podemos citar o *Astah* (que é o sucessor do Jude), o *Magic Draw* e o *Enterprise Architect*, alguns deles com versões *community* (gratuitas) e *professional*, e outros com versões de preços bem acessíveis, variando entre US100eUS 200 por cópia.

Eu acho muito importante que você use uma ferramenta CASE em suas atividades de modelagem. Sugiro que você baixe e instale o *Astah* versão *community*, que é gratuito, opera sobre o *Windows*, *Linux* e *MacOS*, pois é escrito em *Java*, e possui recursos suficientes para a maioria das necessidades de modelagem.

Eu farei comentários sobre os recursos e limitações mais importantes e comuns dos CASEs. Lembre-se, no entanto, de que nossa apostila não é sobre CASE, e sim sobre análise e modelagem com UML.

1.3 Organização do Texto

O restante deste texto é organizado da forma que segue.

No Capítulo 2 fazemos uma revisão dos conceitos que, de maneira geral, motivam e estão associados aos propósitos de nosso texto. Tratamos de processos, mecanismos de estruturação e controle de processos de software, orientação a objetos (propriedades, responsabilidades, operações e colaboração entre objetos), modelos, seus níveis de abstração e os principais recursos para as suas elaborações com vistas a prover a fundamentação para o restante do texto.

Nos Capítulos 3 e 4 tratamos de casos de uso. Inicialmente, no Capítulo 3,

apresentamos os conceitos e a notação para elaboração dos diagramas de casos de uso e, no Capítulo 4, tratamos das especificações escritas (descrições) dos casos de uso. No Capítulo 4 também chamamos a atenção para erros típicos de modelagem e damos algumas dicas para a elaboração dos diagramas e das descrições.

Os diagramas de classes são tratados nos Capítulos 5 e 6. Nesses dois capítulos discutimos como identificar os conceitos, seus atributos e os relacionamentos que eles mantêm entre si, além de como elaborar diagramas de classes que os especificam.

Os diagramas de máquina de estados e os diagramas de atividade são tratados nos Capítulos 7 e 8, respectivamente. Embora a UML, a partir da versão 2.0, tenha desvinculado um do outro por conta do aumento de expressividade dos diagramas de atividade, nós mantemos a correlação entre os dois para facilitar a passagem e a assimilação dos conceitos. As especificidades e os detalhes na notação de cada diagrama são tratados dentro de um nível de detalhamento e abrangência que julgamos suficiente para a modelagem conceitual de sistemas e, portanto, para os propósitos do texto.

Os conceitos e notação dos diagramas de sequência são abordados nos Capítulos 9 e 10, por conta da extensão do assunto.

Os diagramas de visão geral da interação, de pacotes, de componentes e de instalação são discutidos no Capítulo 11, onde também falamos de palavras-chave (os antigos estereótipos da UML), último tópico tratado neste texto.

As soluções comentadas dos exercícios propostos nos finais dos capítulos se encontram no Apêndice A. No Apêndice B apresentamos alguns minimundos completos para a elaboração dos modelos UML, conforme haja disposição e disponibilidade de tempo do leitor.

CAPÍTULO 2

FUNDAMENTOS DA MODELAGEM DE SISTEMAS

> I do not think much of a man who is not wiser today than he was yesterday.
>
> Abraham Lincoln

Sistemas computacionais têm desempenhado um papel cada vez mais importante em nossas vidas e, em muitas situações, o atendimento às necessidades dos usuários, questões de performance e o funcionamento correto ou incorreto desses sistemas pode fazer a diferença entre manter a vida ou causar a morte de alguém. Por outro lado, os custos e os prazos de seu desenvolvimento também constituem preocupações para os patrocinadores, os projetistas e os construtores desses sistemas.

Inúmeras variáveis contribuem para o sucesso ou o insucesso de um projeto de desenvolvimento de software: a complexidade técnica, a disponibilidade de pessoal capacitado e motivado para a gerência e para as demais atividades necessárias ao desenvolvimento do projeto, o emprego de técnicas e tecnologias adequadas e o conhecimento do que é para ser feito são apenas algumas delas.

Neste capítulo apresentaremos os conceitos e outros aspectos relacionados a essas variáveis: trataremos de qualidade, engenharia e processos de software, do uso da UML como linguagem de modelagem de sistemas. Trataremos, também, do uso de

ferramentas CASE. Nosso objetivo, em linhas bem gerais, é a conscientização quanto à importância do uso desses ingredientes para o desenvolvimento no preço, no prazo e com a qualidade esperados de um sistema computacional. Podemos começar pelo conceito de software.

2.1 O que é Software, Afinal?

Há alguns anos ouvi alguém dizendo que "software é produto projetado e construído pelos engenheiros de software". Se pararmos para pensar um pouco vemos que, por trás dessa definição, em princípio simplista, estão escondidas duas informações importantes. A primeira delas diz respeito a processo de produção, por conta da associação entre as palavras software e engenharia, em aparente choque com a forma como a produção de software era entendida antigamente, sendo resultado de uma atividade quase totalmente criativa. A segunda informação trata do que consiste o software, sendo não apenas linhas de código, mas o produto da atividade de desenvolvimento de software, que é muito mais do que conjuntos de comandos, como era comumente entendido.

Atualmente entendemos que software abrange programas de computador, dados e documentos, tangíveis (manuais impressos, por exemplo) e eletrônicos (arquivos em .pdf, por exemplo). Sendo assim, os modelos em UML que desenvolveremos, que são representações de conceitos por meio de figuras, fazem parte do software. Esse material com que lidamos durante o ciclo de vida (Figura 2.1) do software chamamos de conjunto de artefatos[1] que compõem o software.

O ciclo de vida de software compreende o tempo decorrido entre a percepção da necessidade do software e a sua retirada de produção. Muitos autores, no entanto, definem que o início do ciclo de vida coincide com o início do ciclo de desenvolvimento, quando é iniciado o projeto de construção dele, incluindo análise de viabilidade. A estrutura de chaves aninhadas a seguir ilustra o ciclo de vida, detalhando as etapas que compõem o ciclo de desenvolvimento interno a ele e que, por sua vez, pode ser entendido como sendo composto por quatro fases: concepção, elaboração, construção e transição. A Figura 2.1 ilustra.

Uma característica importante do software é que ele não desgasta, mas sim *deteriora*. Software deteriorado é o que deixa de atender a todas as necessidades dos usuários e, com isso, deixa de ser um produto de qualidade. Isso pode acontecer por vários motivos. Relacionamos dois:

[1] Artefato é qualquer documento ou arquivo produzido ou consumido durante o ciclo de vida de um software. Pode ser um módulo de código, um componente de software, um conjunto de dados de entrada de teste, um arquivo de modelo de uma ferramenta CASE, uma especificação textual em MS-Word, um manual etc.

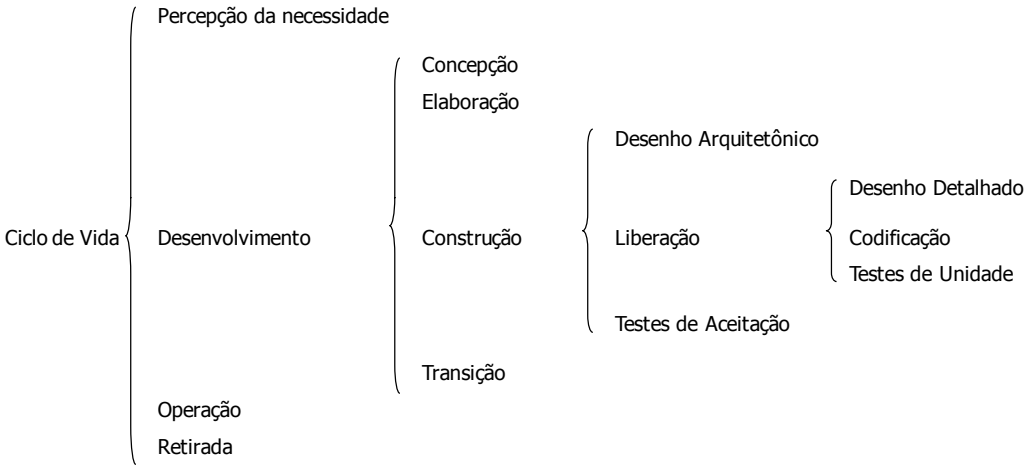


Figura 2.1: O ciclo de vida do software.

1. Mudança nas regras de negócio (leis, estatutos internos, boas práticas etc.);

2. Aumento dos dados a serem processados por expansão dos negócios. Como consequência, isso aumenta o tempo de resposta do sistema, podendo torná-lo inaceitável.

A deterioração é neutralizada por correções feitas no código, quando essas correções são possíveis e viáveis.

Mencionamos a palavra *qualidade* relacionada a software. Esse é um assunto importante, merecendo que o tratemos com maior profundidade.

2.2 Qualidade de Software

Suponha que nós contratamos o desenvolvimento de um sistema para um grupo de profissionais. Durante algum tempo mantivemos contatos com esse grupo no sentido de estabelecer o que o sistema deveria fazer, a quantidade de dados que ele precisaria processar na unidade de tempo, o padrão dos usuários (tipicamente tratando da proficiência dos usuários em informática e de possíveis limitações físicas), o ambiente operacional que deveria ser usado e restrições diversas, dentre outros itens.

O que esperamos de um serviço prestado com qualidade é que toda informação que passamos para a equipe de desenvolvimento fosse levada em consideração e que o produto final estivesse de acordo com nossas necessidades.

Pressman ([12]) considera que software de qualidade é o que está em conformidade com:

- Requisitos funcionais e de desempenho explicitamente estabelecidos;
- Padrões de desenvolvimento explicitamente documentados;
- Características implícitas que são esperadas em todo software desenvolvido profissionalmente.

Pressman acredita que possuir as funcionalidades solicitadas (atender aos requisitos funcionais) é a base para a medição da qualidade. Os padrões de desenvolvimento explicitamente estabelecidos definem de antemão um conjunto de critérios que guiam o processo de desenvolvimento. Esses padrões dizem respeito aos nomes de variáveis, classes, componentes e pacotes, tratando inclusive da documentação e da forma de endentação e quebra de linhas dos programas. Isso torna o código mais manutenível (que possui a característica de ser mais facilmente mantido) e elimina esse tipo de decisões do processo de sua construção, tornando-o mais rápido.

Algumas dessas falhas percebidas em um ou outro ponto colocam todo o software sob suspeição.

Outra questão bastante ligada à qualidade do produto é a qualidade do processo de produção. Dizem que "de uma mesa organizada sai um bom trabalho", no sentido amplo de que, se o nosso ambiente de trabalho é organizado (e nesse ambiente incluímos o processo que usamos para fazer algo), é maior a chance de produzirmos algo de qualidade.

Curiosidade: Você se lembra do primeiro filme da série *O Parque dos Dinossauros*? Lembra-se da desorganização que era a mesa do "profissional" que desenvolveu o software que controlava os portões do parque? O objetivo era passar a sensação de que daquele ambiente desorganizado não sairia um resultado de qualidade – como de fato aconteceu.

Deve-se dar, portanto, mais ênfase à qualidade do processo de produção do software, na medida em que a qualidade do software é vista quase como uma consequência.

Existem vários modelos de qualidade de processo, cada um com sua forma de medição da qualidade. Um deles é o *Capability Maturity Model Integration* (integração do modelo de nível de maturidade) – CMMI, que categoriza empresas em níveis de maturidade segundo os quais elas gerem seus processos de software. O CMMI organiza e auxilia a melhoria do processo de desenvolvimento de software em uma organização e é adotado, por exemplo, pelo departamento de defesa americano – o DoD — para avaliação de seus fornecedores de software.

2.3 Engenharia de Software

Segundo o IEEE (*Institute of Electrical and Electronics Engineers*), engenharia de software é o estudo e a aplicação de procedimentos sistemáticos, disciplinados e quantificáveis ao desenvolvimento, operação e manutenção de software, ou seja, é a aplicação de práticas de engenharia no desenvolvimento de software.

A engenharia de software estabelece, portanto, que o ciclo de vida do software é gerido por meio de processos bem comportados, envolvendo a definição, antes do início do ciclo de desenvolvimento, das etapas do processo de desenvolvimento e produção (incluindo manutenção) e seus pontos de controle (os marcos), dos níveis de qualidade, dos custos e dos prazos. Assim, são três os ingredientes que dão suporte à engenharia de software: o planejamento, a metodologia e a disciplina.

O planejamento trata da definição das etapas, da sequência de realização dessas etapas, do estabelecimento das metas e dos recursos que serão utilizados, da definição da metodologia, das ferramentas e de tudo o mais que trata do que fazer no sistema e como fazer.

A metodologia é o conjunto de procedimentos e técnicas que serão usadas, normalmente já estabelecidas por experiência anterior, pela academia ou pela instituição a qual estamos ligados.

A disciplina é o esforço que precisa ser desenvolvido no desenrolar do processo, no sentido de cumprir rigorosamente o planejamento, de acordo com a metodologia escolhida.

Como os graus de complexidade dos projetos têm se tornado cada vez maiores, à medida que as atividades que os compõem são cada vez em maior número e se relacionam, no tempo, de forma cada vez mais intrincada, o ambiente deve contar com ferramentas que ajudam a aplicação da metodologia e da disciplina para a manutenção do planejamento. Com esse objetivo, são usados sistemas de controle de fluxo de trabalho (*workflow*), ferramentas de controle da comunicação, ferramentas CASE e ferramentas de gerência de configuração, entre outras (Figura 2.2).

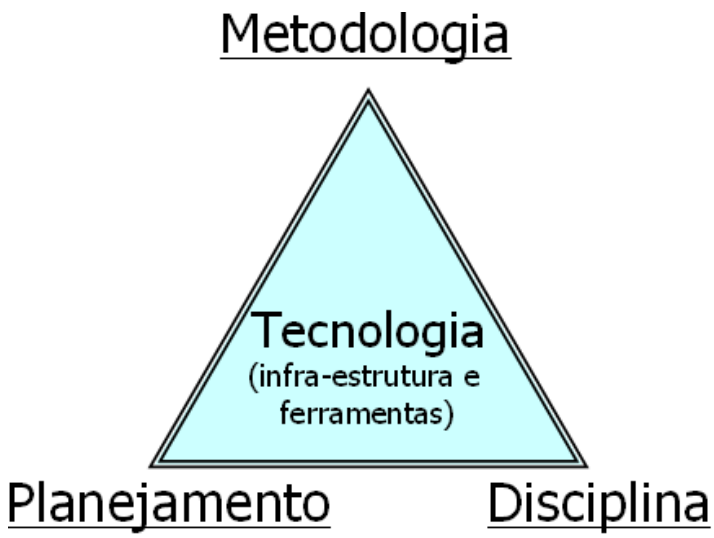


Figura 2.2: A tecnologia ajuda na aplicação da metodologia e da disciplina para a manutenção do planejamento.

A UML compõe a metodologia usada no planejamento de projetos de software orientados a objetos (OO). As ferramentas CASE fazem parte da tecnologia para a gestão do modelo UML.

2.4 Processos de Software

Parte da metodologia usada no planejamento de projetos de desenvolvimento de software trata do encadeamento das atividades técnicas de levantamento e especificação das necessidades dos usuários, especificação da solução (o sistema) dada pela equipe, construção dessa solução, testes, treinamento e colocação do sistema em produção. A esse conjunto de atividades e sua ordenação no tempo damos o nome de *processo (de construção) de software*.

Processo de software é, em outras palavras, um conjunto de atividades, métodos, práticas e transformações que pessoas empregam para definir, desenvolver e manter o software.

> **ATENÇÃO:** Lembre que software não é apenas linhas de código. Os produtos a ele associados, como plano do projeto, documentos do projeto, casos de teste, manuais do usuário etc., também fazem parte do software.

Um processo de software envolve:

- Pessoas com habilidades, treinamento e motivação;
- Procedimentos e métodos definindo o relacionamento entre as tarefas que fazem parte do processo;
- Ambiente adequado para gestão do processo, o que inclui as ferramentas de software (compiladores, programas de gestão da comunicação e da configuração, entre outros) e os equipamentos necessários para processar as ferramentas e os demais itens de hardware.

Qualquer processo de software possui, como vimos, uma série de tarefas definidas que são organizadas de diferentes formas, de acordo com o modelo de processo adotado. Essas tarefas demandam marcos para verificação, documentação e garantia da qualidade do software.

Os modelos de processos de software mais conhecidos são (a ordem não tem um significado especial):

- Em cascata (ou clássico), que divide o ciclo de desenvolvimento em cinco fases: levantamento, análise, projeto, implementação (também chamada codificação) e implantação[2]. O processo em cascata possui um longo ciclo de desenvolvimento (tipicamente meses), ou seja, os programas são construídos e entregues bom tempo depois do levantamento das necessidades. Por isso, não é muito usado atualmente, mas tem sido considerado como base para quase todos os modelos de processo, inclusive os mais usados atualmente, pois as atividades executadas continuam sendo necessárias, apenas mudando de nome e de forma de execução nos outros processos.
- RUP (*Rational Unified Process*), baseado no modelo UP (*Unified Process*) e especificado pela empresa *Rational*, que foi adquirida há algum tempo pela IBM. O RUP é, na realidade, uma base de conhecimento composta de mais de 3.200 modelos de artefatos (planilhas, arquivos de texto etc.) que podem ser adaptados para a organização e para o projeto. Possui pontos de controle (ou marcos) muito bem definidos, servindo como um guia bastante completo para a equipe e para os gerentes de projeto.

[2]Levantamento é a atividade em que relacionamos todas as necessidades dos usuários. A análise trata da especificação do que é para ser feito no sistema, usando uma linguagem precisa, usualmente gráfica. Projeto é a atividade de especificação da solução dada pela equipe de desenvolvimento do software. Ele também usa uma linguagem precisa, idealmente a mesma usada na análise. Implementação é a atividade de construção (programação) do sistema. A implantação trata da colocação do sistema à disposição dos usuários.

- XP (*Extreme Programming*), um dos métodos ditos ágeis, sendo bastante usado atualmente, pois é "leve" com respeito à documentação externa produzida, desonerando a equipe desse trabalho e focando na produção de código que atenda o mais rapidamente aos usuários. Não é exatamente um modelo de processo, mas sim uma coleção de práticas a serem adotadas pela equipe de desenvolvimento. Por essa razão vem sendo adotado atualmente em conjunto com modelos de gerência de projetos, notadamente o SCRUM.

A atividade de modelagem no processo em cascata e no RUP produzem artefatos que se tornam partes integrantes da documentação de análise e projeto e, portanto, do software. No XP, os modelos UML desenvolvidos não têm muito valor como documentação, servindo apenas para discussão das possíveis alternativas de solução. No XP, os modelos, quando necessários, devem poder ser obtidos a partir do código.

> **ATENÇÃO:** Os modelos de processo surgiram como soluções para tornar o processo de desenvolvimento de software uma atividade sistemática, disciplinada e quantificável, de engenharia, portanto, em contraposição a praticas de desenvolvimento do tipo "codifica-remenda", infelizmente ainda usadas hoje em dia.

2.5 Modelos e Modelagem de Software

O conceito de modelo de processo de software discutido até aqui diz respeito a uma das acepções do termo que se refere a *padrão*. No entanto, como o texto é de modelagem de sistemas com UML, é importante, neste ponto, conceituar *modelo*, explicando também a necessidade de desenvolver modelos para a construção de sistemas.

Modelo é a representação de algo, é uma abstração da realidade e representa uma seleção de características do mundo real que são relevantes para o propósito para o qual o modelo foi construído.

Na frase anterior, a palavra *representação* pretende deixar claro que modelo não é a realidade, mas sim algo que permite descrevê-la. Por exemplo: quando pensamos em comprar um apartamento "na planta" temos de tomar nossa decisão com base em modelos que correspondem a uma representação do apartamento que não existe ainda.

A palavra *abstração* refere-se ao "esquecimento" do que não é importante em determinado momento. No caso da compra do apartamento na planta, o primeiro modelo que costumamos olhar é a planta baixa, que seleciona apenas algumas

características da construção (dimensões e disposição dos cômodos, janelas e portas, dentre outras). Outro modelo que comumente consultamos é a maquete, que seleciona características como posição do prédio dentro do terreno e com respeito às construções vizinhas, características do acabamento externo etc. Na maquete são abstraídas muitas características representadas na planta baixa e vice-versa. O engenheiro que irá supervisionar a construção necessitará de outros modelos que não são importantes para quem vai comprar o imóvel: o modelo (também chamado de projeto) de hidráulica, o de eletricidade, o de estrutura... Cada modelo tem, portanto, sua utilidade e seu "público alvo". Antes de ser construído, o prédio só poderá ser bem compreendido por meio de modelos quando juntarmos todas as características representadas em cada modelo.

Trazendo esses conceitos para o contexto deste texto, podemos definir, resumidamente, modelos de sistemas como representações ordenadas, estruturadas e consistentes do conhecimento a respeito do sistema.

Um determinado modelo não pode ser considerado "o melhor" em termos absolutos, na medida em que a modelagem é um processo iterativo, ou seja, modelos sempre evoluem em precisão, conforme nos debruçamos sobre eles e os refazemos. A cada exame de um modelo, a equipe descobre uma forma mais concisa e precisa de especificar o que quer. Costuma-se classificar modelos como "errados" – aqueles que não especificam de forma aceitável a realidade – e "corretos" – os demais modelos. É natural, então, que alguns modelos corretos sejam mais precisos que outros.

As dimensões de um modelo são o conjunto das características da realidade que são enfatizadas nesse modelo. Modelos dos sistemas de informação possuem três dimensões:

1. Dimensão de dados, que especifica as estruturas de informações e seus relacionamentos;

2. Dimensão funcional, que especifica as transformações das estruturas de informações;

3. Dimensão temporal, também chamada dimensão de controle, que especifica as sequências de acessos aos dados e de execução das funções.

O modelo de um sistema que não possui uma dessas dimensões não é um modelo completo.

A atividade de modelar consiste em criar um modelo da parcela do mundo real que é de nosso interesse.

ATENÇÃO: Modelagem é idealmente uma atividade em equipe. Mesmo que um modelo seja criado por um único membro da equipe, é bastante recomendado que ele seja apreciado pelos demais membros da equipe de modelagem, em conjunto com os especialistas do negócio/usuários (as pessoas que conhecem bem as características do negócio e que entrevistamos durante o levantamento).

A modelagem possibilita o estudo do sistema, já que modelos nos permitem entender como os sistemas funcionam ou funcionarão, além de permitir relacionarmos e "experimentarmos" alternativas de como ele poderá ficar melhor. Modelagem permite que as nossas suposições se tornem visíveis e, portanto, disponíveis para revisão e correção. A modelagem possibilita a discussão de correções, modificações e validação com o cliente e com a equipe a um custo baixo. Isso é possível porque a discussão ocorre usando um ambiente de simulação, utilizando agentes e insumos "virtuais" e tempos menores em relação aos necessários para a experimentação em ambiente real. A modelagem facilita a comunicação entre os membros das equipes de análise e projeto e entre eles e os clientes e usuários.

Os modelos são construídos usando-se linguagens que permitem especificar completamente, sem ambiguidade, regras de negócio, aspectos estruturais (neles se incluem os conceitos do negócio, os relacionamentos que eles mantêm entre si, dentre outros), sequências de operações e demais aspectos de ordenação temporal necessários para a realização dos propósitos do sistema. Dessa forma, o entendimento, por todos da equipe, de todos os aspectos do processo, é o mesmo. A modelagem permite, ainda, documentarmos sistemas, registrando todas as suas características, as decisões tomadas ao longo do projeto e os demais aspectos necessários à total compreensão e operação correta do sistema.

As linguagens de especificação gráfica aliam a expressividade à concisão e à facilidade de emprego. Essas linguagens podem ainda possuir mapeamentos (traduções) para código e outras formas textuais processáveis por computadores que nos ajudam a manipular os elementos gráficos do modelo, ao mesmo tempo que garantem a sua consistência.

2.6 Recursos Usados em Modelagem e Construção de Sistemas

Diante do desafio de tornar-se o desenvolvimento de sistemas uma atividade de resultados mais precisamente previsíveis, a comunidade de desenvolvedores passou a adotar recursos que, mais adiante no tempo, se tornaram parte das primeiras metodologias para análise de sistemas. Esses recursos, aplicados desde então, tiveram

como base cinco técnicas usadas desde a antiguidade para a solução de problemas em áreas diversas da atividade humana, tendo sido orientadas para desenvolvimento de sistemas, como apresentamos a seguir.

São cinco os recursos utilizados para a modelagem e construção de sistemas computacionais:

1. Abstração, que consiste em "esquecer" o que não interessa em determinado momento. Iniciamos nossa abordagem no chamado nível conceitual, quando estamos interessados em aspectos de mais alto nível, mais gerais, ou seja, do domínio do problema. No início do desenvolvimento de um sistema não devemos pensar nos detalhes, nas questões de tecnologia, nos aspectos ditos físicos. Devemos nos concentrar no que o usuário realmente precisa em termos de informação, ou seja, devemos nos concentrar na descrição do problema. No extremo oposto, ou seja, imediatamente antes de escrevermos os programas, devemos ter todos os detalhes definidos, incluindo o desenho do banco de dados na tecnologia escolhida (fabricante do sistema de gerenciamento do banco de dados – SGBD, versão etc.), os nomes das classes, métodos e atributos e aspectos da linguagem de programação, entre outros, enfim, tudo que é necessário para a programação. Nesse nível, chamado nível de implementação, a abstração (o "esquecimento") é zero! No meio do caminho, entre o nível conceitual e o nível de implementação, estamos no nível de especificação, que recebe esse nome por estarmos exatamente especificando a solução que estamos construindo para o problema. O nível de especificação começa assim que adicionamos o primeiro aspecto de nossa solução à especificação do problema e termina quanto todos os detalhes dessa solução estão definidos.

2. Rigor, que consiste em adotar uma abordagem metódica e controles estabelecidos a priori. Já falamos sobre isso quando tratamos de engenharia de software (ver Seção 2.3).

3. Formalismo, que se expressa por meio do uso de linguagens de especificação precisas, usualmente gráficas, como diagramas de fluxos de dados (DFDs), diagramas de entidades e relacionamentos (DERs), diagramas da UML etc. O formalismo reduz a ambiguidade da linguagem de descrição, facilitando e tornando precisa a comunicação entre desenvolvedores e usuários e entre os membros da equipe de desenvolvimento.

4. Divisão e conquista, também chamada de divisão no domínio, que consiste em dividir o problema em partes pequenas e, idealmente, independentes, de forma a poder tratar cada parte mais facilmente. Nossa expectativa é de que, dando solução para cada uma dessas partes, podemos compor as soluções para obter a solução do problema original, maior.

5. Organização hierárquica, também chamada de divisão no conceito, que consiste, tal qual a divisão no domínio, em dividir o problema em partes pequenas, dessa vez no conceito ou, em outras palavras, em níveis de abstração cada vez menores. A idéia é obter partes conceitualmente simples o suficiente de forma a descrever o problema e dar solução a ele mais facilmente. Nossa expectativa também é poder compor as soluções das partes do problema, obtendo a solução de todo o problema.

2.7 As Análises Estruturada e Essencial

Como dissemos, os engenheiros de software têm buscado sistematizar a aplicação desses recursos nas metodologias que desenvolvem. Nesse sentido, duas iniciativas precursoras procuraram aplicá-los de alguma forma: a análise estruturada e a análise essencial.

A análise estruturada empregava bem o recurso da organização hierárquica dos sistemas por meio dos DFDs e suas "explosões". Os métodos existentes estabelecem uma notação rigorosa e um processo formal para derivação dos modelos do sistema, ou seja, rigor e formalismo são, com isso, bem aplicados. Embora já houvesse a recomendação para tratar, na fase de análise, apenas os aspectos lógicos, havia a dificuldade de promover a abstração, provendo um isolamento entre as características tecnológicas do problema e o modelo conceitual. Com isso, a separação entre o lógico e o físico não era clara e a experiência do analista contava muito. Faltava também uma técnica para dividir o modelo em partes tão independentes quanto o possível, de forma a aplicar sistemática e efetivamente os recursos de divisão e conquista.

A análise essencial veio a seguir, com o intuito de resolver esses problemas. Com o uso dos conceitos da "tecnologia perfeita", as questões físicas eram sistematicamente postas de lado na fase de análise, e a aplicação do recurso de abstração era garantida. Além disso, por meio dos DFDs particionados por eventos, a divisão do sistema em partes independentes era também garantida.

Nesse tópico apenas "tocamos" nos conceitos das análises estruturada e essencial e talvez tenhamos passado a impressão de que as práticas preconizadas por elas sejam ultrapassadas. O fato é que, assim como a análise essencial herdou muita coisa de análise estruturada, a análise orientada a objetos (a análise OO), de que trataremos a seguir, também herdou boa parte de seus conceitos e das práticas das duas.

Se você está interessado em mais informação sobre análise estruturada e análise essencial, veja, por exemplo, os livros de Gane ([4]) e McMenamim ([3]). Há também muito material gratuito na Internet.

2.8 Análise e Modelagem Orientadas a Objetos (OOAD)

Com a análise essencial, os engenheiros de software poderiam, então, ter se dado por satisfeitos se não fossem dois outros aspectos que os incomodavam bastante:

- a separação entre dados e processos nos modelos. Os diagramas de fluxo de dados especificavam exclusivamente processos, e os diagramas de entidades e relacionamentos especificavam exclusivamente os dados, não sendo possível estabelecer na notação a associação existente entre os processos e os dados que processavam;
- a descontinuidade entre as fases de análise e projeto. O início do projeto obrigava que os analistas e projetistas aplicassem regras de derivação para transformação dos modelos de análise em modelos de projeto, obrigando-os, inclusive, a usar linguagens diferentes em suas especificações, dependendo do tipo de sistema que estava sendo modelado.

Essas questões motivaram a busca por um novo paradigma cujos benefícios se aliassem às conquistas realizadas, até então, no intuito de transformarem o desenvolvimento de sistemas em uma atividade de engenharia.

A análise e projeto orientados a objetos (OOAD – *Object Oriented Analysis and Design*) vieram ao encontro dessas expectativas. Considerando a forma como atuamos colaborativamente em uma organização, analistas e projetistas passaram a especificar e conceber sistemas cujas funcionalidades são realizadas por meio de colaborações entre objetos que encapsulam os dados e que executam operações. OO permite, portanto, reunir, em um mesmo conceito (e no mesmo modelo, como veremos mais à frente em nosso texto), dados e operações.

A busca para um formalismo adequado, permitindo especificar as características do sistema independentemente do tipo de sistema que está sendo tratado e do estágio dentro do ciclo de desenvolvimento, resultou na criação de inúmeras notações e metodologias ao longo dos anos 1985-1995. Felizmente, em 1997, surgiu a UML, que unificou as notações, vindo resolver esses últimos entraves nos projetos de sistemas computacionais.

2.9 UML – Breve História e Objetivos

Para que o entendimento de um modelo se faça de forma inequívoca, é necessário adotar para sua especificação uma linguagem cujos símbolos tenham semânticas (significados) e regras de combinação definidos precisamente. E ainda, para que o

modelo seja suficientemente completo, a linguagem deve ser expressiva, de forma a descrever concisamente todos os aspectos necessários da realidade modelada.

Existem várias linguagens que atendem satisfatoriamente, em graus diferentes, a essas necessidades. A UML é uma delas.

O mercado de ferramentas e de metodologias para desenvolvimento de sistemas estava bastante dividido e competitivo em meados da década de 1990, com inúmeros autores importantes liderando grupos independentes de simpatizantes de suas ideias a respeito de análise e projeto orientados a objetos. Cerca de cento e cinquenta métodos surgiram até 1995, dando origem principalmente a dúvidas com respeito à notação a ser usada e à falta de uma linha a ser seguida pela indústria de produção de ferramentas CASE. Isso caracterizou o que se chamou de "a guerra dos métodos".

Nessa época, três dos grandes nomes em OOAD – Grady Booch, Jim Rumbaugh e Ivar Jacobson – juntaram forças na Rational Corporation com o intuito de combinar seus métodos de desenvolvimento de software em uma maneira única de trabalhar, que eles imaginavam se tornaria um padrão da indústria de desenvolvimento de software. Os três entenderam que um primeiro passo nesse sentido seria a unificação das notações para especificação dos aspectos de análise e projeto OO. Esse esforço resultou na submissão de uma versão inicial da UML ao *Object Management Group* – OMG –, entidade que representa toda a indústria de desenvolvimento de software OO. Após muita discussão e alguma contribuição externa, a OMG adotou em 1997 a UML na versão 1.1. A partir daí, a UML rapidamente se tornou um padrão da indústria para a documentação de software desenvolvido segundo o paradigma OO. Atualmente, a UML está na versão 2.2 e continua sendo atualizada com base no feedback da indústria e da academia.

A UML serve para construir modelos concisos, precisos, completos e sem ambiguidades, tendo, de maneira geral, as seguintes características:

- Modela os aspectos estruturais (estáticos) e comportamentais (dinâmicos) do sistema, ou seja, pode especificar os conceitos do negócio e seus relacionamentos (invariantes com o tempo) e os estados, sequências de atividades e de colaborações (aspectos que contemplam a dimensão temporal, ou seja, que variam conforme o tempo passa). Em outras palavras, a UML provê elementos de notação para modelar dados, funções de transformação dos dados e as restrições aplicáveis aos dados e às funções, como regras de negócio, por exemplo. Essas características são necessárias, como já dissemos, à produção de bons modelos.

- Provê uma linguagem que permite o entendimento e a utilização por humanos e a leitura por máquinas. Além dos elementos gráficos da notação que são compreensíveis aos humanos (desde que, obviamente, conheçam a

linguagem), a UML conta com mecanismo padronizado para mapeamento entre a representação gráfica do modelo e a sua representação textual em XML (*Extensible Markup Language* – Linguagem Extensível de Marcação). A representação textual facilita o intercâmbio do modelo entre ferramentas de modelagem de fabricantes diferentes e possibilita a exportação desses dados para outras ferramentas, com finalidades diversas.

- Provê elementos de notação para modelar todos os tipos de sistemas de computação.
- Permite a modelagem do conceito ao artefato executável, utilizando técnicas OO. Usando os mesmos elementos da notação, podemos modelar desde os aspectos do negócio associados a níveis de abstração maiores até os níveis de implementação, associados a níveis de abstração "zero" (nenhuma abstração). Podemos especificar, portanto, negócios e sistemas com o uso de uma única linguagem, o que permite, quando necessário, a transição natural entre modelos de negócio e modelos de sistema.
- Embora a especificação já contenha elementos de notação que permitem o atendimento a um grande número de situações e propósitos, a linguagem é extensível e adaptável a necessidades específicas; palavras-chave permitem que se modifique a semântica de elementos da linguagem. Com o uso de palavras-chave é possível manter um conjunto relativamente reduzido de elementos gráficos da notação, porém permitindo a adaptação da UML para uso em modelagem em domínios.
- Contempla as necessidades de produção de modelos pequenos e simples a grandes e complexos. A UML possui diversos conectores e contêineres, o que permite dividir os modelos em agrupamentos pequenos no domínio e em níveis de abstração, de forma a torná-los compreensíveis independentemente da complexidade (se devida ao tamanho do que está sendo estudado ou ao domínio que está sendo tratado).
- Modela processos manuais ou automatizados, estes independentemente da tecnologia que usam.
- É uma linguagem para visualização do modelo, facilitando o entendimento pelas equipes de análise de negócio, desenvolvimento de sistemas e pelos clientes.
- Serve para construir código de computador, embora não seja uma linguagem de programação de computadores.
- A UML é o padrão *de facto* usado em análise e projeto de sistemas de informática orientados a objetos.

- Está em evolução, mesmo após mais de dez anos da publicação da versão inicial. Ainda são publicadas, com certa frequência, novas versões resultantes das discussões entre membros de diversas áreas da indústria e da academia.

ATENÇÃO: Embora o intercâmbio de modelos entre ferramentas CASE esteja previsto com o uso de um padrão, o XMI[DI] – *XML Metadata Interchange, Diagram Interchange* –, os produtores de ferramentas CASE até hoje não adotaram plenamente o padrão por questões de mercado e, com isso, o intercâmbio não é possível entre muitas ferramentas.

A UML, sendo uma linguagem gráfica, conta ainda com a facilidade de emprego, pois os elementos da notação são símbolos gráficos que podem ser compostos com a ajuda de ferramentas gráficas interativas que garantem a corretude e a consistência do modelo. Essas ferramentas estão amplamente disponíveis em diversas faixas de preços, muitas delas gratuitas.

Como consequência da extensão de sua especificação e da aplicabilidade em uma gama ampla de domínios, a UML é muitas vezes considerada intimidativa. No entanto, com um subconjunto mais reduzido de conceitos e elementos da notação – os que tratamos nesta apostila – contamos com um ferramental suficiente para tratar boa parte (senão a totalidade) das necessidades de modelagem de sistemas encontradas no dia a dia.

2.10 Resumo do Capítulo

Software é o produto da atividade de engenharia de software, abrangendo programas de computador, dados e documentos. Ele se deteriora quando deixa de atender às necessidades de seus usuários, por mudanças nas regras de negócio ou por não mais responder aos requisitos de performance. A deterioração é neutralizada por alterações apropriadas no código.

Um software de qualidade é aquele que atende aos requisitos funcionais e de desempenho explicitamente estabelecidos, foi desenvolvido conforme padrões de desenvolvimento previamente e explicitamente documentados e possui características que são esperadas em todo software desenvolvido profissionalmente, como manutenibilidade e usabilidade.

Um processo de software é um conjunto de atividades, métodos, práticas e transformações que pessoas empregam para definir, desenvolver e manter o software. Envolve pessoas com habilidades, treinamento e motivação, procedimentos e métodos definindo o relacionamento entre as tarefas que compõem o processo, além

de ferramentas e equipamentos adequados. Processos de software buscam manter o ambiente de desenvolvimento organizado, já que a qualidade do produto depende muito da qualidade do processo de produção.

Modelos de sistemas são representações ordenadas, estruturadas e consistentes do conhecimento a respeito do sistema. A modelagem possibilita o estudo do sistema e a discussão de correções, modificações e validação com o cliente, a um custo baixo. A modelagem facilita a comunicação entre os membros das equipes de análise e projeto e entre eles e os clientes e usuários.

A UML serve para construir modelos concisos, precisos, completos e sem ambiguidades, provendo uma linguagem extensível que permite modelar todos os tipos de sistemas de computação, o entendimento e utilização por humanos e a leitura por máquinas, além de atender os analistas e projetistas em todo o ciclo de vida do software.

2.11 Exercícios Propostos

1. Como os processos e a modelagem de software podem contribuir para a qualidade do software?

2. Os modelos são construídos para que possamos especificar, em uma primeira fase, as necessidades de nossos usuários. Mais adiante, eles permitem discutir com os demais membros da equipe as possíveis opções para a construção do sistema. Com base nas características e nos objetivos da UML que vimos, selecione aspectos da linguagem que ajudam nessas duas fases, explicando, de forma sucinta, o porquê da seleção que você fez.

As soluções encontram-se a partir da Página 201.

CAPÍTULO 3

DIAGRAMAS DE CASOS DE USO: CONCEITOS E ELEMENTOS BÁSICOS DA NOTAÇÃO

> What we see depends mainly on what we look for.
>
> John Lubbock

João encontrou sobre a mesa de trabalho dele um bilhete do chefe contendo o seguinte texto:

> *"João, temos que resolver os assuntos pendentes o quanto antes. Faremos uma reunião, eu, você e o diretor. Esteja em sua sala hoje às 15h."*

Onde, afinal, João ficará esperando a reunião acontecer? Na sala onde trabalha ou na sala do diretor?

Vimos no Capítulo 2 que a qualidade de um sistema está fortemente associada ao atendimento às necessidades da clientela, dos patrocinadores e dos usuários desse sistema, no que diz respeito, em última instância, às funções que o sistema precisa executar. Assim sendo, talvez a atividade mais importante durante a análise de um sistema seja a conversa com o cliente. Na conversa, precisamos compreender e

registrar corretamente as necessidades do cliente com respeito ao sistema que será desenvolvido, de forma que os demais membros da equipe entendam e não tenham dúvida sobre o que foi registrado.

A atividade de compreensão e registro das necessidades do cliente é conhecida como *levantamento (e captura) dos requisitos do sistema*. Poderíamos, claro, fazer os registros em linguagem coloquial, em formato livre, mas textos escritos dessa forma são susceptíveis a ambiguidades, tal qual o bilhete que João recebeu de seu chefe... E nós queremos evitar isso.

Neste capítulo iniciaremos o estudo dos diagramas de casos de uso da UML, que são usados para especificar os requisitos funcionais de um sistema. Esses diagramas são também usados para confirmar com o cliente o que ele disse durante o levantamento e para passar essas informações, de forma precisa, sem ambiguidade, à equipe de projeto e construção do sistema.

3.1 Enfoques dos Diagramas de Casos de Uso

Os diagramas de casos de uso da UML têm dois enfoques: o de negócios e o de sistemas. Os dois enfoques são úteis, mas precisamos distingui-los porque, durante o trabalho de análise, eles são usados em tempos diferentes. Eu costumo ilustrar as diferenças entre os dois enfoques usando a Figura 3.1.

Imagine um atendente de balcão do INSS que atende segurados em suas diversas necessidades. O primeiro segurado da fila deseja executar o processo de negócio `Averbar Tempo de Serviço`. Para isso, o atendente consulta um arquivo convencional, pega um ou mais formulários sobre a mesa e, possivelmente, consulta o tempo de contribuição do segurado no sistema.

Durante a realização desse processo de negócio, o atendente e o segurado trocam informações e documentos, ambos participando, portanto, do processo. Os dois são ditos *atores do negócio*, pois cada um deles desempenha seu papel específico no processo de negócio. O atendente também consulta o tempo de contribuição do segurado no sistema. Por usar o sistema, o atendente é *ator do sistema*, além de ator do negócio. O segurado, no entanto, não interage com o sistema e, portanto, não é ator do sistema. É importante ressaltar que, mesmo que o atendente retire um relatório do sistema e o entregue ao segurado, este não se torna um ator do sistema por isso.

Um indivíduo só é considerado ator de um sistema se ele é usuário desse sistema, ou seja, se ele insere um dado, pressiona um botão ou tecla, passa um cartão, toca a tela e seleciona uma opção, rola uma tela ou se identifica biometricamente, interagindo diretamente com o sistema.

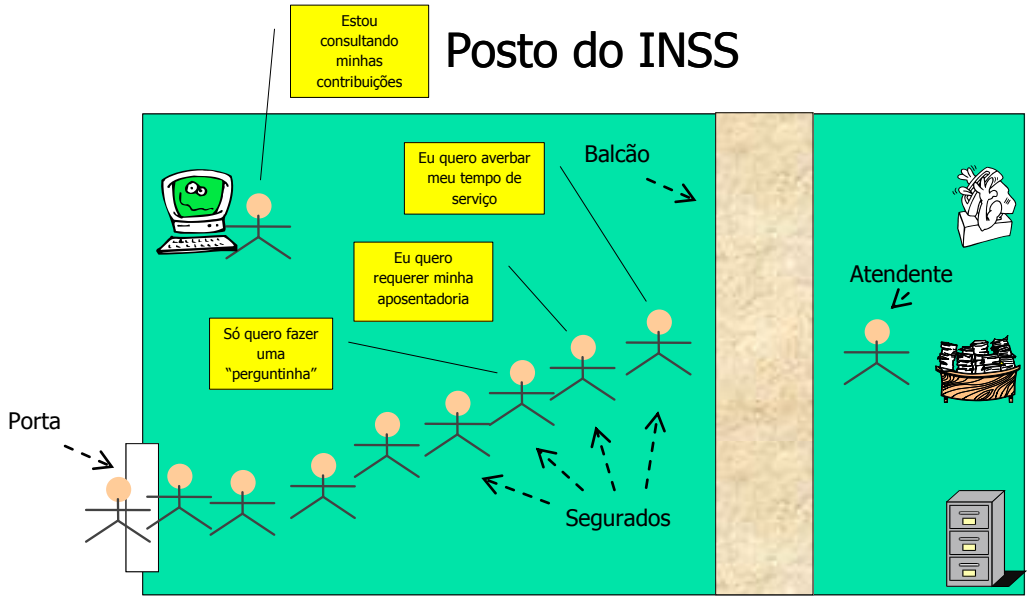


Figura 3.1: Casos de uso de negócio e casos de uso de sistema sendo executados no posto do INSS.

Situação análoga pode acontecer com os demais processos de negócio dos quais o atendente do INSS participa: `Registrar Requerimento de Aposentadoria` e `Retirar Dúvidas`.

A Figura 3.1 também ilustra a situação em que um segurado precisa consultar o seu tempo de contribuição e acessa diretamente outra funcionalidade do sistema do INSS: `Consultar Tempo de Contribuição Via Terminal do Cidadão`. Nesse caso, o processo de negócio do INSS `Informar Tempo de Contribuição` é realizado totalmente pela funcionalidade `Consultar Tempo de Contribuição Via Terminal do Cidadão`. O segurado é ator de um processo de negócio e de uma funcionalidade de sistema.

De maneira geral, processos de negócio envolvem relações entre organizações, entre organizações e seus clientes e fornecedores, entre colaboradores em uma organização e entre todas as demais entidades que precisam colaborar de alguma forma para que as organizações cumpram seus objetivos. Esse é o enfoque de negócio e, segundo esse enfoque, os casos de uso de negócio representam os processos realizados e os atores de casos de uso de negócio são todos os que participam

deles. Eventualmente, um processo em uma organização é informatizado, parcial ou totalmente. As funcionalidades, demais características desse sistema e seus usuários dizem respeito ao enfoque de sistema. Segundo esse enfoque, os casos de uso de sistema representam as funcionalidades do sistema; os atores dos casos de uso de sistema são seus usuários.

Os processos de negócio podem ser modelados com o uso de diagramas de casos de uso de negócio e as funcionalidades de um sistema podem ser modeladas com o uso de diagramas de casos de uso de sistema. A notação usada nos dois pode ser exatamente a mesma, contanto que mencionemos no diagrama a que enfoque corresponde.

Casos de uso de negócio não são o foco deste texto, mas eu pessoalmente estimulo todos os analistas a realizarem o estudo dos processos de negócio e, portanto, a desenvolverem o diagrama e a descreverem os casos de uso de negócio como forma de conhecerem bem os processos que irão automatizar com a criação dos sistemas.

Elaboramos o diagrama de casos de uso de sistema para representar os requisitos funcionais, ou seja, as funções que deverão estar disponíveis no sistema para que as necessidades que motivaram a sua construção sejam satisfeitas. Este é o enfoque do texto. Por isso, ao mencionarmos simplesmente "caso de uso" e "ator" estaremos nos referindo neste contexto a "caso de uso de sistema" e "ator de caso de uso de sistema", respectivamente.

A seguir apresentaremos a notação gráfica usada nos diagramas de casos de uso e os conceitos de cada elemento da notação. Na Figura 3.2 são ilustrados seis atores e dois casos de uso de um Sistema de Registro de Compras e Devoluções de um supermercado hipotético.

3.2 Os Atores

O termo *ator do sistema* se refere ao papel que alguém ou alguma coisa interpreta enquanto interage com o sistema sendo modelado. No diagrama da Figura 3.2, os "bonequinhos" representam atores do Sistema de Registro de Compras e Devoluções. A UML se refere à representação gráfica como sendo de *stick men*, ou seja, bonecos feitos de linhas, de forma bem simples.

Embora, em boa parte das vezes, atores sejam seres humanos, eles também podem ser outras coisas, como dispositivos eletrônicos ou outros sistemas computacionais que se relacionam com o sistema em estudo.

Um único indivíduo pode interpretar o papel de vários atores (por exemplo, Joel, além de ser – ou interpretar o papel de – caixa, pode atuar como o atendente de balcão);

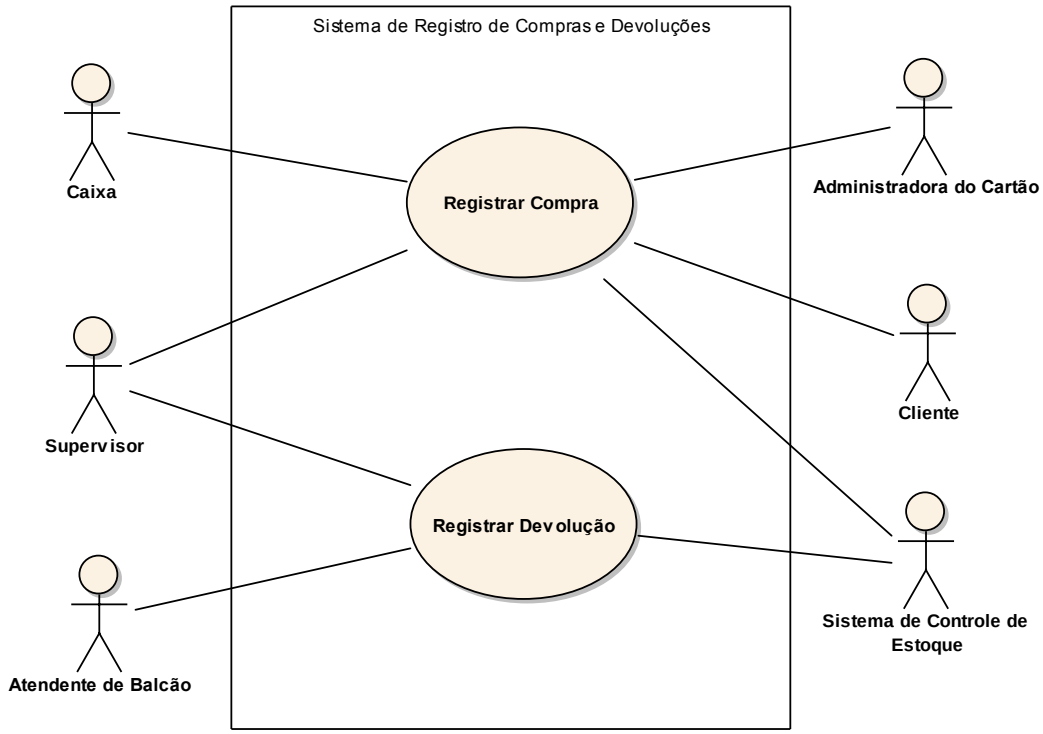


Figura 3.2: Diagrama de casos de uso de um Sistema de Registro de Vendas e Devoluções de um supermercado hipotético.

vários indivíduos podem interpretar o papel de um único ator (por exemplo, Joel e Pedro podem ser, ambos, atores caixa). Atores podem participar de um ou mais casos de uso; na Figura 3.2, os supervisores podem participar de registros de compras e de devoluções.

O nome do ator, idealmente uma expressão breve no singular, deve sugerir claramente o papel que o ator representa, dentro do jargão do negócio, ou seja, não deve ser, por exemplo, uma expressão de uso restrito ao ambiente da equipe de modelagem. A Tabela 3.1 mostra alguns exemplos do que usar e do que não usar como nomes de atores.

A versão atual da UML permite representarmos um ator de uma forma gráfica mais sugestiva quanto ao seu tipo, ou seja, atores sistemas podem ser representados por figuras de computadores. Outra representação alternativa é com a notação de

Tabela 3.1: Exemplos do que usar e do que não usar como nomes de atores.

Usar	***Não Usar***
Contador	Contadores
Sistema de Controle de Estoque ou simplesmente SCE	João
Sensor de Presença	Usuário

classes, ou seja, retângulos com a palavra-chave "«actor»" em seu topo, já que atores também podem ser entendidos como categorias ou classes de usuários dos sistemas.

Quando desenhamos o retângulo que representa os limites do sistema, os atores são colocados fora dele. Isso significa que, para o propósito do modelo a ser desenvolvido, não interessa saber como eles agem, qual a lógica de funcionamento e como são seus detalhes internos; o que interessa é apenas o que eles fazem durante a interação com o sistema que está sendo estudado.

Eles podem aparecer repetidos em um mesmo diagrama. Para muitos analistas, isso só tem *efeito cosmético*, pois possibilita eliminar cruzamentos entre relacionamentos, o que não se justifica pelo aspecto prático e, na maioria das vezes, de clareza do modelo. Isso, pelo contrário, só adiciona complexidade visual ao modelo.

Resta, agora, identificar os atores do sistema. Essa atividade parece, em princípio, simples, mas devemos ter em mente as diferenças entre participar do processo de negócio e interagir com o sistema.

Os atores são descobertos classificando-se os indivíduos que efetivamente usarão o sistema ou identificando-se o software – tipicamente outros sistemas – ou hardware externo que inicia um caso de uso do sistema ou que é necessário durante a execução desse caso e uso.

Não se pode ter certeza de que todos os atores foram descobertos antes de especificar[1] (descrevermos em detalhes) todos os casos de uso do sistema, pois durante a especificação é que entendemos quem faz o que no sistema.

Um ator não é uma pessoa específica. Quase sempre é possível achar pelo menos duas pessoas cujas responsabilidades e atividades se encaixam no perfil de um mesmo ator, isso para cada ator do modelo. Em outras palavras: se você não achou mais do que uma pessoa que interpreta o papel de determinado ator, é bem provável que você tenha modelado a pessoa, e não o ator, embora exista a possibilidade de um papel ser interpretado por somente uma pessoa. Um exemplo disso é em um Sistema de Controle de Clientes de uma padaria, em que o dono, o *Seu Manoel*, é o único que executa o caso de uso `Manter Lista Negra de Clientes`.

[1] Especificações de casos de uso serão vistas na próximo capítulo.

3.3 Os Casos de Uso

Um caso de uso corresponde a um conjunto de ações executadas durante a realização de uma funcionalidade do sistema. Casos de uso concentram-se nas relações entre as funções do sistema e os usuários que delas participam de alguma forma. Um caso de uso de sistema tem as seguintes características:

- Captura as ações para a realização de uma função do sistema, enfocando as interações entre os usuários e o sistema;
- É uma unidade coerente de passos, expressa como uma transação entre os atores e o sistema, compondo-se tipicamente de várias ações dos atores e respostas do sistema;
- É uma sequência de ações que produzem resultados observáveis de valor para os usuários;
- Expressa o que acontece quando um caso de uso é executado, incluindo suas possíveis variações. Não há preocupação em como os participantes executam suas ações, embora a ordem delas seja relevante.

Nos diagramas, os casos de uso são denotados por ovais ou elipses que representam as funcionalidades do sistema. Casos de uso têm nomes que devem ser ativos, ou seja, um verbo no infinitivo concatenado a um substantivo. Exemplos: `Aprovar Crédito`, `Registrar Venda de Automóvel` e `Aprovar Fatura`. Os nomes são colocados dentro ou abaixo das ovais. De certa forma, a colocação do nome sob a oval traz mais complexidade visual ao diagrama, já que nome e oval passam a constituir dois elementos visuais distintos.

Por exemplo: do caso de uso `Registrar Compra` da Figura 3.2 participam o `Caixa` (que registra os itens de compra no sistema), o `Cliente` (que digita a senha do cartão de crédito), a `Administradora do Cartão` (que aprova o débito do valor no cartão), o `Sistema de Controle de Estoque` (que é informado da compra para que possa controlar o estoque), e o `Supervisor` (que eventualmente retira um item de venda da lista de compras). A forma como eles participam não é especificada no diagrama.

Usamos, basicamente, duas técnicas para descobrir as funcionalidades do novo sistema:

1. Iniciando a partir da relação dos atores: para cada ator, identificar as funcionalidades de que necessita.

2. Iniciando a partir da relação dos eventos. Isso é feito em três etapas, conforme segue.

   a) Identificar os eventos externos aos quais o sistema deve responder;

   b) Associar os eventos aos atores que atuam para tratá-los;

   c) Identificar as funcionalidades que eles necessitam executar em resposta aos eventos.

A primeira técnica é a mais simples e a mais comumente usada. Apenas para ilustrar a segunda técnica, tomemos como exemplo o evento de recebimento de uma nota fiscal de entrega pelo setor de controle de estoque de uma organização. O encarregado do estoque que recebe a nota (o ator) precisa de uma funcionalidade (o caso de uso) para registrar a chegada dessa fatura, que indica a chegada de material para reposição de estoque. Essa funcionalidade poderia, por exemplo, adicionar os novos itens no estoque e indicar ao sistema de contas a pagar da organização que há um novo compromisso a ser pago.

3.4 A Fronteira do Sistema

A fronteira do sistema, também chamada limite ou escopo do sistema, é representada pelo retângulo que contém os casos de uso (veja a Figura 3.2). A representação da fronteira é opcional, segundo a UML; colocamos a fronteira quando queremos e podemos, já que, às vezes, não conseguimos definir uma fronteira retangular com os casos de uso dentro e os atores fora. A fronteira é colocada para salientarmos o que é o sistema e, portanto, é nosso interesse estudar. Estar fora da fronteira, em contrapartida, significa que não estamos interessados, para efeito de nosso estudo, nos detalhes internos e na lógica de seu funcionamento. Por essa razão, os atores do sistema ficam necessariamente fora da fronteira.

No topo do retângulo, internamente a ele, centralizado, colocamos o nome do sistema.

3.5 Relacionamentos em Diagramas de Casos de Uso

Com a UML podemos especificar os tipos diferentes de relacionamentos que existem entre atores, entre atores e casos de uso e entre casos de uso. Eles estão detalhados a seguir.

Associação

O único relacionamento possível entre um ator e um caso de uso leva o nome particular de *associação*. As associações especificam quais atores participam de quais casos de uso, sendo representadas nos diagramas de casos de uso por meio de segmentos de retas, curvas ou poligonais que ligam os atores aos casos de uso de que participam.

A UML admite que ambas as pontas das associações possuam *multiplicidades*. Quando um ator tem uma associação com um caso de uso com uma multiplicidade que é maior que um na ponta da associação do lado do caso de uso (veja a Figura 3.3), isso significa que o dado ator pode estar envolvido com múltiplas execuções (instâncias) do caso de uso. A natureza específica desse relacionamento múltiplo depende do contexto e não é definida na UML. Dessa forma, um ator pode iniciar uma ou mais instâncias de um mesmo caso de uso concorrentemente ou elas podem ser mutuamente excludentes no tempo. Por exemplo, um colaborador pode iniciar ao mesmo tempo um determinado tipo de atendimento a vários clientes no balcão ou instituir uma fila para atendimento um a um. Quando essa multiplicidade é maior do que um do lado do ator, significa que mais de uma instância daquele ator participa do caso de uso. A maneira como os atores participam depende do contexto (concorrentemente, de forma colaborativa ou numa sequência) e também não é definida na UML.

> **ATENÇÃO:** Multiplicidades serão vistas adiante com mais detalhes, quando tratarmos de diagramas de classes.

Quando não há multiplicidades nas pontas das associações é porque elas ainda não foram determinadas ou essa informação não é relevante para o propósito do modelo. Nesse caso, deve-se admitir que um ator participa de um número qualquer de instâncias do caso de uso e participa do caso de uso qualquer número de instâncias do ator (usuários daquela classificação).

As formas segundo as quais os atores participam dos processos de negócio não são capturadas nos diagramas de casos de uso. Casos de uso precisam ser descritos (especificados) para que os detalhes fiquem esclarecidos. Especificações de casos servem ainda para validação junto ao usuário, funcionando com um "contrato" entre o usuário e o desenvolvedor.

Especialização-Generalização de Atores e Casos de Uso

Os relacionamentos de especialização-generalização podem ocorrer entre dois atores e entre dois casos de uso. Esses relacionamentos são representados por setas com

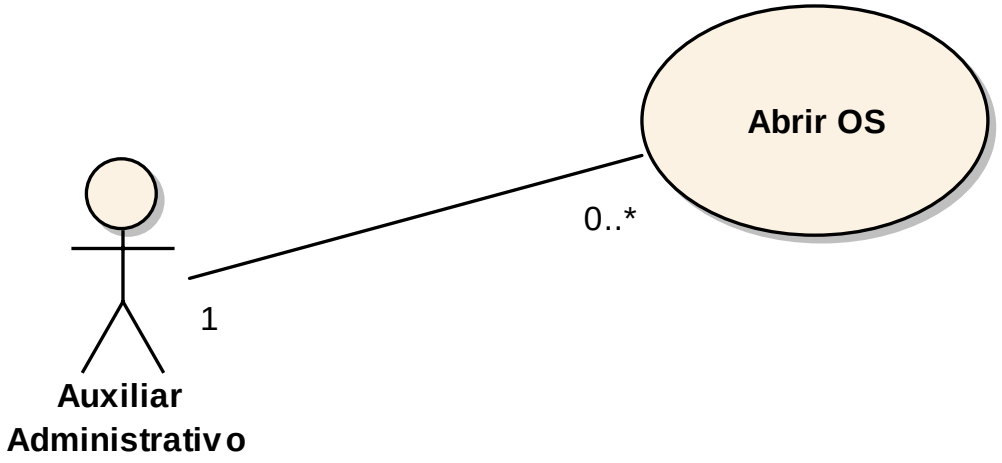


Figura 3.3: Multiplicidades nas pontas das associações entre atores e casos de uso.

pontas triangulares vazadas (veja as Figuras 3.4 e 3.5), que indicam o sentido da generalização; o sentido oposto é o da especialização.

Muitos atores podem interpretar o mesmo papel em um determinado caso de uso. A Figura 3.4 ilustra a situação em que o ator `Gerente de Vendas` aprova financiamentos, mas também pode atuar como `Vendedor`, vendendo automóveis. De fato, um gerente de vendas vai querer atuar como um vendedor (e ganhar toda a comissão pela venda) quando, por exemplo, um cliente antigo seu chega à loja de automóveis para trocar os cinco automóveis Mercedes-Benz da família, como costumeiramente faz todo início de ano.

Essas participações específicas, por sinal, são as diferenças de comportamento dos atores que justificam suas especializações. Isso quer dizer que o modelo da Figura 3.4 só se justifica se houver pelo menos um caso de uso associado a `Gerente de Vendas`.

Note que a recíproca não é verdadeira, ou seja, um vendedor não poderia entrar no sistema para registrar a aprovação de um financiamento.

ATENÇÃO: Especializações-generalizações de atores são os únicos relacionamentos possíveis entre atores em um diagrama de casos de uso.

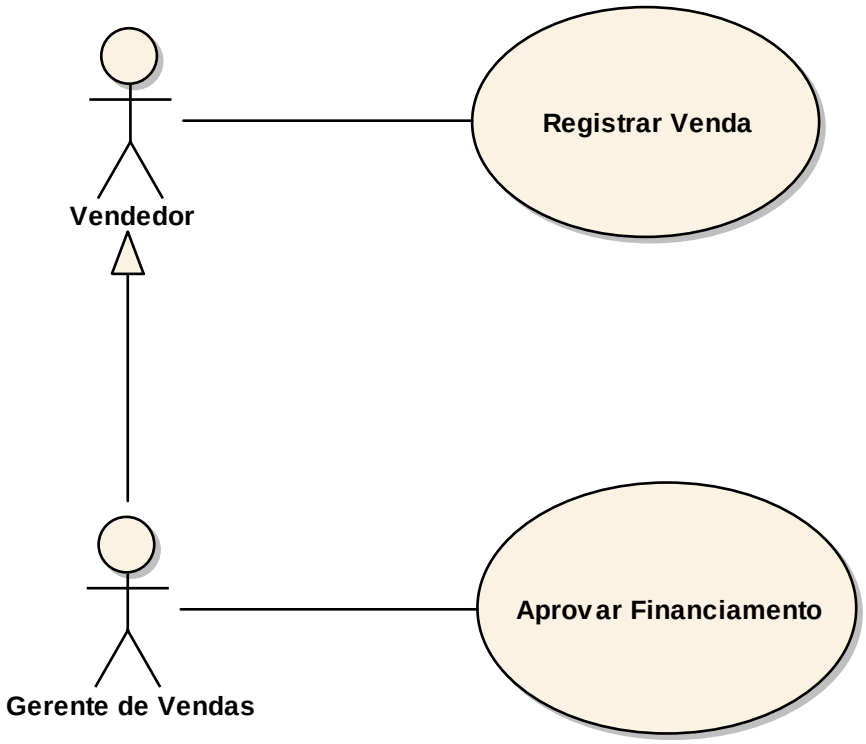


Figura 3.4: Especialização-generalização de atores.

A Figura 3.5 ilustra uma situação em que o vendedor registra a venda de um automóvel; esse processo pode ser feito de três maneiras ligeiramente diferentes:

1. para o caso em que o limite de crédito do comprador se encontra ultrapassado (caso de uso Registrar `Venda de Automóvel - Cliente Com Limite de Crédito Atingido`);

2. para o caso em que o cliente é um cliente habitual (caso de uso `Registrar Venda de Automóvel - Cliente Habitual`);

3. para o caso em que ocorre o processo básico, normal, que corresponde ao caso de uso base `Registrar Venda de Automóvel`. Esse caso de uso é chamado de caso de uso *base*.

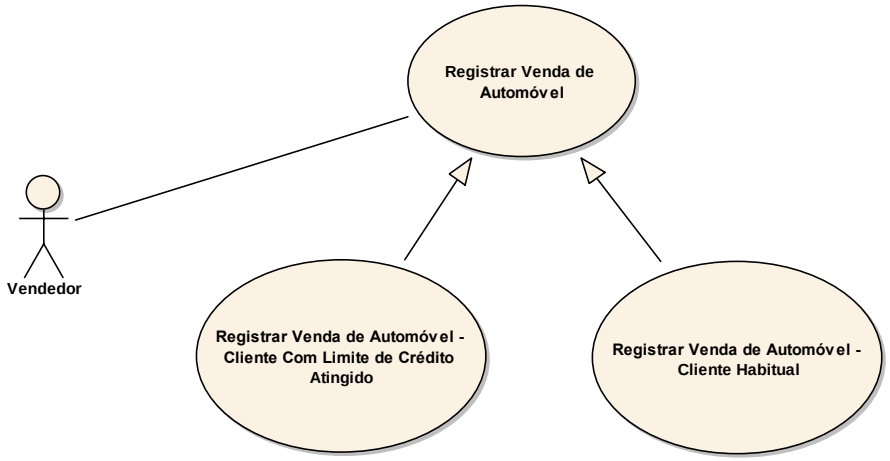


Figura 3.5: Especialização-generalização de casos de uso.

A especialização de um caso de uso significa, portanto, que o caso de uso que especializa (por exemplo, o caso de uso `Registrar Venda de Automóvel - Cliente Com Limite de Crédito Atingido`, da Figura 3.5) é um pouco diferente do caso de uso especializado (o caso de uso `Registrar Venda de Automóvel`), podendo acrescentar ou remover passos deste.

Na segunda edição do livro *UML Essencial* ([6]), Fowler e Scott dizem que usam especializações de casos de uso "quando se tem um caso de uso que é semelhante a outro, mas faz um pouco mais". Eles reforçam essa ideia quando dizem que usam especializações quando não querem ser muito precisos, o que equivale a dizer quando querem se manter em um nível de abstração mais alto.

O processo de venda de automóvel para clientes com limite de crédito atingido é realizado com base no processo básico para clientes que compram a crédito, possivelmente demandando outras informações do cliente, outras garantias e a anuência do gerente de vendas. O processo de venda para um cliente habitual possivelmente altera o prazo de entrega e agiliza o processo de coleta de informações do cliente.

A generalização sugere uma leitura do modelo na forma "é um tipo de". No exemplo da Figura 3.4 podemos ler: gerente de vendas é um tipo de vendedor. No exemplo da Figura 3.5 podemos ler: registrar a venda de automóvel para um cliente habitual é um tipo de registro de venda de automóvel.

O uso de especialização-generalização, como já dissemos, dota o modelo de um nível de abstração mais alto, o que em geral não perdura por muito tempo dentro do ciclo de modelagem; um pouco mais adiante no tempo será necessário especificar o quão diferentes (em quê, exatamente) são os casos de uso especializados entre si e em relação ao caso de uso base. Por essa razão, eu particularmente prefiro usar um tipo de relacionamento que se aproxima mais da forma precisa de especificação dessas diferenças: o relacionamento de extensão, que você verá mais adiante.

Inclusão de um Caso de Uso em Outro

Inclusão é um relacionamento somente usado entre dois casos de uso. Uma inclusão ocorre em um caso de uso quando é necessária a invocação do comportamento especificado em outro; em determinado momento da especificação de um processo "se chama" (se faz referência a) o outro. É como se fosse a chamada feita a uma sub-rotina, em um programa de computador.

Uma inclusão se faz necessária quando se tem um conjunto relevante de ações que é comum a dois ou mais casos de uso. Nesse caso, se fatora esse comportamento em um novo caso de uso, ou seja, se retira dos casos de uso esse conjunto de passos que é repetido, colocando-o em um caso de uso à parte.

A Figura 3.6 ilustra uma situação em que determinado restaurante opera de formas diferentes no almoço e na janta: no almoço ele oferece refeição por quilo e à noite a refeição é oferecida à la carte. Existe um conjunto de ações comum a ambos os casos de uso, que ocorrem obrigatoriamente e de forma idêntica segundo as regras do restaurante: o pagamento pela refeição.

O pagamento da conta, em geral, não é um processo trivial, pois envolve a escolha do meio de pagamento e o cumprimento das formalidades correspondentes. Não ser trivial significa demandar um número relevante de passos para a sua especificação, o que justifica a criação de um caso de uso para representá-lo no diagrama.

Outro aspecto importante é que o pagamento da conta é obrigatório, no almoço e no jantar... Pelo menos nesse restaurante, pois a inclusão está associada à obrigatoriedade; se o pagamento fosse facultativo, por alguma razão ou forma (se a despesa pudesse ser "pendurada" ou "deixada pra lá", por exemplo), não seria uma inclusão, e sim uma *extensão*.

Extensão de um Caso de Uso por Outro

A extensão também é um relacionamento que só ocorre entre dois casos de uso.

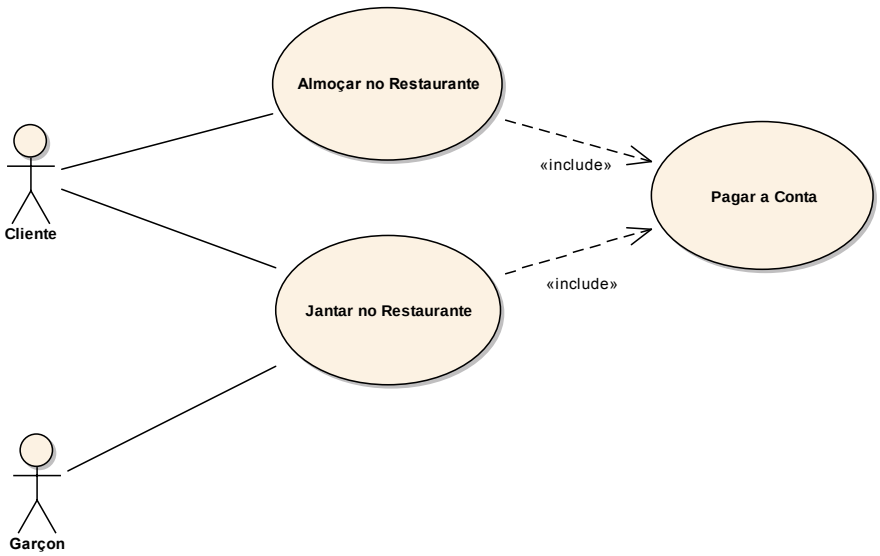


Figura 3.6: Relacionamento de inclusão entre casos de uso.

Extensões são essencialmente generalizações, só que possuem regras mais explícitas. Quando tratamos de generalizações/especializações, mencionamos que elas não perduram por muito tempo no ciclo de modelagem porque, mais cedo ou mais tarde, precisamos especificar as diferenças entre as especializações e entre elas e o caso de uso base, sob pena do nosso modelo ficar incompleto. Fowler e Scott recomendam que se use extensão "quando você estiver descrevendo uma variação do comportamento normal de um caso de uso e deseja utilizar a forma mais controlada". Essa forma mais controlada é feita especificando-se *pontos de extensão*, que não veremos neste texto. Cremos que modelar extensões sem os pontos de extensão, ao invés de generalizações, seja suficiente para cumprir os objetivos da modelagem de casos de uso no nível conceitual.

A extensão também significa a invocação de um caso de uso por outro, mas difere da inclusão na situação em que, de acordo com as regras do negócio, a invocação feita ao outro caso de uso não ocorre obrigatoriamente. Difere ainda porque a invocação ocorre no sentido inverso da seta.

Voltando à situação ilustrada pela Figura 3.6, imagine a situação em que o restaurante faz promoções para os almoços, de modo que os clientes habituais não pagam a refeição a cada determinado número de idas ao restaurante. Isso significa

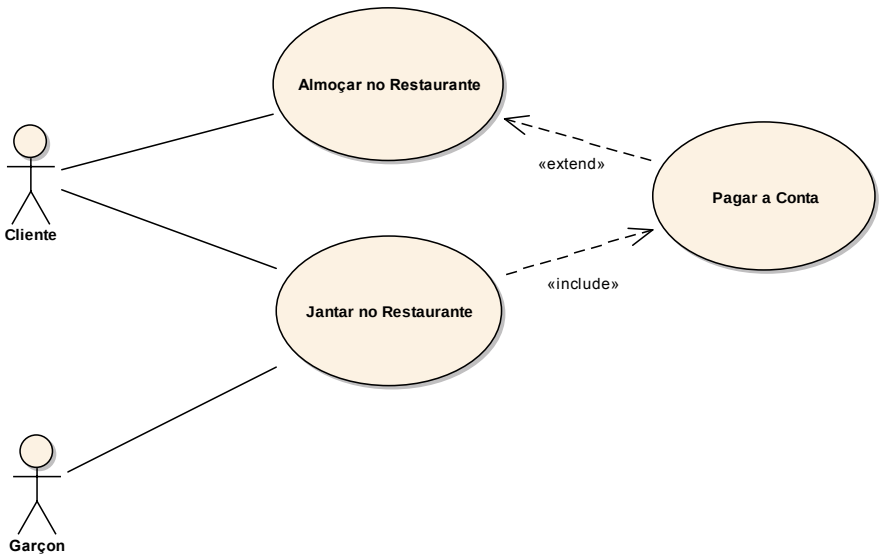


Figura 3.7: Relacionamento de extensão entre casos de uso.

que, de acordo com as regras do restaurante, o pagamento pelo almoço pode não ocorrer. Nesse caso, o modelo fica conforme ilustra a Figura 3.7 (lê-se `Pagar Conta` estende `Almoçar no Restaurante`).

Extensão significa que, em determinada posição da execução da funcionalidade representada pelo caso de uso, outro caso de uso é invocado opcionalmente. O nome extensão significa que o caso de uso invocado *estende* (aumenta) o caso de uso que invoca, acrescentando novas ações a ele.

Fechando a ideia sobre inclusões e extensões, os relacionamentos são sempre lidos no sentido da seta (que é tracejada, segundo a notação). Inclusão significa que o caso de uso apontado pela seta é incluído no caso de uso que o aponta. Extensão significa que o caso de uso que aponta estende (agrega mais passos a) o caso de uso apontado. Adicionalmente, a extensão ocorre opcionalmente e a inclusão ocorre obrigatoriamente, segundo as regras do negócio.

É importante ressaltar que uma inclusão, embora obrigatória pelas regras do negócio, pode não ocorrer. Tomemos como exemplo a situação ilustrada pela Figura 3.6, que especifica que o pagamento pela refeição é obrigatório; nada impede, no entanto, que o cliente drible a segurança, pule a janela e saia do restaurante sem pagar.

Temos um pequeno "macete" para descobrir se devemos usar inclusão ou

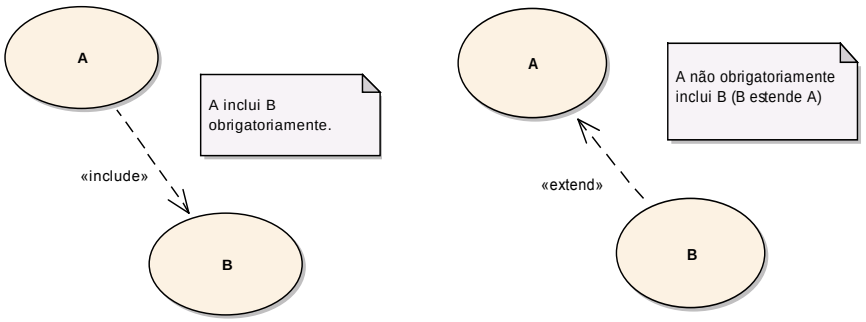


Figura 3.8: Uso de inclusão e extensão quando a inclusão ocorre obrigatoriamente ou não obrigatoriamente, respectivamente, segundo as regras de negócio.

extensão em um modelo: se, por exemplo, o caso de uso `A` chama em determinado ponto o caso de uso `B`, indicamos no modelo que `A` inclui `B` (o relacionamento de inclusão aponta de `A` para `B`). Nesse ponto perguntamos: essa inclusão ocorre obrigatoriamente, segundo as regras do negócio? Se a resposta for "sim", permanece a inclusão; sendo "não", substituímos a inclusão por extensão no sentido contrário (com o relacionamento de extensão apontando de `B` para `A`). A Figura 3.8 ilustra.

A abordagem feita sobre inclusões e extensões, embora não tão completa sob o ponto de vista da UML, mostra-se adequada para a maioria das situações de modelagem de sistemas.

Modelos de casos de uso só capturam as características estruturais dos processos (que elemento do modelo está relacionado a que outro(s)). Os detalhes das sequências em que as ações ocorrem durante o desenrolar dos casos de uso, dentre outros, só são tratados por meio das descrições dos casos de uso, que veremos no próximo capítulo.

3.6 Os Itens Anotacionais da UML

Na Figura 3.8 há um novo elemento da notação da UML: o "cartãozinho" com a ponta dobrada, que é usado para colocar comentários, qualquer texto que elucide algum aspecto importante do modelo ou um requisito não funcional, como, por exemplo, uma restrição. Esse é o chamado *item anotacional* da UML, que é associado ao diagrama ou opcionalmente a um elemento comentado por meio do segmento de reta tracejado.

Qualquer elemento do modelo pode ser associado a um comentário: um ator, um caso de uso, uma associação, uma classe, um objeto, uma atividade, ou seja, itens anotacionais podem aparecer em qualquer diagrama da UML. Embora comentários não estejam associados exclusivamente a casos de uso, como nós veremos muitos deles nas aulas que virão, decidi mencioná-los neste ponto do texto.

3.7 Resumo do Capítulo

Os diagramas de casos de uso são vistos como mecanismos bastante eficazes de captura de requisitos funcionais de um sistema, de forma a evitar a ambiguidade. A notação é simples o suficiente para que possamos validar, com os usuários, as funcionalidades do sistema e quem as utilizará.

Utilizando diagramas de casos de uso de negócio, podemos especificar quais são os processos de negócio e quais são os participantes desses processos. Em seguida, elaborando um diagrama de casos de uso de sistema, podemos especificar as funcionalidades que comporão o sistema que automatizará partes dos processos de negócio, bem como quem serão seus usuários, categorizados conforme suas interações com o sistema. Esses são, respectivamente, os casos de uso do sistema e seus atores.

Atores e casos de uso se relacionam. Esses relacionamentos são representados nos diagramas como associações, especializações-generalizações, inclusões e extensões.

No próximo capítulo prosseguiremos com o estudo dos casos de uso, tratando das especificações (descrições textuais) deles, citando erros frequentes de modelagem e dando dicas para a elaboração das especificações e mais dicas para elaboração dos diagramas.

3.8 Exercícios Propostos

1. Identifique os atores de casos de uso de sistema para cada uma das situações a seguir. Considere que as situações dizem respeito à especificação de três sistemas totalmente distintos entre si.

    a) ... o atendente abre uma nova OS (ordem de serviço)...

    b) ... o atendente abre uma nova OS e entrega uma cópia do relatório de abertura ao cliente que se encontra no balcão...

c) ...o atendente abre uma nova OS. Ao final do processo de abertura da OS o supervisor é informado por e-mail...

2. Identifique os casos de uso de sistema para cada uma das situações a seguir. Considere que as situações dizem respeito à especificação de três sistemas totalmente distintos entre si.

   a) O atendente informa a conclusão da OS...

   b) (em um sistema de segurança computadorizado) ...O sensor de presença do sistema de segurança aciona o alarme que pode ser desligado pelo supervisor de segurança...

3. Desenvolva os diagramas de casos de uso de sistema para as situações a seguir. Imagine que as situações são trechos de especificações de sistemas distintos.

   a) O atendente informa ao sistema a conclusão das OS cujos dados são, então, passados ao Sistema de Contas a Receber (SCR), que efetuará a cobrança...

   b) ...o atendente informa ao sistema a conclusão das OS. Uma cópia impressa do relatório de conclusão segue junto com o equipamento para o cliente e outra cópia vai para o setor de cobrança...

   c) ...o atendente abre uma nova OS, informando os dados do cliente e do equipamento...

   d) ...o atendente abre uma nova OS. Durante esse processo, o sistema solicita a definição dos campos de um formulário de cadastro de clientes. Esse mesmo formulário pode ser apresentado ao supervisor, para eventual alteração cadastral...

   e) ...o atendente abre uma nova OS e, caso o cliente não esteja cadastrado, essa é a hora de fazê-lo. O atendente ou o supervisor podem, a qualquer momento, cadastrar novos clientes sem que estes solicitem qualquer serviço...

   f) ...clientes do laboratório podem se cadastrar via Internet. O cadastro também pode ser feito na chegada do cliente, pela recepcionista, na abertura de uma lista de exames.

   g) Às sextas-feiras, às 18h, o expediente para o público é encerrado e às 18h30min o sistema automaticamente imprime a relação de inadimplentes...

   h) ...o chefe do suporte é informado pela rotina de autenticação do sistema, via "torpedo", de qualquer pedido de autenticação feito pelos usuários cadastrados na lista negra de usuários.

As soluções encontram-se a partir da Página 202.

CAPÍTULO 4

DIAGRAMAS DE CASOS DE USO: DESCRIÇÕES, DICAS E ERROS COMUNS DE MODELAGEM

> A writer is somebody for whom writing is more difficult than it is for other people.
>
> Thomas Mann

Vimos no Capítulo 3 que os diagramas de casos de uso são atemporais, ou seja, não capturam as sequências da interação com os usuários nem as das ações necessárias para a realização das funções do sistema. A especificação das ações (e em que ordem elas são realizadas) fica por conta de uma descrição textual, em formato padronizado pela organização ou para o projeto. A essa atividade damos os nomes de especificação ou descrição dos casos de uso.

As descrições dos casos de uso complementam os diagramas, ajudando a validar a interação sistema-usuário com o cliente e a ordem de execução das ações dos usuários e do sistema. São também usadas como artefatos de entrada para o processo de especificação dos testes funcionais, aqueles que atestam que o sistema faz o que o cliente pediu.

Diagramas de casos de uso complementados por suas descrições são, portanto,

artefatos de grande importância para as equipes de projeto e construção de sistemas.

Neste capítulo trataremos das descrições dos casos de uso e faremos o "fechamento" do assunto *Casos de Uso*, relacionando algumas dicas e apresentando os erros mais comumente encontrados na modelagem de casos de uso, incluindo em suas descrições.

4.1 Padrão de Descrições nas Organizações

Vimos que os diagramas de casos de uso não capturam todas as características das funções que o sistema precisará executar; por exemplo, por não contemplar a dimensão temporal, não há como especificar sequências de execução das ações dos usuários ou do sistema. Pelos diagramas também não podemos inferir como atores e sistema interagem para a realização das funções do sistema; sequer podemos inferir se todos os atores relacionados a um caso de uso no diagrama precisam atuar colaborativamente ou se cada um pode, isolada e independentemente, executar todo o processo. Esses detalhes só são capturados quando descrevemos os casos de uso.

As descrições dos casos de uso, também chamadas de *especificações*, não são padronizadas pela UML. Dessa forma, as organizações definem seus próprios padrões usualmente baseados em padrões disponíveis na Internet ou em livros. Elas criam e instituem templates do MS-Word para as descrições, segundo uma estrutura complexa e contendo muitas informações e referências cruzadas.

Essas informações são muitas vezes repetidas e/ou não tão relevantes para o propósito da modelagem. Minha crítica a esse respeito é que informação demais torna a especificação dos casos de uso uma atividade enfadonha e potencialmente geradora de inconsistências. Essas organizações esquecem que a completude e a concisão podem andar juntas.

Outro aspecto importante a ser observado é a dependência da especificação produzida com respeito à ferramenta que a mantém (que a visualiza, altera, elimina e imprime a descrição), seja ela gratuita ou paga. Documentos em formato proprietário estabelecem um vínculo de longo prazo com a ferramenta e com a empresa que a produziu, além de estarem sujeitos a corrompimento. O ideal é armazenar a documentação em formato textual aberto (por exemplo, DocBook), visualizável por qualquer editor de textos e intercambiável entre ferramentas de fabricantes e de versões diferentes.

4.2 Descrições Abreviadas e Descrições Detalhadas

As descrições podem ser feitas de forma abreviada, em alto nível de abstração, ou detalhada. Quando estamos no início do processo de levantamento, é normal optar por fazer uma aproximação "em largura", abordando o quanto antes o maior número de casos de uso do sistema, em detrimento dos detalhes de cada caso de uso. Nessa situação, elaboramos as descrições abreviadas. Nas demais etapas do levantamento fazemos o refinamento dos casos de uso, quando elaboramos as descrições detalhadas. Se determinada função do sistema estiver associada a algum tipo de risco maior, é recomendado detalhar o quanto antes a descrição dessa função.

> **ATENÇÃO:** O nível de detalhamento da especificação de um caso de uso deve depender do risco envolvido na execução da função do sistema.

Independentemente do padrão adotado, a descrição em alto nível deve conter o nome do caso de uso, a relação dos atores e, em poucas linhas, a descrição do processo adotado.

É usual que descrições em alto nível contenham, também:

- *pré-condições*, ou seja, tudo que precisa ser verdade para que o caso de uso inicie. Um exemplo de pré-condição é quando, para a execução de um caso de uso, é necessário que seus usuários estejam autenticados no sistema. O mais conciso, nesse caso, é colocar como pré-condição a informação "Usuário autenticado no sistema", ao invés de apontar com o relacionamento de inclusão o caso de uso *Autenticar Usuário* de todos casos de uso no diagrama que demanda autenticação;
- *pós-condições*, o seja, o que se torna verdade quando o caso de uso termina. Como isso usualmente depende de como o caso de uso termina, eu evito usar pós-condições ou as restrinjo ao final que tipicamente acontece. Acho importante deixar isso bem claro ao usar-se pós-condições;
- *garantias*. Essas, diferentemente das pós-condições, são o que se torna verdade quando o caso de uso termina, independentemente da forma como ele termina.

As descrições abreviadas são aproveitadas nas descrições detalhadas, pois estas refinam aquelas.

A descrição detalhada deve conter, além do que já foi relacionado, as seguintes informações:

- os passos que compõem o curso típico;

Tabela 4.1: Exemplo de tabela de regras de negócio.

Tabela de Regras de Negócio	
Regra de Negócio	***Descrição***
RN01	Todo dependente estará associado a um único funcionário.
RN02	Dependentes de funcionários devem ter idade inferior a 24 anos.
...	...

- os passos que compõem os cursos alternativos;

- a relação de regras de negócio que devem ser observadas. Há situações em que determinadas informações fornecidas pelos usuários ou produzidas pelo sistema precisam estar de acordo com uma ou mais regras do negócio. Nesses casos, o que se costuma fazer para que as descrições fiquem concisas é referenciar a regra ou regras de negócio onde elas precisam ser observadas.

Regras de negócio (RNs) são usualmente numeradas para que possam ser referenciadas mais facilmente nas descrições dos casos de uso. Usualmente, as regras compõem uma tabela de regras ou um capítulo à parte na documentação. Assim sendo, a tabela de RNs precisa ser colocada em um ponto específico da documentação; no capítulo (ou item) *Regras de Negócio*, por exemplo. A Tabela 4.1 ilustra.

Para ilustrar como referências a regras de negócio podem simplificar a descrição de casos de uso, imagine, por exemplo, uma situação em que os dados fornecidos por um usuário do sistema dizem respeito ao dependente de um funcionário que, por regra do negócio, não pode ter mais do que vinte e quatro anos. Poderíamos ter algo do tipo (mostrando apenas um trecho da descrição):

- ...Sistema exibe formulário para fornecimento dos dados do dependente do funcionário;

- Usuário preenche os campos com os dados do dependente;

- Sistema verifica idade do dependente com respeito à RN02...

A RN02, no caso, refere-se à regra de negócio de número dois (ver Tabela 4.1).

4.3 Especificando o Curso Típico e os Cursos Alternativos

Cada caso de uso tem o seu *curso típico*, em que relacionamos os passos da interação usuário(s)-sistema que compõem a situação que típica ou idealmente acontece. É comum relacionarmos o curso típico à "situação dos sonhos". Por exemplo, em um sistema de registro de compras de um supermercado, a situação típica é o código de barras poder ser lido pelo leitor ótico, ou seja, muito eventualmente o/a caixa precisa entrar com o código pelo teclado.

Quando não conhecemos o que tipicamente acontece, por estarmos lidando com uma situação nova, podemos considerar como curso típico os passos que comporão a situação que deverá, idealmente, acontecer.

Um caso de uso também pode possuir *cursos alternativos*, em que descrevemos as possíveis variações – em geral muitas – de execução do caso de uso. Situações alternativas são comumente – e erradamente – associadas a situações ruins. Na realidade, nem todas as situações alternativas são situações ruins, pois também podem estar associadas a escolhas feitas pelos usuários.

A descrição do curso típico e dos cursos alternativos é feita de forma não procedimental, em linguagem coloquial e usando o jargão do negócio, pois a especificação é normalmente usada para validar os casos de uso junto aos especialistas do negócio e demais envolvidos no processo.

> **ATENÇÃO:** Na descrição de um caso de uso se deve informar *o que é feito* pelo sistema em cada ação, sem haver a preocupação de relatar como a ação é realizada. Por não estarmos interessados, no momento da descrevermos o casos de uso, em como as ações são realizadas, referimo-nos à descrição como sendo *não procedimental*.

Considerando a necessidade de se ter, numa especificação de caso de uso, as suas informações básicas (nome, pequena descrição, relação de atores, pré e pós-condições, dentre outras), e sendo o conjunto de passos para a realização de um processo dividido em duas partes (o curso típico e o conjunto de cursos alternativos), os formulários usados nas organizações para descrição dos casos de uso são normalmente organizados conforme a Figura 4.1. O número/nome do caso de uso, a relação de atores, a descrição abreviada e, possivelmente, as pré e pós-condições ficam no cabeçalho.

As práticas comumente adotadas para a descrição dos casos de uso recomendam:

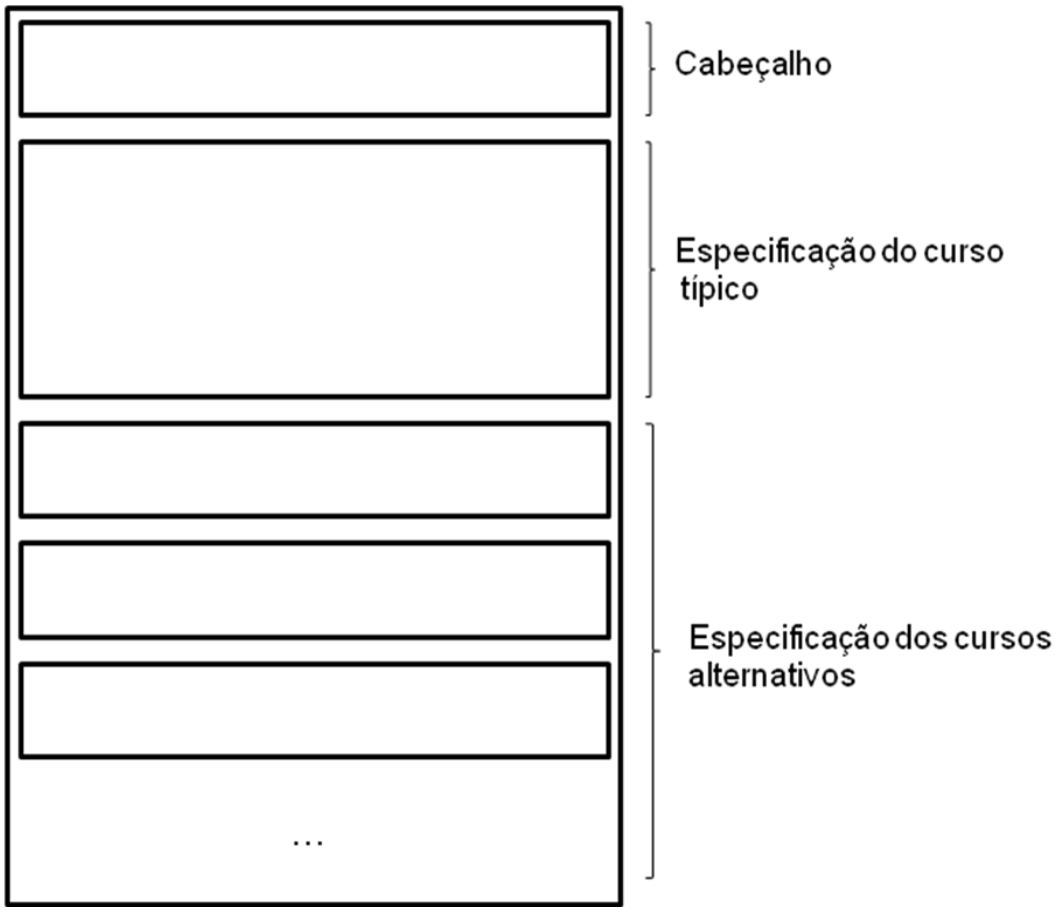


Figura 4.1: Leiaute típico do formulário de descrição de casos de uso.

- Tratar as alternativas mais frequentes primeiro e as menos frequentes em cursos alternativos subsequentes, ou seja, sempre procurando descrever antes o comportamento mais típico (daí o nome de curso típico para o primeiro bloco de passos) e depois a(s) variação(ões) mais típica(s) desse comportamento. Essa regra deve ser aplicada recursivamente, ou seja, se existe um comportamento mais frequente dentro de um comportamento alternativo, tratamos o mais frequente primeiro. No exemplo de descrição da Tabela 4.2, informamos no passo 1 do curso típico que o sistema determina que o usuário cuja senha será trocada é o próprio operador; isso porque essa alternativa é a ideal ou a mais frequente. Note que a possibilidade de ele não ser o próprio operador é tratada no passo 1 do curso alternativo 1.

- Evitar o uso de expressões do tipo "Se... então...", ou seja, devemos informar que ocorre a situação mais típica e, nos cursos alternativos, tratamos a(s) outra(s) possibilidade(s). Por exemplo, se um participante do processo pode responder "sim" ou "não" a uma pergunta do sistema e se, por exemplo, ele responde "sim" mais frequentemente a tal pergunta, assumir inicialmente que ele responderá "sim" e, em um curso alternativo seguinte, assumir que ele responderá "não".

- Usar, na especificação de repetições (*loops*) em um caso de uso, blocos "Para cada..." ou "Para todos...". Veja os exemplos no passo 3 do curso típico (CT) e no passo 1 do curso alternativo (CA) 1 da Tabela **??**.

- Numerar os passos do curso típico e de cada curso alternativo, iniciando de 1 (um) para que possam ser facilmente referenciados em outras partes da especificação. É mais usual numerar os passos dos cursos alternativos sempre iniciando de 1 (um), conforme os exemplos das Tabelas 4.3 e **??**. Outra maneira é prefixando a numeração com o número do curso alternativo (exemplos: 1.1 e 1.1.1, para passo 1 do curso alternativo 1 e passo 1 do curso alternativo 1.1, respectivamente). Eu prefiro a primeira maneira, por ser a mais simples.

- Referenciar um outro caso de uso quando ocorre uma inclusão ou uma extensão, explicitando a "chamada" no ponto da descrição do caso de uso que chama. Normalmente se usa a expressão "Executar caso de uso *tal*" no passo onde essa chamada deve ocorrer. No caso de uma inclusão, o caso de uso que chama é o de onde parte o relacionamento de inclusão; no caso de uma extensão, a chamada é feita no caso de uso aonde chega o relacionamento de extensão. Uma regra básica é a de que inclusões são especificadas como chamadas nas descrições do curso típico e extensões como chamadas em um dos cursos alternativos.

A título de ilustração do que acabamos de apresentar, veja nas Tabelas 4.2 e 4.3 (a Tabela 4.3 é uma continuação da Tabela 4.2) um exemplo de descrição de caso de uso de troca de senha de acesso em um sistema de monitoramento remoto e, nas Tabelas 4.4, um exemplo de descrição de um caso de uso de acionamento de relés em um sistema de automação, dando uma sugestão de como tratamos repetições e chamadas a outros casos de uso.

Note que, no exemplo da Tabela 4.3, mencionamos um *curso de exceção*. Esses cursos são cursos alternativos normalmente associados a possíveis falhas no sistema. Entendemos que esses cursos só precisam ser mencionados se suas ocorrências acarretarem impacto apreciável, seja no negócio sendo automatizado, seja no restante da execução do sistema. Por exemplo, se um erro do sistema demandar a execução de um outro caso de uso ou uma forma diferente de interação com o usuário, esse erro precisará ser tratado como um curso de exceção. Costumo exemplificar esse impacto citando o exemplo do fim da fita de impressão de um sistema de caixa de

Tabela 4.2: Exemplo de descrição de caso de uso de troca de senha de acesso (cabeçalho e curso típico).

Caso de Uso:	Trocar senha de acesso
Propósito:	Para o usuário trocar sua senha de acesso ao sistema.
Descrição geral:	O sistema atualiza os arquivos de senha com a nova senha do usuário. Este caso de uso é incluído pelo caso de uso “Alterar Senha de Acesso – Outro Usuário”, para a troca de senhas de outro usuário pelo Administrador de Usuários.
Atores:	Ator do caso de uso chamador (“Alterar Senha de Acesso – Outro Usuário”): • o próprio usuário para a troca de sua própria senha, ou • o Administrador de Usuários, na troca da senha de outro usuário do sistema.
Iniciador:	-
Pré-condições:	Operador autenticado no sistema. Operador habilitado a executar essa funcionalidade.
Pós-condições:	Caso sucesso: usuário tem sua senha trocada no arquivo de senha para acesso ao sistema e no arquivo de senha de cada diretório das câmeras às quais tem acesso.
Curso típico (CT)	
1. Sistema determina que o usuário cuja senha será trocada é o próprio operador. 2. Sistema obtém o *username* e o nome completo do usuário cuja senha está sendo trocada. 3. Sistema exibe formulário com os dados obtidos, um campo para a informação da senha atual, um campo para o fornecimento da nova senha e um campo para uma cópia da nova senha. Exibe as teclas “Alterar Senha” e “Sair”. 4. Operador informa senha atual. 5. Operador informa a senha nova. 6. Operador informa novamente a nova senha. 7. Operador pressiona o botão “Alterar Senha”. 8. Sistema verifica que a senha atual é válida. 9. Sistema verifica que as duas cópias da nova senha são idênticas. 10. Sistema obtém do perfil do usuário as câmeras às quais tem acesso. 11. Sistema criptografa senha. 12. Sistema armazena a nova senha criptografada em todos os diretórios correspondentes às câmeras às quais tem acesso, sobrescrevendo a senha anterior. 13. Sistema armazena a nova senha criptografada no arquivo de senhas para acesso às suas funcionalidades, sobrescrevendo a senha anterior. 14. Sistema informa sucesso na operação de troca de senha e exibe tecla “Sair” para o operador. ** Fim do Caso de Uso **	

Tabela 4.3: Exemplo de descrição de caso de uso de troca de senha de acesso (cursos alternativos e de exceção).

Cursos alternativos (CA)
CA 1: Passo 1 do CT - Sistema determina que o usuário cuja senha será trocada não é o próprio operador
1. Sistema obtém o *username* e o nome completo do usuário cuja senha está sendo trocada. 2. Sistema exibe formulário com os dados obtidos, um campo para o fornecimento da nova senha e um campo para uma cópia da nova senha. Exibe as teclas “Alterar Senha” e “Sair”. 3. Operador informa a senha nova. 4. Operador informa novamente a senha nova. 5. Operador pressiona o botão “Alterar Senha”. 6. Volta ao passo 9 do CT.
CA 1.1: Passo 5 do CA 1 - Operador pressiona o botão “Sair”.
** Fim do Caso de Uso **
CA 2: Passo 4 em diante do CT – Usuário pressiona o botão “Sair”
** Fim do Caso de Uso **
CA 3: Passo 8 do CT – Sistema verifica que senha atual não é válida
1. Sistema informa que senha atual não é válida e exibe as teclas “Voltar”. 2. Usuário pressiona a tecla “Voltar”. 3. Volta ao passo 1 do curso típico.
CA 4: Passo 9 do CT – As duas cópias da nova senha não são idênticas
1. Sistema informa que as duas cópias da nova senha não são idênticas e exibe as teclas “Voltar” e “Sair”. 2. Usuário pressiona o botão “Voltar”. 3. Volta ao passo 1 do curso típico.
CA 4.1: Passo 2 do CA 4 – Usuário pressiona a tecla “Sair”
** Fim do Caso de Uso **
Cursos de exceção (CE)
CE 1: Passo 12 ou 13 do CT – Não foi possível trocar a senha em um ou mais arquivos de senhas
1. Sistema desfaz as trocas de senhas já feitas. 2. Sistema registra erro no arquivo de log. 3. Sistema gera uma ocorrência para o Administrador. 4. Sistema informa insucesso na troca de senhas e exibe a tecla “Voltar”. 5. Usuário pressiona a tecla “Voltar”. 6. Volta ao passo 1 do CT.

Tabela 4.4: Exemplo de descrição de casos de uso ilustrando repetições e referência a outro caso de uso.

Caso de Uso:	Relés e seus estados
Propósito:	Produzir uma relação dos relés e seus estados (se ligados ou desligados).
Descrição Geral:	O supervisor de segurança visualiza a relação de números, nomes e estados dos relés, podendo ativá-los ou desativá-los, caso esteja habilitado para tal.
Atores:	Supervisor de segurança (usuário).
Iniciador:	Supervisor de segurança.
Pré-condições:	Usuário autenticado no sistema. Usuário habilitado a executar essa funcionalidade.
Pós-condições:	Caso sucesso: Relação de relés (números e nomes) e seus estados.
Curso típico (CT)	
1. Sistema obtém dos parâmetros o número de relés e seus nomes. 2. Sistema verifica que usuário está habilitado para alterar os estados dos relés. 3. Para cada relé, o sistema exibe a. o número do relé, b. o nome do relé, c. dois *radio buttons* (*ativados*), definindo-os com o estado correspondente ao estado corrente do relé. 4. Sistema exibe as teclas “Sair” e “Ligar/Desligar relés”. 5. Usuário define novos estados dos relés. 6. Usuário pressiona a tecla “Ligar/Desligar relés”. 7. Executar caso de uso “Acionar relés”. 8. Volta ao passo 1 do CT.	
Cursos alternativos (CA)	
CA 1: Passo 2 do CT – Usuário não está habilitado para alterar os estados dos relés	
1. Para cada relé, o sistema exibe a. o número do relé, b. o nome do relé, c. dois *radio buttons* (*não ativados*), definindo-os com o estado correspondente ao estado corrente do relé. 2. Sistema exibe a tecla “Sair”. 3. Usuário pressiona “Sair”. ** Fim do Caso de Uso **	
CA 2: Passo 6 do CT – Usuário pressiona a tecla “Sair”	
** Fim do Caso de Uso **	
Cursos de exceção (CE)	
Não há	

supermercado, quando ela ocorre no meio de uma lista de compras. O que precisa ser feito? Reiniciar a impressão do primeiro item quando a fita for trocada? Imprimir uma observação na nova fita ou simplesmente continuar a imprimir como se nada tivesse acontecido? Quando esse evento ocorrer, o supervisor de vendas precisará executar alguma outra funcionalidade do sistema? Note também nas Tabelas 4.2 e 4.4 como restringimos a aplicação das pós-condições às situações de sucesso de execução dos casos de uso.

Curiosidade: Em determinadas situações não é tão trivial definir se um curso é alternativo ou de exceção. Nesses casos acho que não precisamos perder muito tempo tentando descobrir se um curso é alternativo ou é de exceção. Existe, no entanto, uma técnica que pode ser útil para você estabelecer essa diferença, se você está muito preocupado com ela: cursos alternativos normalmente são trilhados por opções que os atores fazem durante a interação com o sistema; cursos de exceção são as "opções" que o sistema faz.

Como já dissemos, a UML não defina um padrão de descrição de casos de uso e a forma de especificar casos de uso que foi apresentada acima se alinha com o que se observa na prática e com o que se encontra na bibliografia que trata desse assunto[1]. Nada impede, no entanto, que definamos um novo padrão dentro da organização ou para um sistema em particular.

Cabe ainda mencionar que casos de uso são bem especificados usando os diagramas de atividade (DAs) da UML, que veremos no Capítulo 8.

4.4 Erros Frequentes e Más Práticas de Modelagem

Apresentamos a seguir alguns erros e más práticas de modelagem.

Associação Entre Atores

A comunicação entre dois ou mais atores só é de interesse para o negócio quando é necessária para a realização colaborativa de alguma funcionalidade do sistema. É um tanto frequente encontrarmos diagramas contendo associações diretas entre dois atores em um diagrama numa tentativa de se especificar um caminho (que

[1] Sobre padrões de descrição de casos de uso, faço uma especial referência ao livro *Writing Effective Use Cases*, de Alistair Cockburn ([2]), e recomendo a pesquisa e leitura das inúmeras páginas que ele e outros autores mantêm na Internet.

eventualmente existe) de comunicação entre os atores. Se indivíduos trocam informação necessária para a realização colaborativa de algum caso de uso, esse processo deve ser modelado como um caso de uso, e a comunicação entre os atores deve ser feita não diretamente, mas sim por meio desse caso de uso. Não há, portanto, sentido em haver associação entre dois atores sem que haja um processo representado por um caso de uso entre eles.

Casos de Uso Muito Pequenos ou Muito Grandes

Outra prática incorreta é identificar-se casos de uso que representam passos individuais, operações ou ações dentro de um processo. Casos de uso representam funções do sistema e, tipicamente, envolvem um número significativo de passos (costumo dizer que casos de uso são mais para "gordinhos"). Além dos casos de uso que representam funcionalidades disponíveis para os usuários do sistema, novos casos de uso podem ser criados nas seguintes situações:

1. Quando existe um comportamento comum a dois ou mais casos de uso, contanto que o número de passos comuns justifique essa fatoração. O novo caso de uso passa a constar do diagrama, sendo incluído pelos casos de uso originais;
2. Extraindo comportamentos complexos (com números significativos de passos na especificação), obrigatórios ou variantes de dentro de casos de uso. Assim, os novos casos de uso aparecerão no diagrama, sendo incluídos pelos casos de uso originais ou os estendendo.

A situação 2 está relacionada a casos de uso muito grandes, correspondendo a descrições muito longas, o que considero uma prática "abominável". Devemos lembrar que as descrições servem, em um primeiro momento, para validar com os usuários o nosso entendimento a respeito de suas necessidades. Se a descrição é longa demais, o usuário "dorme" ao lê-la, fica com preguiça de ler, valida de qualquer maneira... E isso não é bom!

Em decorrência disso, procure evitar o uso de casos de uso do tipo "manter", que tratam da inclusão, exclusão, alteração e pesquisa de informações em um cadastro. Esses casos de uso são conhecidos como casos de uso "CRUD", de *Create*, *Read*, *Update* e *Delete*. Se tratarmos essas quatro, diríamos, subfuncionalidades em um único caso de uso, provavelmente produziremos uma descrição muito grande, principalmente porque as regras de negócio e de integridade aplicáveis são normalmente muito diferentes para cada uma delas. Por exemplo:

- só se pode incluir um item em um cadastro se ele ainda não consta do cadastro;

- só se pode excluir um item do cadastro desde que ele exista e não seja referenciado por um item de outro cadastro;

- só se pode detalhar um item em uma consulta se ele consta do cadastro;

- só se pode atualizar um item se ele consta do cadastro e não estiver sendo referenciado por um item de outro cadastro.

Sempre considere, portanto, a possibilidade de dividir esses casos de uso em quatro, usando casos de uso "manter" unicamente em situações de cadastros bem simples. Em caso de dúvida, a melhor métrica é o tamanho da descrição: a meu ver, mais do que cinco páginas de descrição já é muito.

Atores "Voadores" e Atores "Gordinhos"

Atores não "voam" no modelo. Cada ator está associado a, pelo menos, um caso de uso do modelo. Se isso não ocorre para algum ator, remova-o do modelo.

Um determinado ator tipicamente não interage com o sistema de muitas e distintas formas. Se a descrição do papel que interpreta é longa e compreende atividades independentes entre si, quase certamente há mais de um papel sendo descrito. Nesse caso, devemos separar os atores, um para cada papel.

Complexidade Visual dos Modelos

Modelos visualmente complexos dificultam seu entendimento. O mesmo acontece com modelos contendo elementos e textos diminutos. Quando o número de elementos do modelo é tal que somos obrigados a diminuir a escala para que ele caiba em uma página da documentação impressa, é hora de pensarmos em dividir o modelo em partes. Existe uma regra empírica que estabelece que o número ideal de elementos de um modelo em uma mesma página deve variar de 5 a 9 ("regra do 7 +/- 2" – sete mais ou menos dois) para uma boa compreensão. Na prática, observamos que, contrariando a regra, modelos de casos de uso com até quinze casos de uso ainda são bem compreensíveis se não há muitos cruzamentos de associações. No caso de isso não ser possível em função da complexidade do negócio, somos obrigados a usar *pacotes* (agrupamentos) de atores e de casos de uso. Pacotes serão tratados no Capítulo 11.

Fim do Caso de Uso

O indicativo de "fim de caso de uso" (veja as descrições das Tabelas 4.2, 4.3 e 4.4) devem significar o encerramento do processo e o término do que virá a ser o programa que implementará o caso de uso, e não o fim da descrição de um curso típico ou alternativo.

Consistência Diagrama-Descrições

Embora diagramas de casos de uso e as descrições dos casos de uso sejam partes distintas do modelo, eles possuem uma coerência mútua que deve ser observada e preservada. Essa coerência consiste de diversos aspectos. Dentre eles:

- todos os casos de uso desenhados nos diagramas precisam ter suas descrições feitas. Se existe uma descrição sem caso de uso ou vice-versa, é sinal de que algo está errado, no diagrama ou nas descrições;
- casos de uso que incluem ou são estendidos não possuem referência das respectivas inclusões nas especificações (como "Executar caso de uso *tal*" – veja exemplo no passo 7 do CT da Tabela 4.4). Se uma inclusão ou extensão não aparece em algum ponto da descrição do caso de uso que inclui, seja no curso típico, seja em um alternativo, algo está errado no diagrama ou na descrição;
- atores nos diagramas não podem "desaparecer" ou mudar de nome nas relações de atores nas especificações, ou seja, atores associados a casos de uso devem fazer parte da lista de atores dos respectivos casos de uso;
- atores associados a um caso de uso em um diagrama (e, portanto, também na relação de atores da descrição desse diagrama) devem ser mencionados de alguma forma, em algum ponto da descrição.

Esses e outros lembretes, dicas e orientações sobre casos de uso podem ser encontrados do artigo *How to avoid use-case pitfalls*, de Suzan Lilly ([9]), disponível gratuitamente na Internet (em `http://www.drdobbs.com/184414560`, por exemplo). Vale a pena conferir.

4.5 Resumo do Capítulo

A especificação ou descrição dos casos de uso define as ações necessárias à realização das funções do sistema e em que ordem elas são realizadas. As descrições dos casos de uso complementam os diagramas, ajudando a validar a interação sistema-usuário

com o cliente, além de serem artefatos importantes para o pessoal de projeto e para a equipe de formulação dos casos de testes funcionais.

As descrições dos casos de uso não são padronizadas pela UML. As organizações definem seus próprios padrões usualmente baseados em padrões disponíveis na Internet ou em livros.

As descrições podem ser feitas na forma abreviada ou na detalhada, dependendo do estágio no ciclo de desenvolvimento e dos riscos associados à execução do sistema. Elas especificam o nome, o propósito, a lista de atores, as pré e pós-condições, além dos fluxos típico (apenas um para um caso de uso), alternativos e de exceção (usualmente vários) da interação entre os usuários e o sistema, dentre outras informações.

Descrições são não procedurais, ou seja, especificam *o que* o caso de uso faz e não como ele faz.

4.6 Exercícios Propostos

1. Por que mencionamos que o caso de uso que chama outro é aquele de onde parte o relacionamento de inclusão ou o aonde chega o relacionamento de extensão?

2. Por que estabelecemos a regra básica de que inclusões são especificadas nas descrições do curso típico e extensões em um dos cursos alternativos?

3. Esboce o diagrama e descreva o caso `Registrar Compra` em um sistema para um supermercado hipotético, do qual participa o `Caixa`, registrando a compra, eventualmente o `Cliente`, quando o pagamento é feito por débito ou crédito no cartão e ele precisa informar a senha, além do `Supervisor de Vendas`, quando é necessário retirar um ou mais itens da lista compras ou reimprimi-la. Use sua vivência para estabelecer os passos que compõem a descrição, mas não se esqueça de considerar as situações em que:

    - tudo dá certo;
    - você não tem o dinheiro suficiente para pagar toda a compra, podendo perceber isso durante o registro ou ao final dele;
    - a fita de papel da máquina registradora acaba no meio da compra e o supervisor precisa intervir com seus "superpoderes" para comandar a reimpressão da lista desde o início;
    - você discorda do preço de um item que estava em oferta e pede ao caixa que retire o item da lista. Nesse caso, o supervisor também precisa intervir;

- o código de barras não pôde ser lido pela leitora ótica e o caixa o informa pelo teclado;
- o código do item não consta do cadastro;
- você paga em cartão com *chip* (no débito ou no crédito) ou em dinheiro, o que é bem menos frequente naquele supermercado.

As soluções encontram-se a partir da Página 205.

CAPÍTULO 5

DIAGRAMAS DE CLASSES: CONCEITOS, PERSPECTIVAS, ELEMENTOS BÁSICOS DA NOTAÇÃO E ASSOCIAÇÕES

> Call it a clan, call it a network, call it a tribe, call it a family:
> Whatever you call it, whoever you are, you need one.
>
> Jane Howard

Durante nossa conversa com os usuários, no processo de levantamento dos requisitos do sistema, além das funções e informações que o novo sistema precisa executar e produzir, são passadas informações que tratam de "coisas" ou conceitos com os quais os colaboradores em uma organização lidam no dia a dia. Por exemplo, os estoquistas lidam com peças, solicitações de reposição e pedidos de fornecimento; o pessoal do contas a pagar lida com faturas e notas fiscais; o pessoal de vendas, com pedidos e clientes; e o pessoal de manutenção de campo com ordem de serviço – OSs –, orçamento, visita etc. Se esses aspectos, ou *conceitos do negócio*, não se relacionassem entre si e se não tivessem outros conceitos embutidos, provavelmente bastaria que registrássemos os termos com seus significados em um dicionário ou glossário para que o projetista pudesse iniciar a concepção da solução.

Quase invariavelmente, no entanto, esses conceitos possuem muitos detalhes e se inter-relacionam de formas intrincadas e obedecendo a determinadas regras ou restrições, o que nos desaconselha elaborarmos uma simples descrição textual, por mais estruturada que seja.

Tomemos como exemplo a situação em que o nosso cliente especialista no negócio menciona que um pedido feito a um dos seus fornecedores deve ser associado a possivelmente mais de uma nota fiscal de entrega e que uma nota fiscal de entrega só pode conter itens referentes a um único pedido, e que pedidos devem conter as informações *tais e tais* e que devem estar associados aos clientes que os colocaram, e que notas fiscais devem ter as informações *tais e tais*, além, claro, dos detalhes (descrição, preço unitário, quantidade e preço total) dos itens que as compõem... ufa! Com três ou quatro conceitos apenas a nossa especificação já ficou complicada demais para uma descrição textual.

Note que as informações que são passadas não tratam de requisitos funcionais e de informação, somente, mas também de características diversas que precisamos capturar de alguma forma para que o sistema seja projetado e construído corretamente.

Neste capítulo iniciaremos o estudo dos diagramas de classes da UML, que vêm em nosso socorro para nos atender em situações em que necessitamos especificar a *estrutura da informação*, que é a visão estática do sistema, especificando as relações atemporais (que não variam com o tempo, daí a ideia de visão estática) entre os conceitos que compõem o domínio.

Os diagramas de classes proveem as bases de qualquer metodologia de análise e projeto de sistemas computacionais orientada a objetos. Não há, portanto, um sistema minimamente documentado para o qual não tenhamos desenvolvido um diagrama de classes.

5.1 Perspectivas em Diagramas de Classes

Os diagramas de classes em modelos de sistemas podem especificar as perspectivas conceitual, de especificação e de implementação.

Cada perspectiva representa o problema ou a solução com graus diferentes de abstração:

- um diagrama de classes conceitual contém apenas classes de conceito (daí o nome *conceitual*), dotando o modelo de alto grau de abstração, ou seja, onde os detalhes são esquecidos. Modelos conceituais especificam parte do problema a ser solucionado pelo sistema. Costumamos dizer que diagramas conceituais de

classes compõem os modelos de análise[1]. Como todos os elementos colocados no modelo conceitual correspondem a "coisas" do negócio, eles são, portanto, de conhecimento dos especialistas do negócio. Se algum conceito representado no modelo conceitual é estranho aos especialistas do negócio, muito provavelmente você está sendo *prematuramente físico*, ou seja, erradamente pensando em detalhes de projeto na fase de análise.

- no extremo oposto, na perspectiva de implementação, representamos todos os detalhes necessários para a implementação do sistema considerando todas as características das tecnologias escolhidas. Dizemos que, na perspectiva de implementação, estamos no nível de abstração zero, em que nada é esquecido. Diagramas de classes de implementação são bastante extensos e complexos por detalharem as minúcias da solução que os projetistas conceberam;

- a perspectiva de especificação se situa entre essas duas, ou seja, começa no instante em que adicionamos ao modelo conceitual completo a primeira classe ou detalhe da solução que o projetista está dando para o problema e termina quando obtemos o modelo de implementação. O nome *especificação* está associado à fase em que o projetista especifica a solução que está dando para o problema.

Neste texto nos manteremos no nível conceitual, fazendo apenas "incursões" no nível de especificação quando tratarmos de diagramas de sequência. Os diagramas de classes que elaboraremos terão o objetivo de representar os conceitos de negócio, seus relacionamentos e restrições (regras do negócio).

A Figura 5.1 ilustra os sentidos de diminuição do nível de abstração e de aumento do detalhamento, tipicamente conforme o tempo passa, enquanto caminhamos em direção à perspectiva de implementação.

Os diagramas de classes compõem-se de classes, dos relacionamentos entre elas e de restrições do negócio. Trataremos agora da notação gráfica da UML para diagramas de classes.

5.2 Classes Conceituais ou de Entidade

A classe é o elemento-chave em um diagrama de classes. No nível conceitual, uma classe representa um conceito do negócio. Assim, conforme ilustra a Figura 5.2,

[1]Modelos de análise são os modelos conceituais elaborados durante a fase de análise do sistema, quando nos preocupamos apenas em especificar com precisão o que o sistema fará, e não como fará. A informação de como o sistema fará (os detalhes da solução, portanto) fazem parte dos modelos que chamamos de modelos de especificação ou de projeto.

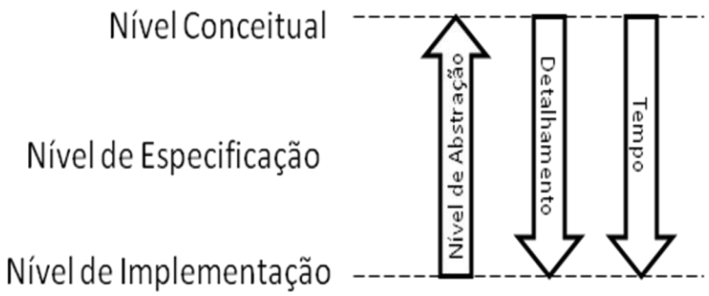


Figura 5.1: Níveis de abstração em uma especificação.

`Pedido`, `Cliente` etc. são conceitos que fazem parte de um contexto típico em uma empresa fictícia – à qual demos o nome de ZYX – que lida com pedidos feitos por sua clientela.

As classes conceituais, também chamadas de classes de entidades, são entidades das quais nos interessa ter suas propriedades armazenadas em um arquivo convencional (pastas suspensas e fichas), em um sistema manual, ou no banco de dados de sistema informatizado. Por essa razão, quando projetamos um sistema informatizado quase invariavelmente assinalamos no CASE[2] as classes conceituais como *persistentes*, significando que suas instâncias serão armazenadas em banco de dados ou outro tipo qualquer de arquivo em disco para uso futuro.

Os conceitos representados pelas classes conceituais são de pleno conhecimento dos participantes do negócio, ou seja, no exemplo da Figura 5.2, nosso cliente conhece bem os conceitos de `Pedido`, `Fornecedor`, `Pedido de Reposição de Estoque`, etc.

Classes conceituais também são chamadas de classes de entidades e de classes de negócio.

Nos diagramas de classes, as classes são representadas por retângulos com um ou mais compartimentos, dependendo do nível de detalhamento[3].

O nome da classe é colocado no primeiro compartimento em negrito e

[2]Algumas ferramentas CASE ajudam a elaborar os projetos físicos dos bancos de dados que usaremos no sistema. Com base nas classes persistentes e seus relacionamentos e na tecnologia a ser adotada (fabricante do SGBD e versão), esses CASEs geram automaticamente os comandos de criação do banco e de suas tabelas.

[3]Alguns CASEs permitem que escolhamos o nível de detalhamento, exibindo ou não compartimentos e as informações correspondentes em cada classe. Outros, a partir da informação de que se trata de um diagrama de análise, apresentam classes com dois compartimentos: o do nome e o dos atributos.

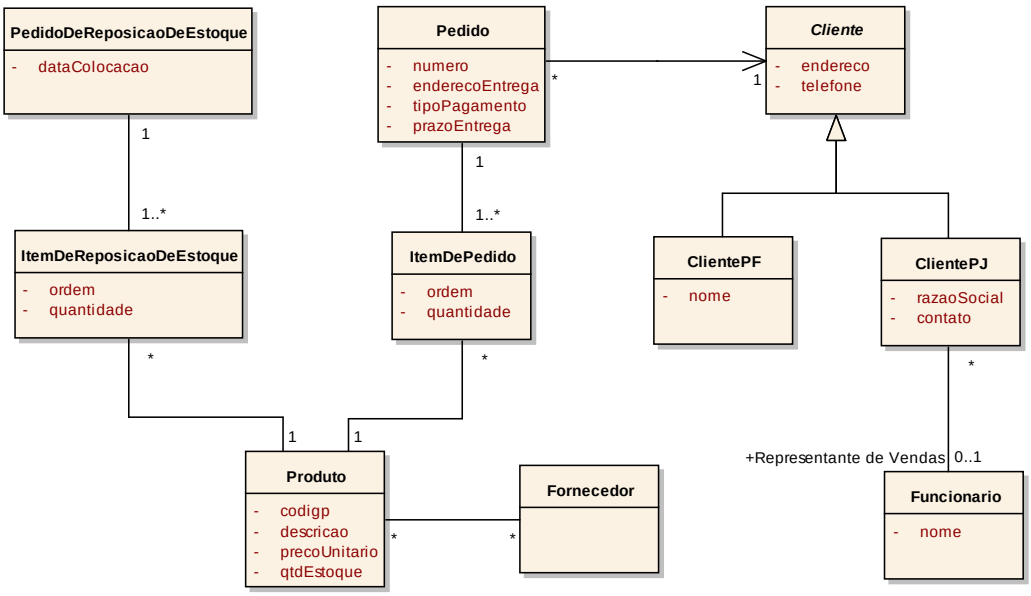


Figura 5.2: Diagrama de classes conceitual da empresa fictícia ZYX.

centralizado. Recomenda-se que os nomes sejam substantivos no singular ou expressões breves, preferencialmente com base no jargão usado no negócio. Os nomes são únicos em um *espaço de nomes* (*namespace*, em inglês), pois identificam univocamente as classes no modelo.

Um espaço de nomes é um local abstrato que fornece contexto para os itens colocados nele. Um espaço de nomes é um conjunto abstrato de coisas, um contêiner. Em um dado espaço de nomes, cada elemento nele contido precisa ter um identificador – um nome – que deve ser único nesse espaço. Identificadores podem ser repetidos em espaços de nomes distintos, entretanto, quando compostos com o respectivo espaço de nomes, se tornam únicos no domínio.

Uma dada classe pode ter mais de uma cópia em um mesmo (ou em outro) diagrama do modelo. A propósito, podemos, sim, ter mais de um diagrama de classes compondo o modelo de classes de um sistema. Isso, por sinal, é até bastante usual, especialmente em sistemas grandes.

Durante o desenvolvimento do modelo de classes, o analista deve ter preocupação com o nome que dará a cada classe; o nome, embora deva ser um substantivo ou uma expressão breve, deve transmitir bem o conceito que a classe

representa. Em modelos conceituais, o trabalho de dar nomes às classes é, de certa forma, facilitado, já que os nomes devem ser preferencialmente retirados do jargão do negócio.

Identificar classes é o primeiro passo para a elaboração do diagrama de classes.

Às vezes nos enganamos e esquecemos de identificar uma classe ou identificamos uma ou outra erradamente. Como os modelos conceituais são detalhados ao longo do projeto, passando por modelos de especificação até virarem modelos completos, de implementação, classes necessárias, mas não identificadas, invariavelmente o serão ao longo do projeto. Classes identificadas a mais invariavelmente não serão usadas e serão eliminadas do modelo... cedo ou tarde. Com isso, não há muito como errar na identificação de classes ao longo de um projeto completo.

Uma boa técnica para descobrirmos se determinada classe é ou não conceitual é perguntar sobre o conceito a ela relacionado ao cliente especialista do negócio que está sendo entrevistado. Se ele não souber responder a respeito, não conhece, nunca usou aquele termo no seu dia a dia, provavelmente a classe não deverá fazer parte do modelo conceitual.

Objetos são instâncias ou ocorrências das classes. Cada pedido da coleção de pedidos feitos à empresa ZYX, por exemplo, é uma instância da classe `Pedido`. As classes que podemos instanciar, ou seja, das quais podemos solicitar a criação de objetos, são chamadas de classes concretas. A classe `Pedido` é, portanto, um exemplo de classe concreta. Outros exemplos de classes concretas na Figura 5.2 são `ItemDePedido`, `Produto`, `Funcionario`, `ClientePF` e `ClientePJ`.

5.3 Atributos das Classes

As informações a respeito dos conceitos (por exemplo, o endereço e o telefone do cliente na Figura 5.2) que gostaríamos de manter em um cadastro são chamadas de *atributos das classes*. A relação de atributos é colocada no segundo compartimento do retângulo da classe, justificada à esquerda. A necessidade de mantermos valores de atributos para as ocorrências de uma determinada classe justifica, como já mencionamos, a existência dessa classe, ou seja, se desejamos armazenar as informações sobre uma categoria de coisas em um negócio, provavelmente essa categoria se tornará uma classe no modelo de classes do sistema.

Os atributos que desejamos relacionar no modelo conceitual são aqueles que os especialistas do negócio mencionam. Não é certo relacionarmos atributos nessa fase além daqueles que os especialistas julgam necessários. Podemos, claro, lembrá-los de alguns atributos que são típicos, mas eles é que dão a palavra final sobre a necessidade

ou não. Também não é certo nos preocuparmos com detalhes, como os tipos dos atributos, se cadeias de caracteres, se numéricos e com qual precisão numérica etc. No modelo da Figura 5.2, a classe Fornecedor não possui atributos relacionados, o que sugere que ainda não terminamos o modelo conceitual.

Em certas situações, no entanto, não conseguimos identificar facilmente um atributo para uma classe, mas temos a intuição de que aquele conceito é importante, inclusive porque se relaciona com outro(s) conceito(s) importante(s) (veja a observação feita no final da resposta para Exercício 1, na Página 212). O fato de um conceito ter relacionamento com outro também justifica sua colocação no modelo. A razão para isso é que os relacionamentos que uma classe possui com outras podem ser entendidos como atributos dessa classe.

Nomes de atributos devem ser expressões breves, sem espaços em branco. Nomes de atributos, por hábito da comunidade de desenvolvedores, usam o estilo *CamelCase*[4], começando com letras minúsculas. É usual, também, mesmo no nível conceitual, suprimir cedilhas, acentos etc., desde que os nomes não fiquem muito confusos.

Os nomes dos atributos são suficientes nos modelos conceituais. Mais adiante, no ciclo de desenvolvimento do sistema, mais especificamente na fase de projeto, devemos completar os nomes dos atributos com outros detalhes. Além dos nomes, a notação UML um pouco mais completa para rótulos de atributos (atributos são referenciados pela UML como propriedades) é:

$$[visibilidade][/]nome : tipo[multiplicidade][= valordefault]$$

onde os valores entre "[" e "]" nem sempre ocorrem e:

- *visibilidade* é o caractere "-" (para privado), "+" (para público) ou "#" (para protegido), que indica se o atributo é visível ou não de outros objetos. Atributos privados só podem ser acessados (consultados diretamente e/ou modificados) pelos objetos que os contêm. Atributos públicos podem ser acessados por outros objetos e atributos protegidos são acessados pelos objetos que os contêm ou por objetos instanciados de classes especializadas. A visibilidade deve ser omitida no modelo conceitual;

[4] *CamelCase* é a denominação em inglês para a convenção de se escrever palavras compostas ou mesmo frases, onde cada palavra é iniciada com maiúsculas e sem espaços entre elas. Essa denominação é associada às corcovas de um camelo. A variação mais comum é grafarmos a primeira palavra da expressão em minúsculas e as demais palavras da expressão iniciando em maiúsculas. Exemplos: `dataDeNascimento`, `numeroDeContato` etc.

- a "/" antes do nome indica que o atributo é derivado, ou seja, seu valor pode ser determinado por um algoritmo a partir de outro(s). Por exemplo, se tivermos os atributos `idade` e `dataDeNascimento`, o atributo `idade` deve ser precedido da "/", já que a idade de um indivíduo pode ser determinada a partir da sua data de nascimento;

- *tipo* define o tipo de dados: inteiro, real, cadeia de caracteres, data etc.;

- *multiplicidade* indica as possíveis cardinalidades[5] para a ocorrência do atributo. Se a multiplicidade é omitida, significa que ela é exatamente 1. Veremos multiplicidades em maiores detalhes um pouco mais adiante;

- o *valor default* é o valor que o atributo assume de início.

Exemplos de rótulos de atributos:

```
-dataDeNascimento :  Date;
-/idade :  int;
+nomeContato :  String[0..2];
```

O primeiro exemplo, o atributo `dataDeNascimento` é privado e do tipo `Date` (data). O segundo exemplo, `idade`, é privado, derivado (calculado) e do tipo inteiro. O terceiro exemplo, `nomeContato` é público, do tipo *string* (cadeia de caracteres) e pode ter nenhuma, uma ou duas ocorrências, ou seja, pode não haver nome de contato registrado ou um ou dois nomes de contato registrados.

A visibilidade à qual nos referimos anteriormente tem a ver com a ideia de *encapsulamento*, que recomenda deixarmos escondido o que não é preciso ser mostrado. Aumentamos a facilidade com que damos manutenção em um sistema (manutenibilidade), definindo como privado o maior número possível de atributos (e operações, como veremos adiante). Mesmo os atributos que precisam ser "vistos" por outros objetos devem ser definidos como privados e devem ser criadas operações públicas de acesso para leitura e escrita a eles. Por meio dessas operações garantimos acessos mais "policiados" aos atributos. O encapsulamento máximo é sugerido pelos CASEs que, por *default*, atribuem visibilidade privado a cada novo atributo que adicionamos em uma classe.

Se você está interessado em detalhes da especificação para rótulos de atributos de classes, consulte o documento de superestrutura da UML ([11]).

[5] *Cardinalidade*, na teoria dos conjuntos, é o número de elementos de um conjunto. *Multiplicidade* na UML denota as possíveis cardinalidades de um conjunto de elementos. Por exemplo, se um atributo tem multiplicidade [0..2] significa que ele pode ter nenhuma, uma ou duas ocorrências (na UML os ".." indicam intervalos). Se um atributo possui multiplicidade [0..1], significa que ele é opcional, pois pode ou não ocorrer.

5.4 Operações das Classes

O terceiro compartimento do retângulo da classe contém a lista de operações que os objetos da classe implementam para realizar suas responsabilidades. É praxe, no entanto, que esses compartimentos fiquem vazios no modelo de análise, pois as operações normalmente só começam a ser descobertas quando iniciamos o nível de especificação, ao elaborar os diagramas de sequência.

Para o propósito de nosso curso, a notação UML suficientemente completa para os rótulos de operações é:

$$[visibilidade]nome([listadeparâmetros])[: tipoderetorno]$$

onde os valores entre "[" e "]" nem sempre ocorrem e:

- *visibilidade* é o caractere "-" (privado), "+" (público) ou "#" (protegido) que indica que a operação é visível ou não de outros objetos, do mesmo jeito que com os atributos;
- o nome da operação também é formado segundo o padrão *CamelCase*;
- *tipo* define o tipo de retorno da operação: inteiro, real, cadeia de caracteres, data etc.;
- a lista de parâmetros é formada pelos parâmetros de entrada e saída separados por vírgulas, da seguinte forma: $[direção]nome : tipo$, onde *direção* (opcional) é "*in*" ou "*out*" ou "*inout*", significando parâmetro de entrada, de saída e de entrada e saída, respectivamente. O nome e o tipo são da mesma forma que nos atributos.

Os rótulos das operações são justificados à esquerda dentro do compartimento. Exemplos:

```
+getIdade() :  int;
+setDataDeNascimento (in dataNascimento :  Date);
```

No primeiro exemplo, a operação `getIdade` é pública e retorna o valor da idade, que é um valor inteiro. Não há parâmetros de entrada ou saída. No segundo exemplo, a operação `setDataDeNascimento` é pública e armazena o valor da data de nascimento fornecida por meio do parâmetro de entrada `dataNascimento`, que é do tipo *Date*. Essa operação não retorna qualquer valor.

Para nossos propósitos, a notação para rótulos de operações apresentada é completa o suficiente. A UML especifica muitos outros detalhes que podem vistos no documento de superestrutura ([11]).

5.5 Restrições e Responsabilidades

As classes podem conter outros compartimentos, conforme estabelece a UML. Um quarto compartimento pode ser usado, por exemplo, para acomodar restrições e responsabilidades de classes.

As responsabilidades de uma classe definem a utilidade dos objetos dessa classe em um sistema. Um sistema desenvolvido segundo o paradigma de orientação a objetos assume que os objetos colaboram para a realização dos objetivos desse sistema. O conjunto de coisas que cada objeto faz para colaborar compõe as responsabilidades desse objeto. O conjunto de todas as coisas que todos os objetos fazem para colaborar compõe as responsabilidades da classe desses objetos. Para realizar essas responsabilidades, os objetos executam as operações. Costumo fazer a seguinte analogia em sala de aula: quais são as responsabilidades de cada contador (os indivíduos, os objetos, instâncias da classe `Contador`, segundo nossa terminologia) em uma organização? Fechar o balanço no final do mês é uma. Então, para realizar essa responsabilidade, que operações todo contador precisa fazer? Somar, diminuir, verificar *voucher* de lançamentos contábeis, verificar lançamentos etc.

Embora, como dissemos, a UML faculte a representação de mais do que três compartimentos, eu não conheço uma ferramenta CASE sequer que ofereça suporte gráfico para mais de três compartimentos. Entretanto, em todos os CASEs com os quais já trabalhei (Rose, Together, Jude, Magic Draw e Enterprise Architect) há caixas de texto na seção *Propriedades* de cada classe onde essas e outras informações podem ficar registradas.

5.6 Relacionamentos Entre Classes

Os conceitos identificados em um dado contexto invariavelmente se ligam de alguma forma. Relacionamentos em diagramas de classes expressam essas ligações, que, por também fazerem parte do conhecimento a respeito do negócio, precisam ser capturadas e especificadas no modelo de classes. Por exemplo, as ligações entre os pedidos feitos pelos clientes (e, reciprocamente, os clientes que fizeram os pedidos) da Figura 5.2 são representadas por um dos tipos de relacionamentos possíveis entre classes previstos na UML: associações.

A associação é um dos tipos mais comuns de relacionamentos representados em diagramas de classes. Outros relacionamentos comuns são as generalizações-especializações, as agregações, as composições e as dependências funcionais. Trataremos cada um deles nas seções a seguir.

5.7 Associações Entre Classes

Associações são o tipo mais comum de relacionamentos entre classes em um diagrama de classes. No diagrama da Figura 5.2, está especificado, por exemplo, que um determinado cliente pode estar associado a qualquer número de pedidos (inclusive zero, ou seja, empresas ou pessoas físicas são consideradas clientes mesmo não tendo feito qualquer pedido), e um determinado pedido está associado a somente um cliente.

Por exemplo, segundo a mesma figura, o cliente `Luiz Antônio` pode estar associado ao pedido `225` e ao pedido `786`. Associações nos diagramas de classes especificam, portanto, as ligações existentes entre os objetos instanciados de cada uma das classes associadas no diagrama de classes.

> **ATENÇÃO:** Associações entre objetos são instâncias (ocorrências) das associações representadas nos modelos de classes.

As associações são representadas nos diagramas de classes por segmentos de retas, poligonais ou arcos que ligam as classes associadas. Uma associação é opcionalmente rotulada com o nome da associação, que deve ser colocado sempre que o significado da associação não é claro no diagrama.

A Figura 5.3 ilustra a situação em que um determinado professor pode estar associado de duas formas distintas (se combinadas, são três) a um determinado departamento de uma universidade: sendo o chefe do departamento, sendo o chefe do departamento e alocado como seu professor ou somente alocado como professor do departamento.

No caso da Figura 5.3, se não colocássemos os nomes das associações, não saberíamos o que significa cada associação. Em contrapartida, se o significado de uma associação é claro no contexto do negócio, como, por exemplo, a associação entre `Cliente` e **Pedido** da Figura 5.2 (cliente está associado ao(s) pedido(s) que coloca na empresa ZYX), podemos pensar em não colocar seu nome para não aumentar a complexidade visual do diagrama.

O nome da associação deve vir acompanhado do símbolo de sentido de leitura (só a ponta cheia de seta) e deve exprimir bem o significado da associação, sendo

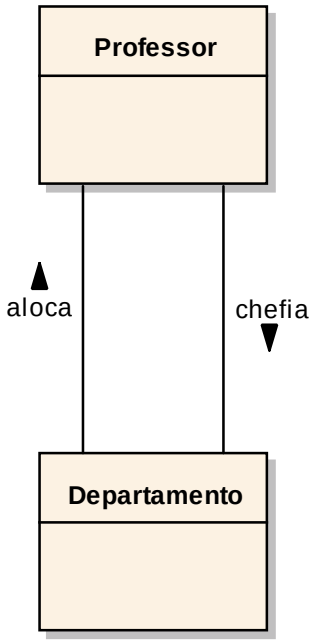


Figura 5.3: Nomes de associações e direções de leitura.

preferencialmente um verbo na voz ativa. A vantagem de usarmos verbos na voz ativa em nomes de associações é que também podemos ler o nome da associação no sentido contrário ao da seta, bastando mudar o verbo para a voz passiva (exemplos: "professor é alocado a departamento"; "departamento é chefiado por professor").

5.8 Papéis nas Associações

As pontas das associações (no pontos onde elas se encontram com as caixas das classes) chamam-se *papéis*, que também podem ser usados para dar nomes aos papéis que as classes representam nas associações. Quando não especificamos o rótulo do papel (que deve ficar bem junto do ponto onde a associação encontra a caixa da classe), este leva o nome da classe.

De volta à Figura 5.2, `Representante de Vendas` é o papel que `Funcionário` desempenha quando está associado a `ClientePJ`. Na Figura 5.4, `chefe` é o papel representado por `Professor` quando associado a `Departamento` por meio da associação de chefia. Repare que, neste caso, omitimos o nome da associação pelo fato de o rótulo do papel do professor já caracterizar bem que a associação se refere à

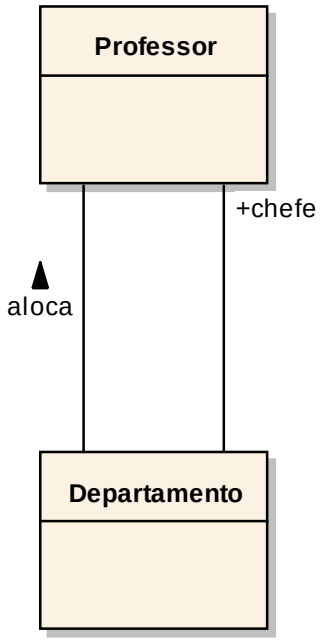


Figura 5.4: Rótulo de papel ajudando no significado da associação.

chefia do departamento.

Como dissemos, se não colocarmos o rótulo do papel, este leva o nome da classe. Sendo assim, na Figura 5.2 lemos: "um `ClientePJ` está associado a zero ou a um `Representante de Vendas`", ao invés de dizermos "um `ClientePJ` está associado a zero ou a um `Funcionário`", como faríamos se o rótulo do papel não tivesse sido especificado. Da mesma forma, na Figura 5.4 lemos: "`Departamento` é chefiado por `chefe`", ao invés de "`Departamento` é chefiado por `Professor`", caso o rótulo do papel não tivesse sido especificado.

5.9 Multiplicidades nas Associações

As pontas de uma associação entre classes devem especificar também as multiplicidades, que indicam quantas instâncias das classes podem participar da associação. Essa indicação é feita por meio dos valores máximo e mínimo. As multiplicidades podem ser:

- obrigatórias, quando especificadas por meio de um número natural diferente de

zero;

- opcionais, se "0..1";
- multivaloradas, se "*" ou "0..*" (as duas notações têm o mesmo significado).

Intervalos de multiplicidades podem ser especificados com os ".." (exemplo, "1..3", para 1, 2 ou 3, ou de 1 a 3). Se houver mais de um intervalo, eles são especificados entre vírgulas (exemplos: "1..3, 5..7", para 1, 2, 3, 5, 6 ou 7).

5.10 Navegabilidade nas Associações

As pontas das associações também podem conter o sinal de *navegabilidade* (ver seta aberta na ponta da associação entre `Pedido` e `Cliente` na Figura 5.2), que representa a responsabilidade que um objeto tem de localizar o(s) objeto(s) da outra classe com o(s) qual(quais) se associa. A navegabilidade representada na Figura 5.2 significa que objetos da classe `Pedido` "têm a responsabilidade" de localizar os clientes a eles relacionados que, em outras palavras, quer dizer que os objetos da classe `Pedido` devem dispor de recursos para localizar os clientes a eles relacionados. Isso equivale a dizer no negócio que um pedido deve conter informações para a localização do cliente que o colocou, possivelmente o nome dele ou seu código para a localização no cadastro de clientes. Em um sistema, essa responsabilidade seria realizada por ponteiros ou chaves estrangeiras.

As navegabilidades podem ser unidirecionais (uma seta), bidirecionais (duas setas ou nenhuma seta, se for assim convencionado) ou indeterminada (nenhuma seta). Navegabilidades são usualmente raras em modelos conceituais, pois normalmente refletem a preocupação dos projetistas com a rapidez de acesso aos objetos na memória e a economia de espaço em disco, preocupações estas associadas à fase de projeto dos sistemas. Eu usaria (como usei na Figura 5.2) a navegabilidade de pedido para cliente para salientar a eventual necessidade manifestada pelo usuário em determinar ao cliente dado o pedido que ele fez.

5.11 Autoassociações de Classes

Autoassociações são associações entre objetos da mesma classe. São representadas no diagrama de classes usando-se poligonais ou arcos partindo de uma classe e chegando nela própria. A Figura 5.5 ilustra a situação em que um determinado subordinado pode estar sem chefe ou ter no máximo um chefe e um determinado chefe pode ter

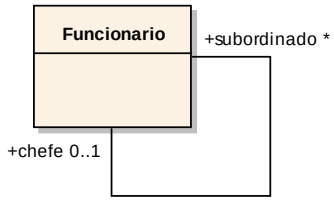


Figura 5.5: Autoassociação chefe-subordinado.

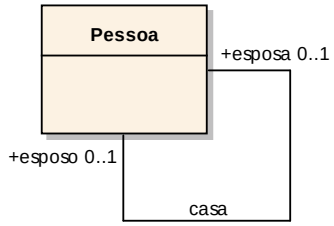


Figura 5.6: A autoassociação do matrimônio.

qualquer número de subordinados, eventualmente nenhum (a situação, por exemplo, de alguém que foi nomeado chefe de um setor que está sendo criado). A Figura 5.5 ilustra, ainda, que chefe e subordinados são funcionários.

Uma questão que frequentemente causa dúvidas surge quando, por exemplo, uma pessoa está associada a outra pelo casamento: de um lado da associação, uma pessoa assume o papel de esposo e, do outro, de esposa (ver Figura **??**).

A dúvida normalmente acontece porque lemos "um esposo está casado com zero ou uma esposa"... mas, na possibilidade de não estar casado, a pessoa não cumpre o papel de esposo.

A forma correta de entendermos o diagrama é que pode não haver a instância da associação de casamento entre duas pessoas. Havendo a instância, uma pessoa assume o papel de esposa e outra de esposo.

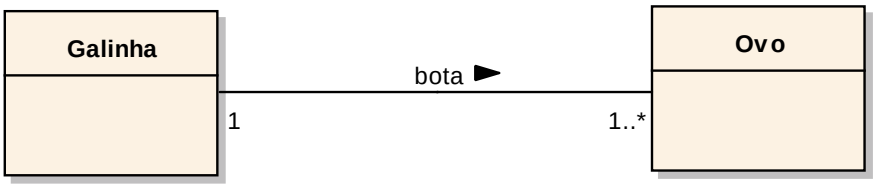


Figura 5.7: Modelo de classes conceitual do Sistema de Controle de Galinhas e Ovos – SCGO.

5.12 Multiplicidades no Projeto e na Implementação

A questão da autoassociação do matrimônio da Figura 5.6 nos leva a discutir outra questão que sempre vem à tona em sala de aula, que ilustraremos com a seguinte situação fantasiosa: o SCGO (Sistema de Controle de Galinhas e Ovos) que iremos desenvolver precisa manter um cadastro de galinhas e de ovos, associando cada ovo à galinha que o botou e cada galinha aos ovos que botou (os ovos serão identificados um a um). Nosso usuário se interessa por ter cadastradas apenas galinhas que tenham posto pelo menos um ovo, ou seja, toda galinha cadastrada no sistema deve estar associada a pelo menos um ovo. Nós estamos exagerando um pouco, claro, só para poder levantarmos a questão. O modelo conceitual de classes que passamos ao projetista do sistema ficou como o da Figura 5.7.

Algum tempo mais tarde, o projetista disse que não era possível projetar o banco de dados porque não era possível responder à clássica e filosófica pergunta: quem seria instanciado primeiro, o ovo ou a galinha? Segundo o modelo conceitual, não seria possível armazenar os dados de um novo ovo na tabela `OVO` antes de ter os dados da galinha que o botou armazenados na tabela `GALINHA`; e que não poderia armazenar os dados de uma nova galinha sem ter os dados de pelo menos um ovo que ela botou armazenados no banco de dados (veja as multiplicidades das pontas da associação `bota` na Figura 5.7.

A pergunta que normalmente me fazem em sala de aula é:

> "Nós, analistas, devemos *flexibilizar* o modelo conceitual, trocando a multiplicidade `1..*` por `0..*` para facilitar os trabalhos do projetista e do programador ou devemos expressar no modelo o que reza o conceito?"

Respondo à pergunta com outras perguntas:

> "De que modelo estamos tratando? De quem vocês acham que é o problema de projeto e implementação do banco de dados, afinal: dos analistas ou dos projetistas/programadores?"

As minhas perguntas, feitas da maneira a facilitarem a resposta, conduzem à resposta (óbvia) que devemos expressar, no modelo conceitual, o que reza o conceito, abstraindo os detalhes de implementação. O problema é, portanto, do projetista e do programador. O projetista provavelmente decidirá por "relaxar" as restrições de integridade do banco de dados, mas determinará que o programador abra uma transação de banco de dados para garantir, por programa, que as multiplicidades do modelo conceitual sejam obedecidas.

5.13 Resumo do Capítulo

Durante o levantamento de requisitos necessitamos especificar a estrutura da informação, especificando as relações atemporais entre os conceitos que compõem o domínio. Isso é feito usando os diagramas de classes da UML.

Os diagramas de classes em modelos de sistemas podem especificar as perspectivas conceitual, de especificação e de implementação. Na perspectiva conceitual representamos os conceitos do domínio e os relacionamentos que eles possuem entre si. Na perspectiva de implementação, todos os detalhes precisam estar representados para que possamos implementar o sistema. A perspectiva de especificação é o meio do caminho entre a perspectiva conceitual e a perspectiva de implementação, ou seja, indo do ponto em que colocamos a primeira característica do projeto até o ponto imediatamente anterior ao projeto 100% especificado, pronto para ser implementado.

No diagrama de classes de nível conceitual, foco do nosso texto, representamos apenas as classes de entidade (também chamadas de classes conceituais) e seus relacionamentos. As classes em um diagrama de classes são representadas por retângulos com compartimentos para seus nomes, seus atributos e para as operações que executam. Os atributos e as operações têm formas de notação definidas pela UML.

As classes conceituais se associam entre si. Nas pontas das associações colocamos as multiplicidades e, opcionalmente, os rótulos dos papéis. As multiplicidades especificam o número máximo e mínimo de objetos com os quais um determinado objeto se relaciona. Um papel, como o nome diz, especifica o papel que um objeto desempenha em sua associação com o outro. Associações também podem possuir rótulos, que servem para ajudar a compreendermos o significado da associação quando este não é claro e não é indicado pelos rótulos dos papéis.

Na próximo capítulo concluiremos o estudo dos diagramas de classes da UML, estudando os outros tipos de relacionamentos e demais conceitos e elementos da notação necessários à interpretação e elaboração de diagramas de classes mais abrangentes.

5.14 Exercícios Propostos

1. Identifique e nomeie as classes conceituais no texto a seguir. Lembre-se de que as classes conceituais são entidades sobre as quais nos interessa armazenar alguma informação. Relacione uma ou mais dessas informações para cada classe identificada, não se preocupando em ser completo. Apenas pense em algumas delas como mecanismos para identificar as classes.

   As universidades do município de Sertãozinho Alegre são divididas em um ou mais departamentos (Letras, Matemática etc.). Um departamento é chefiado por um de seus professores, mas há casos em que esse cargo está vago. Não há acúmulo de chefia. Os professores podem estar alocados em um ou mais departamentos. Um departamento pode ser criado sem que haja professores alocados a ele. Um aluno pode estar matriculado em mais de uma universidade e pode frequentar mais de uma disciplina na mesma universidade. As universidades podem não ter alunos matriculados. Cada departamento tem seu conjunto específico de disciplinas (pelo menos uma). Cada disciplina pode ser ministrada por um ou mais professores. Cada professor pode ministrar qualquer número de disciplinas.

2. Relacione e dê nomes adequados a alguns atributos que você imagina serem importantes para as classes identificadas no Exercício 1. Adote a notação da UML para formar rótulos completos de atributos. Use visibilidades, tipos, multiplicidades e valores *default* que julgar mais convenientes.

3. Complete a tabela 5.1 com nomes de associações entre as classes da Figura 5.2. Ao lado do nome coloque o sentido de leitura (da esquerda para a direita ou da direita para a esquerda). A dica é usar verbos em suas vozes ativas que exprimam bem o significado da associação.

4. Escreva como se leem nos dois sentidos as multiplicidades das associações entre as classes da Figura 5.2.

5. Agora que você já viu classes e associações, elabore o modelo conceitual de classes referente ao texto do Exercício 1 – Ambiente Acadêmico do Município de Sertãozinho Alegre.

As soluções encontram-se a partir da Página 212.

Tabela 5.1: Definindo os nomes de associações entre classes.

Item	**Classes Associadas**		**Nome da Associação**
1	PedidoDeReposicaoDeEstoque	ItemDeReposicaoDeEstoque	
2	Pedido	Cliente	
3	Produto	Fornecedor	
4	ItemDePedido	Produto	
5	ClientePJ	Funcionario	

CAPÍTULO 6

DIAGRAMAS DE CLASSES: OUTROS RELACIONAMENTOS ENTRE CLASSES, CLASSES DE ASSOCIAÇÃO, INTERFACES E RESTRIÇÕES

> Life is relationships; the rest is just details.
>
> Gary Smalley

No Capítulo 5 vimos que há diversas características de um sistema que não se enquadram como requisitos, mas que precisamos capturar de alguma forma para que o sistema seja projetado e construído. Há entidades do negócio (classes) que possuem informações (atributos) e responsabilidades (realizadas por operações) que se relacionam por meio de associações, através das quais a comunicação entre os objetos é possível.

Associações são um dos tipos possíveis de relacionamentos entre classes. Neste capítulo continuaremos e encerraremos o estudo de diagramas de classes da UML, apresentando outros tipos de relacionamentos que as classes podem manter entre si, além de outros conceitos importantes que completarão o ferramental necessário para a interpretação e elaboração de diagramas de classes de nível conceitual.

6.1 Especializações-Generalizações

A nossa empresa fictícia ZYX necessita manter em seu cadastro dois tipos diferentes de clientes: pessoas físicas (classe `ClientePF`) e pessoas jurídicas (classe `ClientePJ`), que possuem semelhanças e diferenças entre si: os endereços e telefones devem ser armazenados nos cadastros para todos os clientes, independentemente se são pessoas físicas ou jurídicas, os nomes são necessários apenas para as pessoas físicas e as razões sociais e nomes de contato são necessários apenas para as pessoas jurídicas.

Utilizando o relacionamento de especialização-generalização, os atributos, operações e relacionamentos comuns ficam na classe que chamamos de *superclasse* ou *classe-base*, enquanto as diferenças vão para as *subclasses*, também chamadas de *classes derivadas*. Estas herdam da superclasse os atributos, as operações e os relacionamentos comuns.

Sendo assim, os clientes pessoas físicas do modelo da Figura 6.1 (que é um trecho do diagrama da Figura 5.2) têm endereço, telefone e nome como atributos e estão associados a qualquer número de pedidos. Os clientes pessoas jurídicas, além do endereço e telefone, têm como atributos a razão social e o contato e estão associados a qualquer número de pedidos. Podem possuir, ainda, um representante de vendas, que é um funcionário da ZYX. Os relacionamentos de generalização-especialização são representados por setas com pontas triangulares vazadas.

Sempre me refiro a esse tipo de relacionamento como sendo de generalização-especialização porque ele é entendido como uma generalização (quando lemos no sentido da seta) e como especialização (quando lemos no sentido oposto ao da seta). Portanto, o sentido da seta indica a generalização; o sentido oposto indica a especialização. Assim, `Cliente` é uma generalização de `ClientePF` e de `ClientePJ`, enquanto `ClientePF` e `ClientePJ` são especializações de `Cliente`.

As generalizações são bem lidas como "é um tipo de": na Figura 6.1, cliente pessoa física é um tipo de cliente, e cliente pessoa jurídica é um (outro) tipo de cliente. Aliás, quando um bom nome para uma associação é "é um tipo de", devemos verificar se a associação deve dar lugar a um relacionamento de generalização-especialização.

> **ATENÇÃO:** Se a ponta da seta de um relacionamento de generalização-especialização não for vazada, ele não é um relacionamento de generalização-especialização ou não estamos falando de UML. Aliás, a necessidade do uso da forma gráfica correta vale para todos os demais símbolos usados na notação, que é bastante rigorosa a esse respeito.

As formas de apresentação de generalizações-especializações estão nas Figuras 6.2 e 6.3, indistintas quanto ao significado, conforme a UML. A forma da Figura 6.2 é

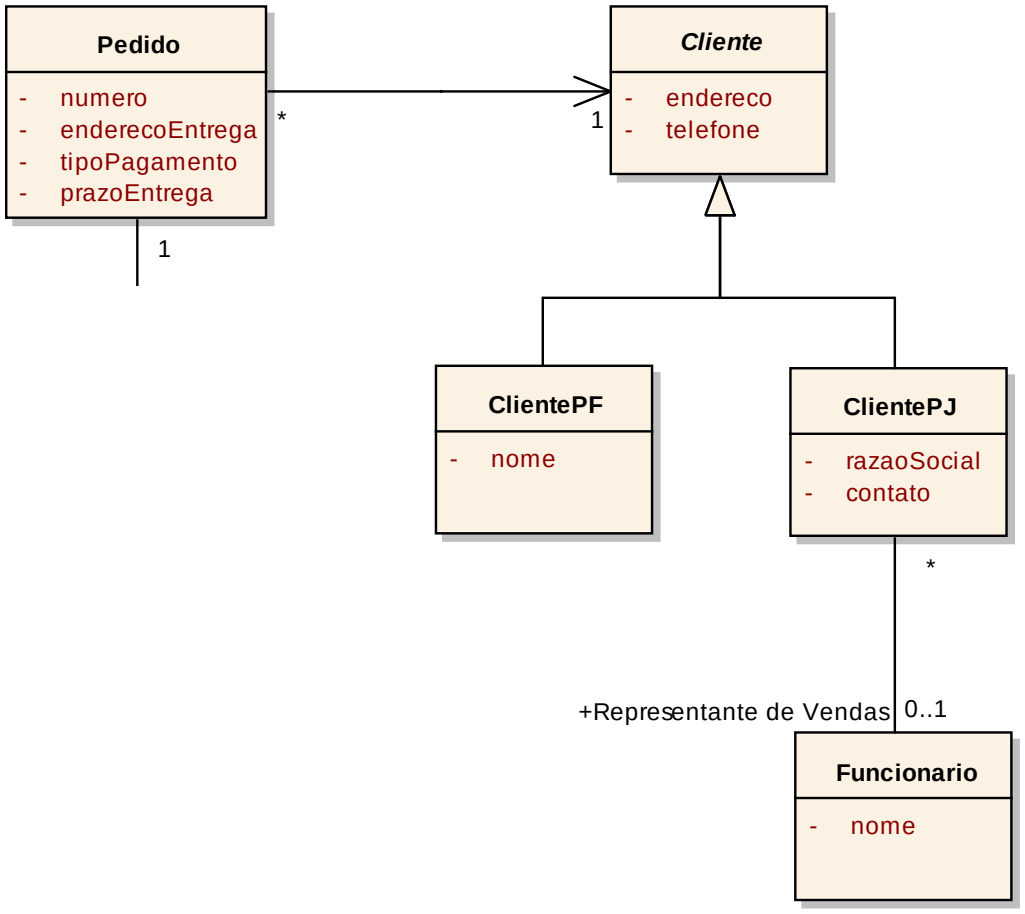


Figura 6.1: Relacionamento de generalização-especialização representado pela seta de ponta vazada (trecho da Figura 5.2).

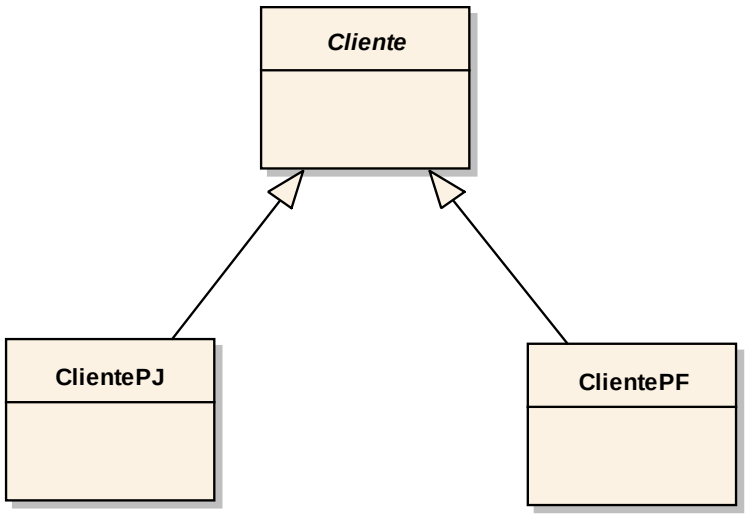


Figura 6.2: Generalização-especialização representada na forma direta ou oblíqua.

chamada de direta ou oblíqua, e a da Figura 6.3, de retilínea.

Cabe aqui explicar um detalhe sobre outro tipo de visibilidade de atributos e métodos que deixamos de explicar no Capítulo 5 porque ainda não tínhamos visto os relacionamentos de especialização-generalização: os atributos e métodos protegidos.

Em uma classe, ser protegido significa que o atributo ou método é protegido do acesso externo de forma geral, mas pode ser acessado pelos métodos das classes que a especializam. Um atributo ou método protegido tem sua visibilidade denotada por um #.

Como ilustra o diagrama da Figura 6.4a, o atributo `nome` da classe `Funcionario` só pode ser acessado diretamente pelo próprio objeto instanciado dessa classe, porque o atributo está marcado como sendo privado. Nesse caso, nem um objeto da classe `Engenheiro` tem acesso direto a esse atributo. Já no diagrama da Figura 6.4b, o atributo `nome` pode ser acessado diretamente por objetos das classes `Funcionario` e `Arquiteto`.

Há situações em que não queremos que uma classe possa ser *instanciada*, ou seja, que não possa haver objetos criados dela (ao contrário das classes concretas, que foram vistas no Capítulo 5). Essas classes são chamadas de *classes abstratas*. Por exemplo: na Figura 6.1, a classe `Cliente` foi definida como uma classe abstrata porque não pode haver objetos instanciados dessa classe, ou seja, não há nenhum cliente da

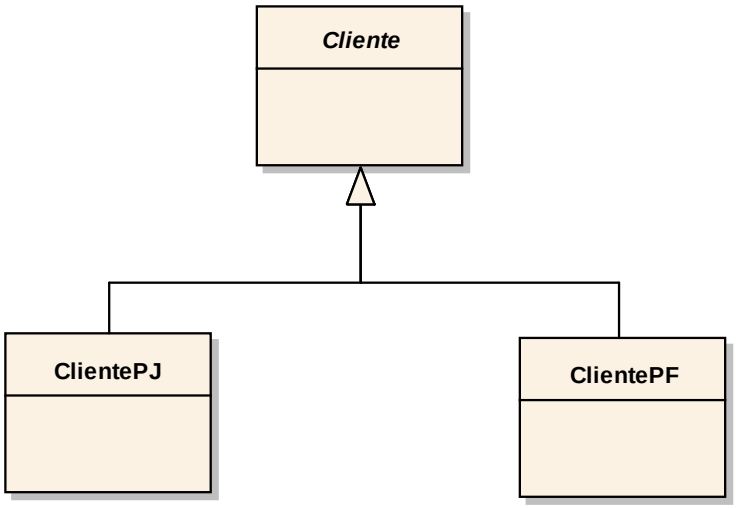


Figura 6.3: Generalização-especialização representada na forma retilínea.

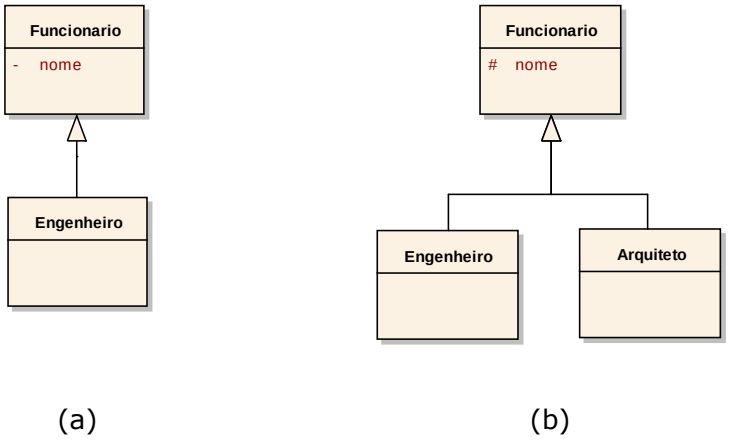


Figura 6.4: Atributos privado (a) e protegido (b) de classes especializadas definindo acessos distintos a objetos das classes que especializam.

ZYX que não seja pessoa física nem pessoa jurídica (eles têm de ser, obrigatoriamente, instâncias de `ClientePF` ou `ClientePJ` – quem nos disse isso foi o especialista do negócio na ZYX).

Classes abstratas existem no modelo somente para agruparmos em uma só classe os atributos, operações e associações comuns a duas ou mais classes. A esse processo de agrupamento, em superclasses, de atributos e operações comuns a duas ou mais classes damos o nome de *fatoração*.

Classes abstratas são denotadas na UML com seus nomes em itálico ou colocando `abstract` logo abaixo de seus nomes, dentro do compartimento do nome na caixa da classe.

Por fim, nada impede que façamos especializações de especializações em qualquer quantidade de níveis. A única recomendação é que muitos níveis de especialização (mais do que cinco, conforme cita a literatura) prejudicam o entendimento do modelo.

6.2 Conjuntos de Generalização e Partições

Conjuntos de generalização são agrupamentos de relacionamentos de generalizações-especializações em categorias. Por exemplo, podemos especializar uma pessoa quanto ao sexo e quanto ao estado civil. Essas categorias são ortogonais (independentes umas das outras), ou seja, uma mulher pode ser casada, solteira ou viúva etc., assim como um homem. Cada categoria representa um conjunto de generalização. A Figura 6.5 mostra a notação padronizada pela UML.

Repare que na Figura 6.5 as linhas tracejadas cortam os relacionamentos relativos ao mesmo conjunto de generalização.

Além de poderem se agrupar em conjuntos de generalização, as especializações também podem ser categorizadas em partições. Uma partição pode ser:

- completa (*complete*), quando as especializações geram ***todas*** as instâncias dos objetos (também chamada de cobertura total);
- incompleta (*incomplete*), quando os objetos podem ser instâncias das especializações ou da generalização (também chamada de cobertura parcial);
- disjunta (*disjoint*), quando as instâncias são de um tipo ***ou*** (exclusivo) de outro. Esse é o padrão quando não há nenhuma marcação de sobreposição no modelo;
- sobreposta (*overlapping*), quando as instâncias podem ser de um tipo ***e*** de outro(s).

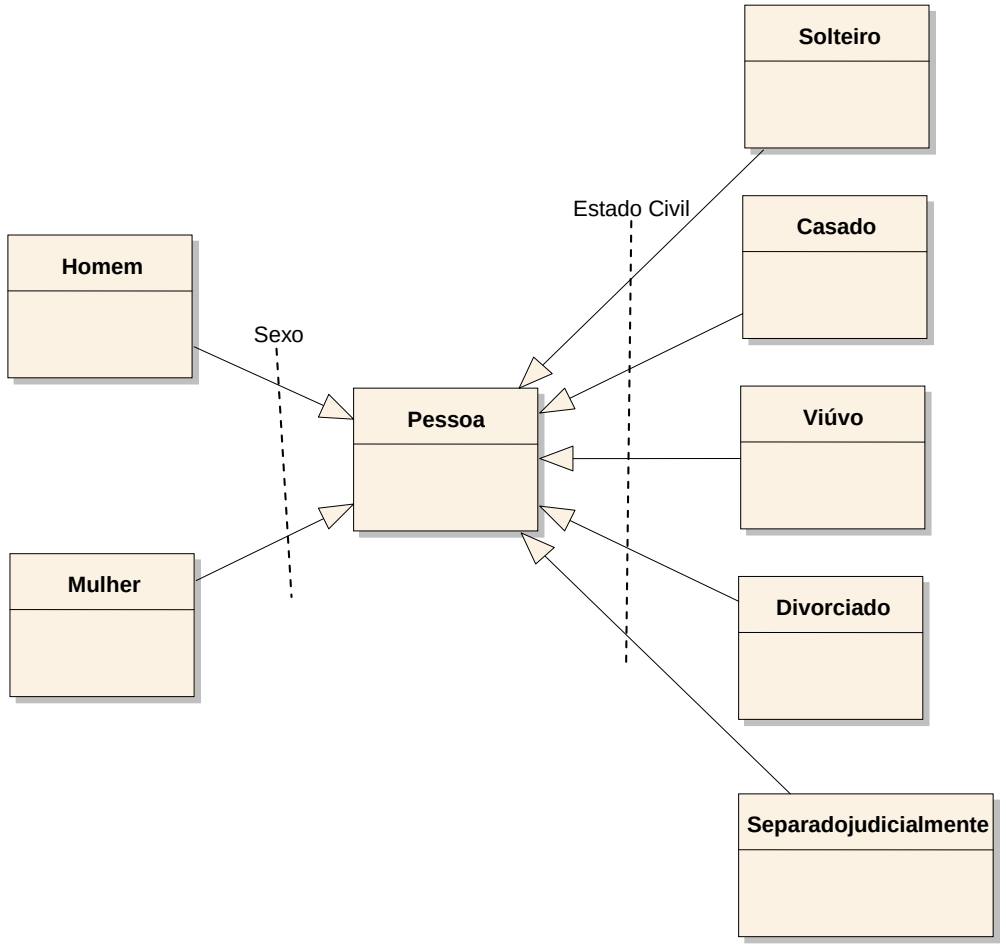


Figura 6.5: Conjuntos de generalização em diagramas de classes.

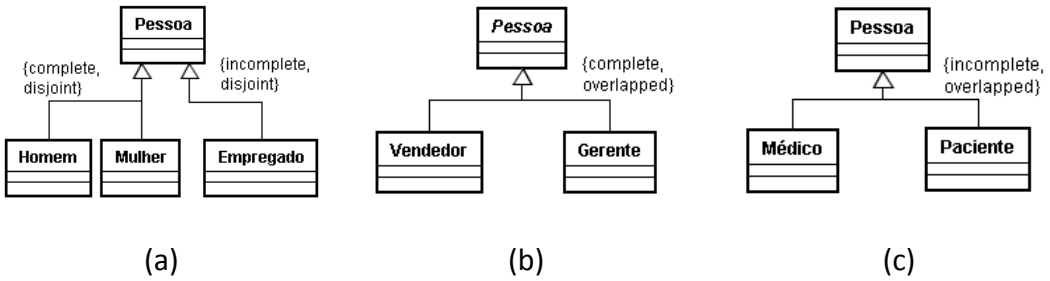


Figura 6.6: Partições em especializações.

A Figura 6.6a ilustra as situações em que a classe `Pessoa` é especializada de forma completa em `Mulher` e `Homem` (não há, segundo o modelo, uma instância de `Pessoa` que não seja mulher ou homem) e de forma disjunta (não há, segundo o modelo, uma instância que seja parte mulher e parte homem). A Figura 6.6a ilustra ainda as situações de as pessoas estarem empregadas ou desempregadas, ou seja, uma pessoa é uma instância da classe `Pessoa`, simplesmente, ou é uma instância da classe `Pessoa Empregada`.

A Figura 6.6b ilustra a situação em que uma pessoa atua como um vendedor e um gerente de vendas, ou seja, tem características dos dois. A Figura 6.6c ilustra situação similar, em que uma pessoa pode atuar como médico e ser paciente concomitantemente, além de poder não ser nem médico nem paciente.

É importante observarmos que especializações completas sugerem que as superclasses sejam classes abstratas, já que nessas especializações só podemos instanciar as subclasses.

6.3 Agregações

Há situações em que precisamos especificar conceitos que representam conjuntos de entidades. Nesse caso, além das entidades que correspondem às partes, os conjuntos também são entidades que devem ser representadas em nosso modelo. O conjuntos e as partes são ligadas entre si por meio de relacionamentos ditos de *agregação* – pois conjuntos agregam suas partes. Exemplos de conjuntos e suas partes são times de futebol e seus jogadores, empregados e seus dependentes, departamentos e suas

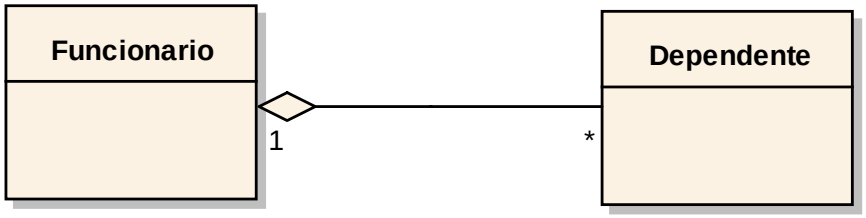


Figura 6.7: Relacionamento de agregação funcionário-dependente.

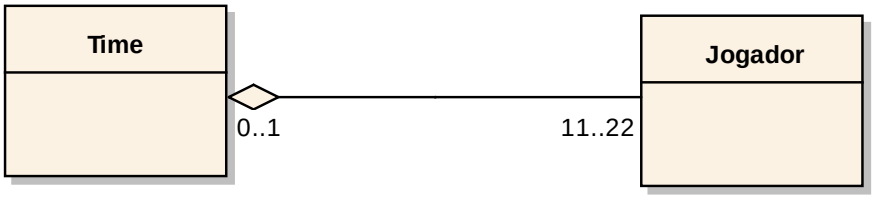


Figura 6.8: Relacionamento de agregação time-jogador.

divisões, divisões e seus colaboradores, etc.

O relacionamento de agregação é um relacionamento do tipo todo-parte; representamos o "todo" associado às "partes" que o compõem.

A UML possui uma notação especial para agregações: um losango vazado, conforme ilustrado na Figura 6.7. Nessa figura representamos os funcionários em uma organização e seus dependentes: cada dependente está associado a um funcionário específico, enquanto funcionários têm qualquer número de dependentes, eventualmente nenhum.

A Figura 6.8 ilustra a situação em que um time é composto de 11 a 22 jogadores e que cada jogador ou não está associado a um time ou está associado a, no máximo, um time.

A Figura 6.9 ilustra a situação em que os departamentos de uma organização são subdivididos em no mínimo duas divisões e estas são divididas em no mínimo duas coordenações. Cada coordenação está associada à sua respectiva divisão e cada divisão ao seu respectivo departamento.

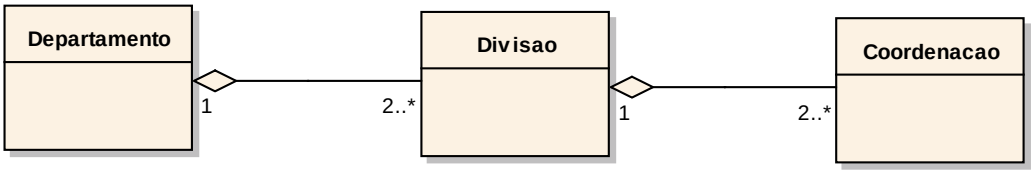


Figura 6.9: Relacionamento departamento-divisão-coordenação.

As agregações são bem lidas com o verbo "tem". Nos exemplos anteriores, funcionários têm dependentes, times têm jogadores e departamentos têm divisões, que têm coordenações.

Agregações são entendidas por muitos autores como "placebos" de modelagem, ou seja, não adicionam semântica ao modelo que justifique seu emprego. Dessa forma, as agregações ilustradas nas Figuras 6.7, 6.8 e 6.9 poderiam ser substituídas por associações simples, sem nenhuma perda de expressividade. Isso é devido à colocação das multiplicidades e à possibilidade de colocar o rótulo "tem" dando nome à associação. Portanto, agregações só se justificam quando estamos em um nível de abstração tão alto que não representamos sequer as multiplicidades.

6.4 Agregações Compostas (ou Composições)

Agregação composta, também chamada composição, é um tipo mais forte de agregação. É também um relacionamento todo-parte representado por um losango cheio (veja a Figura 6.10). A principal, e fundamental, diferença entre composições e agregações é que nas composições as partes não podem pertencer, em um mesmo instante, a mais do que um todo. Como consequência, se o todo deixa de existir, as partes também deixam. É importante notar que, quando permitido pela multiplicidade, uma parte pode ser removida da composição antes de ela deixar de existir.

A Figura 6.10 traz de volta a questão do "placebo" de modelagem, especificando, na forma de composição, o relacionamento entre um funcionário de uma organização e seus dependentes. A pergunta que provavelmente você se fará é: qual é, afinal, a diferença entre os significados dos modelos das Figuras 6.7 e 6.10? A resposta é: neste caso, absolutamente nenhuma. Isso acontece porque, nos dois casos, se um funcionário deixa de existir, os seus dependentes também deixam, pois não pode haver

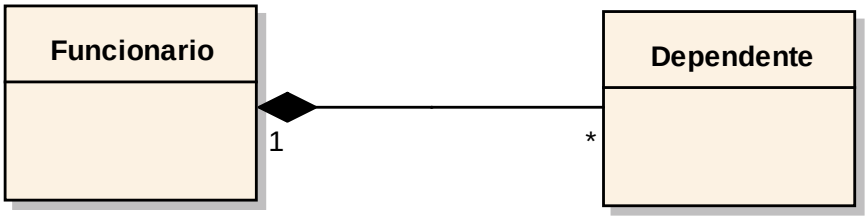


Figura 6.10: Agregação composta.

dependentes não ligados a funcionários (a multiplicidade obrigatória 1 significa isso).

Quando uma entidade parte está associada no diagrama a somente uma entidade todo, como na Figura 6.10, a composição pode ser reduzida a uma agregação que, por sua vez, por se tratar de um "placebo", pode ser eliminada do modelo, reduzindo-se a uma simples associação.

As composições são indispensáveis quando, no modelo, uma entidade parte está associada a mais de uma entidade todo e queremos especificar que uma instância da parte não pode estar associada, ao mesmo tempo, a mais de uma instância de entidade todo. Por exemplo, segundo a Figura 6.11, se um determinado funcionário está associado a uma determinada empresa, não pode estar associado, ao mesmo tempo, a um sindicato.

Na Figura 6.11, uma empresa (o todo) tem qualquer número de funcionários (as partes) associados a ela, da mesma forma que sindicatos (o outro todo). Além disso, um funcionário pode estar associado a nenhuma ou uma empresa ou a nenhum ou a um sindicato. O que o modelo especifica ainda, por se tratar de composições, é que, se um funcionário está associado a uma empresa, não pode estar associado ao mesmo tempo a um sindicato[1], já que uma parte não pode pertencer a mais de um todo nas composições.

[1] Nesse exemplo, não estamos considerando a propriedade ou não da relação sindicato-funcionário-empresa, pois sequer conhecemos qualquer Lei que obrigue ou impeça o que representamos no modelo da Figura 6.11.

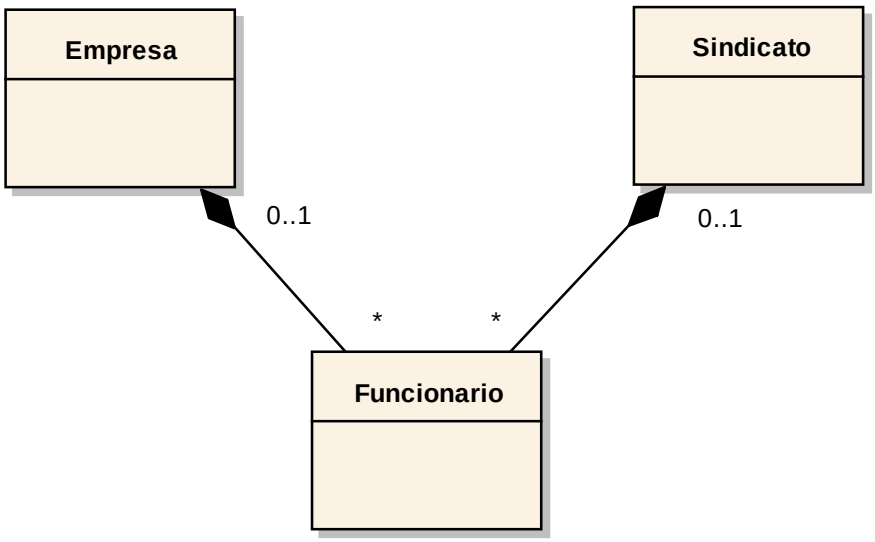


Figura 6.11: Composições definindo excludência mútua de instâncias de associação.

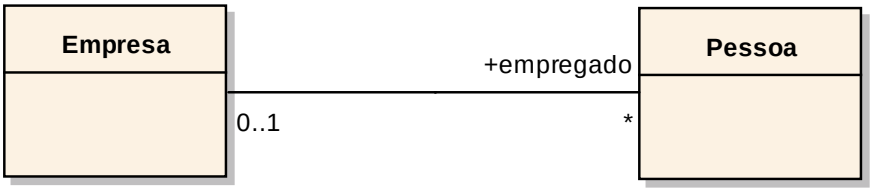


Figura 6.12: Associação de emprego entre uma pessoa e uma empresa.

6.5 Classes de Associação

Considere a situação do diagrama da Figura 6.12, que especifica que uma pessoa está ou não associada a uma empresa por meio de um emprego.

Onde seriam armazenados, nesse caso, os atributos matrícula e salário de um empregado de uma empresa? Na classe `Pessoa`? Na classe `Empresa`?

Repare que não é certo colocar esses atributos na classe `Pessoa`, porque há

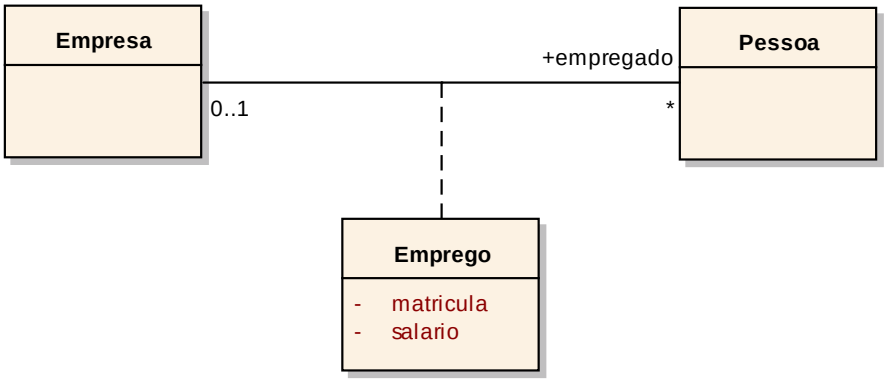


Figura 6.13: Classe de associação contendo atributos e operações relativas a uma associação.

pessoas que nunca trabalharam e não trabalharão em nenhuma empresa e, portanto, não devem ter esses atributos. Também não é certo colocar os atributos na classe `Empresa` porque, segundo o modelo, empresas podem não ter empregados.

Na realidade, esses atributos passam a ser necessários somente quando uma pessoa se associa a uma empresa como seu empregado; os atributos `matricula` e `salario` são, portanto, atributos das associações entre pessoas e empresas.

Atributos e operações relativos a uma associação devem estar reunidos em uma classe chamada *classe de associação*. Classes de associação são representadas conforme ilustra a Figura 6.13.

No caso da Figura 6.13, demos à classe o nome de `Emprego`, pois ela armazena atributos e operações dos vínculos empregatícios entre pessoas e empresas.

Devemos entender classes de associação da seguinte forma: para cada uma das associações existentes entre empresas e empregados, ou seja, para uma associação de emprego (o João como empregado da ZYX, por exemplo), temos uma instância da classe `Emprego`[2], com espaço para armazenar o valor de uma matrícula e de um salário. Esse conceito é bastante satisfatório porque, afinal, uma pessoa só tem emprego

[2]Alguns CASEs rotulam automaticamente as associações, dando a elas os nomes das respectivas classes de associação. Assim, no exemplo da Figura 6.13, teríamos um rótulo `Emprego` dando nome à associação entre `Pessoa` e `Empresa`. Não acho isso muito bom porque, além de ser um rótulo que não precisa ser colocado (o nome da associação é uma decorrência óbvia da existência da classe de associação), ferimos aquele *macete* de nomear associações com verbos na voz ativa. Se trocamos o nome da associação, esses CASEs automaticamente trocam o nome da classe, mas verbos na voz ativa não são bons nomes para classes. Dessa forma, nesses casos, eu opto que o CASE não exiba o nome das

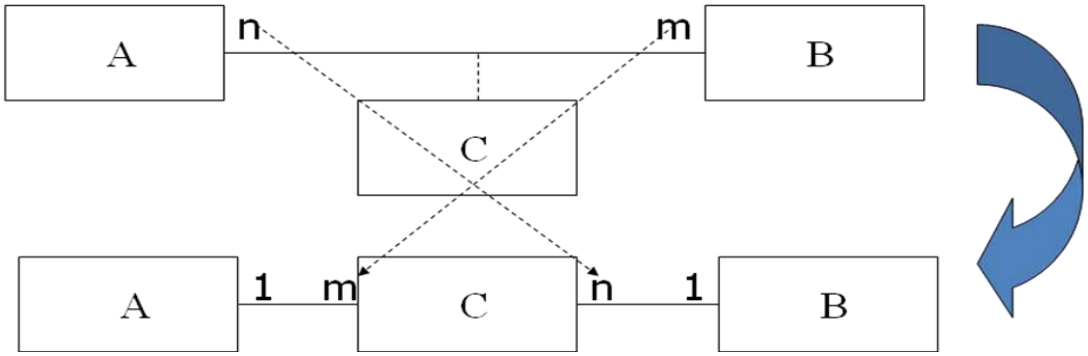


Figura 6.14: Promoção da classe da associação `C` à classe cheia.

quando trabalha em uma empresa.

Essa ideia ilustra esta importante característica: classes de associação só podem ser instanciadas uma única vez para uma dada instância da associação (uma dada empresa associada a um determinado empregado). Costumo até sugerir que os alunos imaginem que existe uma multiplicidade 1 na ponta do segmento de reta tracejado, junto à classe `Emprego`... mas que só imaginem, porque colocar a multiplicidade em uma classe de associação é um erro!

Como, para o exemplo da Figura 6.13, só podemos armazenar um único conjunto de atributos do emprego, ou seja, um salário, uma matrícula, isso significa que não podemos armazenar valores históricos de salário, por exemplo. Com isso, se uma empresa decide aumentar o salário de um empregado, esse novo valor sobrescreve o valor anterior do atributo.

Se precisamos de históricos dos valores dos atributos, não podemos usar classes de associação; devemos promovê-las a classes ditas *cheias* (classes comuns), o que é usualmente feito conforme a técnica ilustrada na Figura 6.14.

Justificamos a técnica de transformação ilustrada na Figura 6.14 da seguinte forma: cada instância da classe `A` está associada a m instâncias da classe `B` e, para cada uma dessas associações, temos uma instância da classe de associação `C`. Raciocínio análogo vale no sentido da leitura de `B` para `A`.

Com isso, flexibilizando as multiplicidades obrigatórias 1 resultantes da transformação, podemos especificar históricos convenientemente.

---

associações.

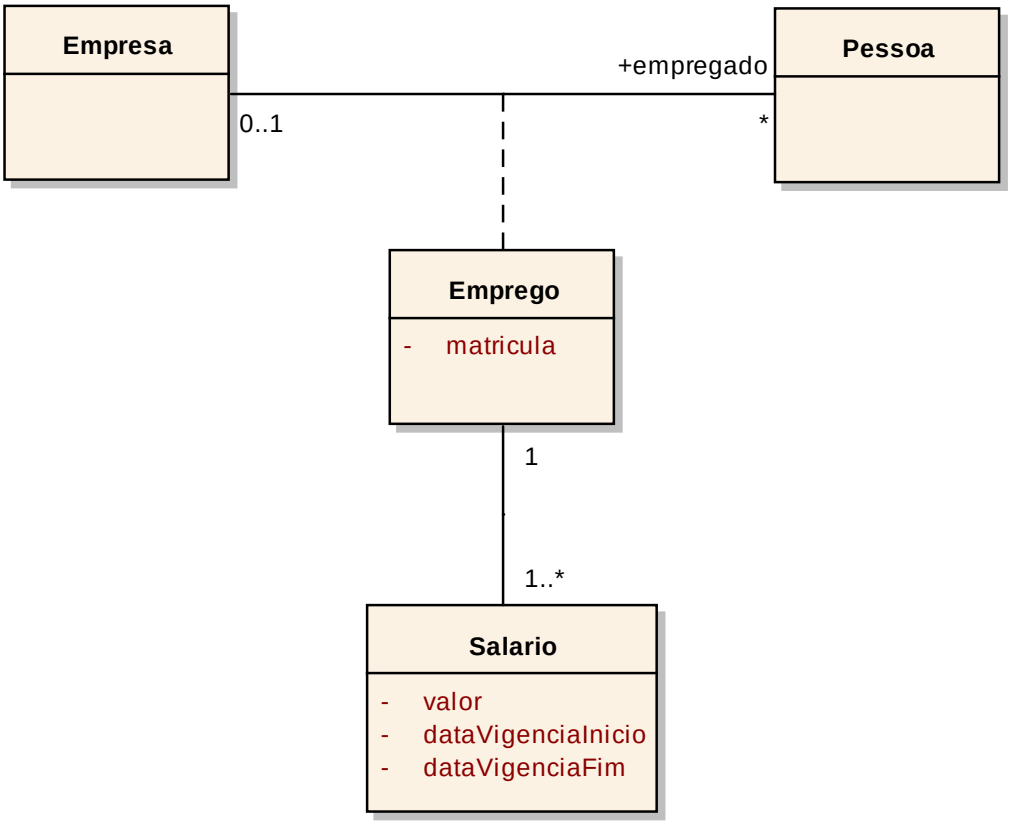


Figura 6.15: Alternativa para armazenar o histórico dos salários mantendo o uso de classe de associação.

Temos outras alternativas para os históricos dos salários da Figura 6.13, caso houvesse essa necessidade. Podemos, por exemplo, definir salário como um atributo multivalorado (`salario [0..*]`) ou podemos associar a classe `Emprego` a uma classe `Salario`, que poderia até armazenar um período de vigência. Dessa forma, podemos trabalhar convenientemente com as multiplicidades nas pontas da associação. A Figura 6.15 ilustra esta última alternativa.

6.6 Operações Abstratas, Interfaces e Dependência

Quando tratamos de classes abstratas, não mencionamos que elas podem possuir *operações abstratas*. Operações abstratas não possuem implementação (o código), apenas a declaração da operação e de seus atributos. Chamamos essa declaração de *assinatura da operação*. As operações abstratas precisam ser implementadas para que possam ser invocadas. Isso é feito nas classes concretas que especializam as abstratas que contêm as tais operações abstratas.

As operações abstratas nos conduzem a mais um conceito e a dois outros tipos de relacionamentos importantes contemplados na UML: o conceito de *interfaces* e os relacionamentos de *dependência* e de *realização* das interfaces.

Para ilustrar o que acabamos de expor, imagine a seguinte situação. Suponha que precisamos desenvolver um editor gráfico para vários ambientes (Windows/Intel, Sun(X), Mac etc.). Queremos isolar, ao máximo, os aspectos conceituais da aplicação dos aspectos de implementação (aspectos do hardware e do ambiente operacional) de cada ambiente, ou seja, a aplicação permitirá a manipulação de textos, linhas, retângulos e círculos de acordo com determinadas regras e forma de interação com o usuário, independentemente do ambiente em que a aplicação estará operando. Suponha que meu editor gráfico precise utilizar as primitivas gráficas $drawText(p : Ponto, texto : String)$, $drawLine(p1, p2 : Ponto)$, $drawRect(p1, p2 : Ponto)$ e $drawCircle(c : Ponto, r : integer)$ para manipulação das formas gráficas na tela.

Podemos modelar essa aplicação na forma da Figura 6.16.

Na Figura 6.16, as classes que compõem a aplicação (não nos interessa saber quais são para o efeito da ilustração) estão dentro do pacote `Editor Gráfico` (pacotes são contêineres onde colocamos classes, atores, casos de uso, outros pacotes...; serão vistos em detalhes no Capítulo 11).

A classe `JanelaGrafica` foi marcada como `«interface»`, significando que ela é uma interface. As operações dessa classe foram grafadas em itálico, significando que são operações abstratas e, como tais, não possuem implementação; apenas apresentam as assinaturas das operações. Interfaces são necessariamente classes abstratas que só têm métodos abstratos.

O pacote `Editor Gráfico` se associa à interface por meio de uma seta tracejada com a ponta aberta, que significa *dependência*. A dependência indica que as classes do editor gráfico dependem, ou usam, a interface.

A interface `JanelaGrafica` é realizada, ou seja, tem suas operações implementadas, pelas operações com os mesmos nomes das classes `JanelaWin`, `JanelaMac` e `JanelaX`. A realização é representada por um relacionamento que se assemelha a uma generalização-especialização, porém com linhas tracejadas.

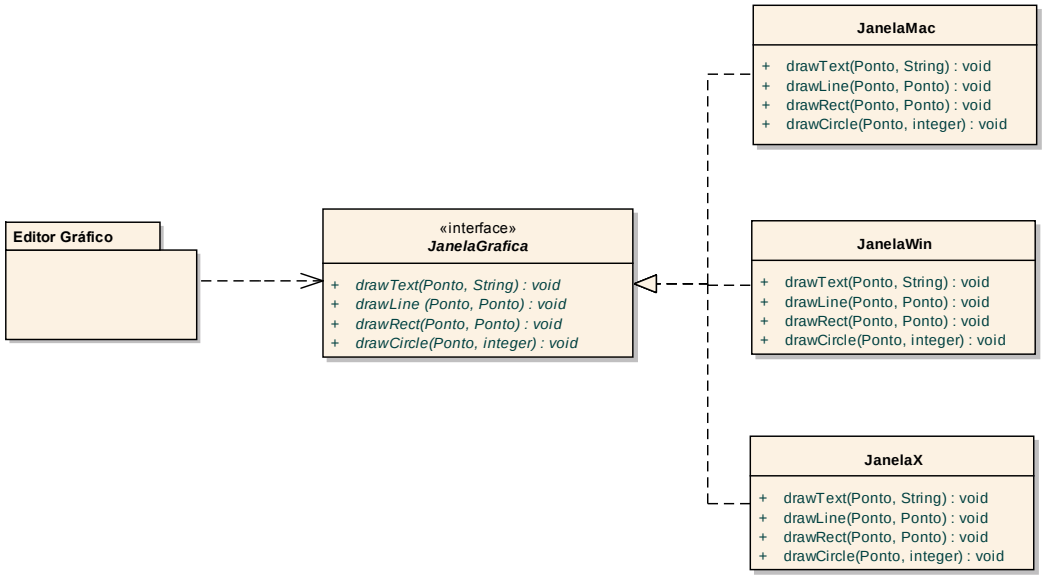


Figura 6.16: Exemplo de interface e classes de realização dessa interface.

O principal uso para classes de interface é o isolamento entre partes conceitualmente distintas e coesas de um sistema (entre subsistemas, por exemplo), diminuindo o acoplamento entre elas.

6.7 A Especificação de Restrições nos Modelos

Uma restrição é uma sentença formulada com o propósito de declarar alguma semântica adicional de um elemento no modelo ([10]). Restrições são regras a serem obedecidas durante a execução do sistema, como por exemplo limites superiores e inferiores que devem ser obedecidos para determinadas variáveis, expressões do tipo *SE... ENTÃO... SENÃO* que fazem parte de especificações de regras de negócio, necessidade de pertinência a um conjunto de valores possíveis (por exemplo, o atributo `sexo` poder assumir somente os valores "M" ou "F"), expressões matemáticas de cálculo etc.

As restrições podem ser especificadas em linguagem natural, português estruturado ou usando uma linguagem legível por máquina (linguagem de programação de computadores, por exemplo), no caso de necessitar ser mais formal

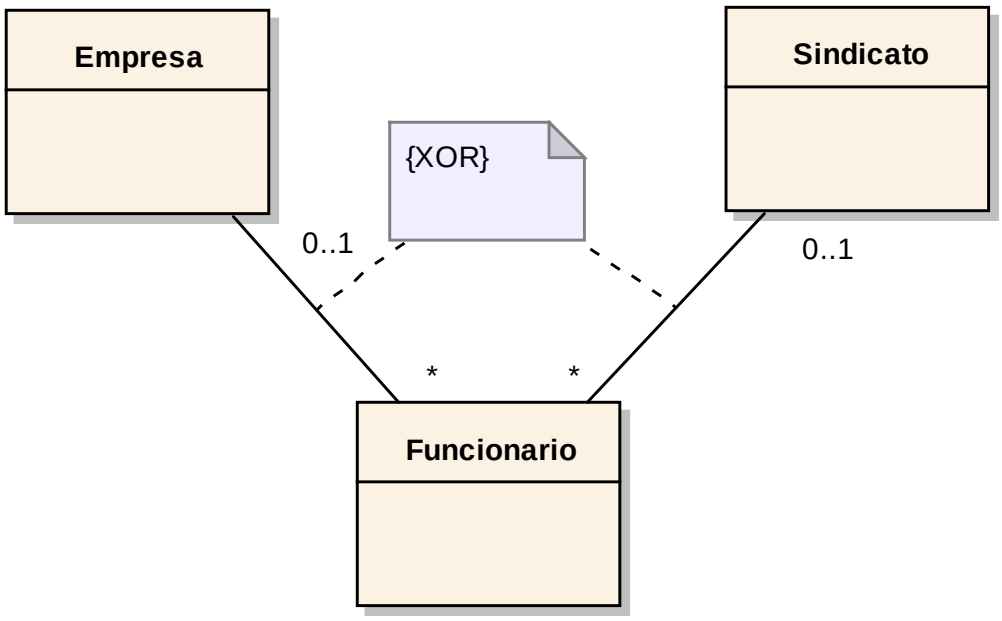


Figura 6.17: Forma alternativa de especificação de associações mutuamente excludentes.

ou de se pretender que o modelo sirva como insumo para geração automática de código. O OMG estabeleceu uma linguagem para especificação de restrições chamada OCL (*Object Constraint Language* – linguagem de restrições de objetos), descrita em um volume à parte da especificação da UML. Restrições são necessariamente especificadas nos modelos com seus textos entre "" e "".

Algumas restrições, tais como `{XOR}` (ou exclusivo), `{AND}` e `{ORDERED}` (em que os elementos associados à restrição são ordenados de alguma forma), são predefinidas e muito usadas na UML.

Citamos como exemplo a forma alternativa de especificar a mesma semântica das agregações compostas, com a expressão que define exclusão mútua aplicada a associações, como ilustra a Figura 6.17.

A Figura 6.17 ilustra a especificação de uma restrição `XOR` aplicada às associações `Funcionário-Empresa` e `Funcionário-Sindicato`. `XOR` é o acrônimo para "*exclusive OR*" ("ou exclusivo"), que significa, no caso, que existe uma e apenas uma instância de associação: entre um funcionário e uma empresa ***ou*** entre um funcionário e um sindicato.

As restrições podem ser especificadas em qualquer diagrama da UML idealmente considerando seus tipos, ou seja, restrições de cunho estrutural são preferencialmente colocadas nos modelos que contemplam a dimensão estrutural (diagramas de classes são espaços muito bons para isso); restrições de cunho temporal são colocadas preferencialmente nos modelos em que a dimensão temporal é contemplada etc.

6.8 Resumo do Capítulo

Utilizando especializações-generalizações, os atributos, operações e relacionamentos comuns são colocados na superclasse, enquanto as diferenças vão para as subclasses, que herdam da superclasse os atributos, as operações e os relacionamentos comuns. Especializações de uma classe podem ser agrupadas em conjuntos de generalização e categorizadas segundo partições.

Quando queremos representar conjuntos e os elementos que os formam, usamos os relacionamentos todo-parte: agregações e composições. Agregações são semanticamente equivalentes a associações quando representamos as multiplicidades em suas pontas, mas composições acrescentam a restrição de que toda parte não pode pertencer a mais do que um todo.

As classes de associação podem ser usadas quando as associações entre classes possuem atributos e/ou operações e suas aplicações nos modelos se restringem a situações em que não há mais do que uma instância da classe de associação para a mesma instância da associação. O emprego de classes de associação em um modelo sempre pode dar lugar ao emprego de classes normais.

Usamos interfaces quando queremos diminuir o acoplamento entre partes conceitualmente distintas de um sistema, isolando, por exemplo, *o que fazer* do *como fazer*.

As restrições são limitações a serem observadas durante o funcionamento de um sistema, impostas por regras de negócio visando à manutenção de consistências, dentre outras razões. Elas devem ser especificadas nos modelos por meio de expressões do tipo *SE... ENTÃO... SENÃO*, expressões matemáticas de cálculo, em linguagem natural, português estruturado ou usando uma linguagem legível por máquina, como Java. A UML conta com uma linguagem própria para especificação de restrições: a OCL da OMG. Nos modelos, restrições são necessariamente colocadas entre "" e "".

6.9 Exercícios Propostos

1. Desenvolva o modelo de classes conceituais com base no texto a seguir, indicando os relacionamentos de associação e de especialização-generalização, as multiplicidades e os atributos das classes (fatore os atributos e associações comuns usando especializações-generalizações).

   O sistema de controle de uma biblioteca deve possuir usuários que se classificam em usuários comuns e usuários funcionários. Para todo e qualquer usuário é necessário que o seu nome esteja no cadastro. Usuários funcionários possuem um nome de login e uma senha de acesso ao sistema. Usuários comuns têm número de registro e se subdividem em alunos e membros da comunidade.

   Para cada obra armazena-se o ISBN, os autores, o título e os assuntos. Os exemplares possuem data de inclusão no acervo e marcação de disponibilidade. A biblioteca só cadastra obras que constem do acervo.

   A biblioteca empresta livros aos alunos e aos membros da comunidade. Alunos podem retirar até dois títulos concomitantemente; membros da comunidade podem retirar apenas um título por vez. Usuários funcionários não tomam livros emprestados.

2. Desenvolva o modelo de classes considerando o texto a seguir:

   Estamos desenvolvendo um editor gráfico para permitir o desenho de polígonos (especificados por conjuntos ordenados de três ou mais pontos ligados por segmentos de retas) e círculos (especificados por seus centros e raios) na tela.

   As restrições a seguir devem ser observadas durante a execução do editor e, portanto, precisam ser especificadas no modelo:

   - *um polígono possui três ou mais vértices;*
   - *um vértice de um polígono não pode ser ao mesmo tempo vértice de outro polígono nem centro de um círculo;*
   - *o centro de um círculo não pode ser centro de outro círculo;*
   - *ao comandarmos a remoção de um polígono, removemos também seus vértices. Ao comandarmos a remoção de um círculo, removemos também seu centro.*

3. Transforme a classe de associação da Figura 6.13 em classe cheia usando a técnica ilustrada na Figura 6.14.

4. Suponha que queiramos isolar as classes do subsistema de controle de vendas da ZYX das do subsistema de cadastro de clientes. Como podemos representar, em

alto nível de abstração, esse isolamento e, ao mesmo tempo, essa dependência mútua, se entendermos que cada subsistema compõe um pacote distinto?

5. Na solução dada para o Exercício 1 mencionamos que a solução dada é parcial. A razão disso é que deixamos de apresentar uma restrição importante. Verifique se as associações entre as instâncias da classe `Exemplar` e as das classes `UsuarioComumAluno` e `UsuarioComumMembroComunidade` podem ocorrer simultaneamente e, caso não possam, aplique a restrição que completa a solução dada para aquele exercício.

As soluções encontram-se a partir da Página 215.

CAPÍTULO 7

DIAGRAMAS DE MÁQUINA DE ESTADOS

> Anybody can make history. Only a great man can write it.
>
> Oscar Wilde

Por vezes, uma determinada classe possui instâncias que atravessam um número significativo de estados. Por exemplo, no contexto bancário, uma determinada conta-corrente cinco estrelas, ou seja, uma determinada instância da classe `Conta-Corrente-Especial`, pode passar do estado `superavitária` ao estado `credora` pela ocorrência de um saque a descoberto. Ela também pode passar de `credora` ao estado `bloqueada` pelo atingimento do limite do cheque especial. Pode, também, passar de `bloqueada` ao estado `superavitária vip platinum plus plus` pela ocorrência do depósito de um prêmio de R$ 50.000.000 da mega-sena. Repare que nem todas as instâncias da classe `Conta-Corrente-Especial` estão, passaram ou passarão pelos mesmos estados durante seus ciclos de vida[1].

A passagem de um estado a outro, que chamamos de *transição*, é o caminho entre dois estados a ser trilhado à medida que acontecem eventos (no contexto bancário, por exemplo, os cheques que são sacados, os prêmios que são depositados etc.).

Podemos tentar descrever os estados, as transições e os eventos usando textos

[1] Ciclo de vida de um objeto é o tempo decorrido entre a sua instanciação e a sua destruição.

em linguagem coloquial. Porém, nos casos em que o número de estados, eventos e transições é grande e as condições para as transições são complexas, o texto possivelmente se tornaria bastante longo, confuso e pleno de ambiguidade. Imagine, por exemplo, uma classe cujas instâncias podem passar por sessenta estados, transitando por 90 transições!

Por essa razão, a UML trouxe do passado os diagramas de máquina de estados (DMEs).

DMEs especificam não só os estados pelos quais os objetos podem passar e as possíveis transições entre os estados, mas também as reações dos objetos aos eventos que os atingem.

Portanto, diagramas de máquina de estados representam as possíveis *biografias* dos objetos da classe enfocada, cobrindo todos os possíveis ciclos de vida desses objetos.

7.1 Conceitos Iniciais Importantes

Considere a entidade `Pedido` no contexto da nossa empresa fictícia ZYX, conforme o diagrama de classes da Figura 5.2. Queremos especificar as possíveis situações (estados) em que os pedidos (instâncias da classe `Pedido`) podem estar. Como exemplo de algumas possibilidades que precisamos considerar:

- o pedido de número 346, que João fez, está na rua, para entrega;
- o pedido de número 349, da Maria, está aguardando a reposição de estoque de determinado produto, para que possa ser embalado e despachado;
- o pedido de número 330, que Pedro colocou, foi devolvido;
- o de número390481, que Bete acabou de colocar, está sendo verificado para determinar se todos os itens que o compõem estão disponíveis em estoque.

Temos, com isso, mais um exemplo de que instâncias diferentes de uma mesma classe podem possuir (ou estar em) estados diferentes.

Com base nesses exemplos e nas demais informações que os especialistas no negócio da ZYX deram, desenvolvemos o DME apresentado na Figura 7.1. Não se preocupe em entender agora os elementos da notação gráfica usada. Ela será explicada em detalhes ao longo do texto adiante.

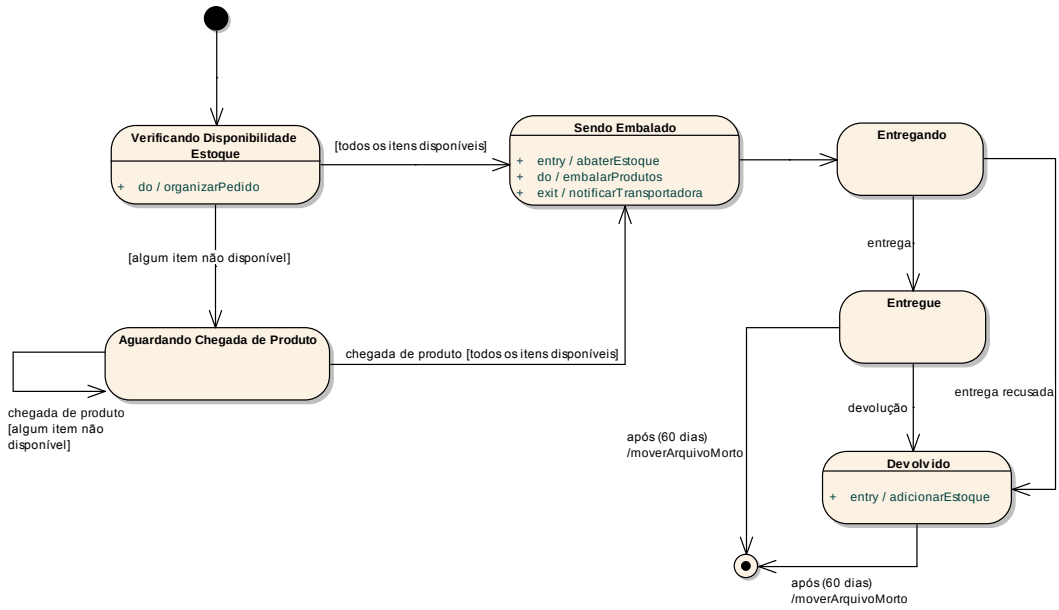


Figura 7.1: Diagrama de máquina de estados para os objetos da classe `Pedido`.

Nem todos os pedidos seguem o mesmo caminho[2]. Possivelmente há pedidos que, ao serem colocados, tiveram todos os seus itens em estoque, não tendo passado pelo estado `Aguardando Chegada de Produto`. Haverá pedidos que não serão entregues por conta de recusa no recebimento, passando diretamente de `Entregando` a `Devolvido`. Há pedidos que foram acolhidos na entrega, permanecendo, da entrega em diante, no estado `Entregue`.

Pelo fato de um DME "contar as histórias" dos objetos de uma classe, costumo referir-me a eles como sendo as possíveis *biografias* desses objetos. Sendo assim, desenvolvemos diagramas de máquina de estados para classes de objetos que possuem histórias que valem a pena ser contadas ou, em linguagem mais técnica, possuem comportamento dinâmico relevante, passando por vários estados e respondendo a diversos eventos. Seria o equivalente a um biógrafo só investir seu tempo na biografia de alguém que tem ou teve uma vida que despertará o interesse dos leitores e que, por isso, merece ser contada. Para isso, verificamos se cada classe do modelo de classes "merece" ter um DME desenvolvido para ela.

[2]Os caminhos são indicados no diagrama pelas setas – as transições.

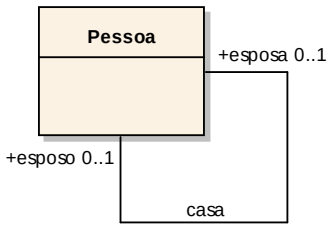


Figura 7.2: A autoassociação do matrimônio.

Nas seções a seguir você verá os conceitos relativos a cada um dos elementos da notação gráfica da UML para DMEs. Faremos isso com base no diagrama da Figura 7.1.

7.2 Estados

Cada estado em que um objeto se encontra reflete a situação dos atributos e dos relacionamentos desse objeto. Veja dois exemplos para ilustrar essa afirmação.

1. Determinados valores de atributos de uma pessoa evidenciam que essa pessoa está (no estado) saudável ou não. Os exames de sangue ajudam a conhecer os valores desses diversos atributos.

2. Se um objeto da classe `Pessoa` se encontra associado a outro objeto dessa mesma classe por meio de uma associação de casamento (como vimos no Capítulo 5, essa associação é chamada de autoassociação), ou seja, se há uma instância da autoassociação ilustrada pela Figura 7.2, dizemos que o estado civil de cada uma das duas pessoas desse autorrelacionamento é `casado`.

Os estados são representados por "retângulos" com os vértices arredondados, com um ou dois compartimentos. O nome do estado é colocado centralizado no único compartimento ou no primeiro, quando há dois compartimentos. A Figura 7.3 ilustra.

O segundo compartimento é opcional e é destinado a relacionar, quando for necessário, as atividades a serem executadas assim que o objeto entra no estado, imediatamente antes de sair do estado e enquanto permanece no estado, dependendo do *rótulo*. Para isso, a notação usada no segundo compartimento é

$$rótulo + "/" + atividade$$

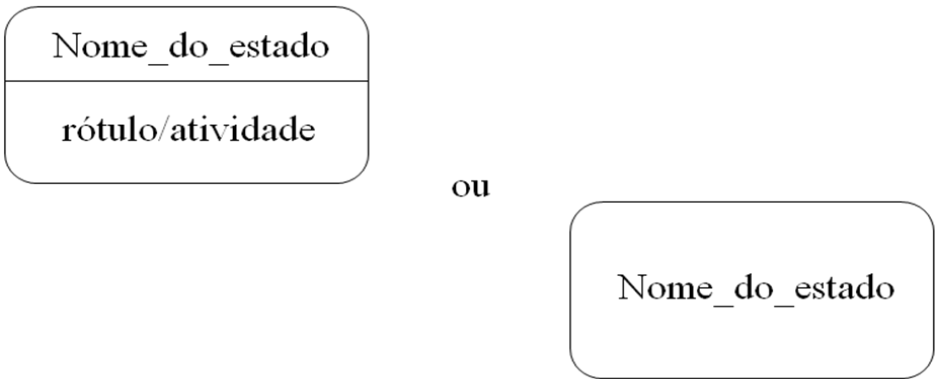


Figura 7.3: Duas formas de apresentação da caixa de estado em DMEs da UML.

Figura 7.4: Rótulos indicando atividades nos estados (um detalhe da Figura 7.1).

O *rótulo* indica quando a *atividade* será executada: se imediatamente após a entrada (rótulo = "*entry*" – entrada, em inglês), se durante a permanência do objeto no estado (rótulo = "*do*" – fazer, em inglês) ou se imediatamente antes da saída (rótulo = "*exit*" – saída, em inglês). A "/" serve para separar rótulo e atividade.

Sendo assim, a Figura 7.4 ilustra a situação de um pedido que, ao entrar no estado `Sendo Embalado`, executa a atividade `abaterEstoque`. Durante a permanência do pedido nesse estado, a atividade `embalarProdutos` é executada. Imediatamente antes de o pedido sair desse estado, a atividade `notificarTransportadora` é executada.

Qualquer estado é necessariamente de um dos tipos descritos a seguir:

- estado de atividade (por exemplo, `Verificando Disponibilidade Estoque`, `Sendo Embalado` e `Entregando`);
- estado de espera (ex.: `Aguardando Chegada de Produto`);
- estados que refletem que o objeto atingiu determinada condição (por exemplo, `Entregue` e `Devolvido`).

Ser um estado de atividade significa que o objeto executa alguma tarefa enquanto permanece no estado. Estados de atividade devem, portanto, possuir, no segundo compartimento, um rótulo "`do`" indicando a atividade a ser executada enquanto o objeto permanece no estado. Essas atividades duram indefinidamente ou terminam em algum momento. Neste caso, quando as atividades terminam (quando se trata de um cálculo, por exemplo), o objeto vai transitar para outro estado pela ocorrência do evento de final da atividade sendo executada (trataremos de eventos mais detalhadamente adiante, neste capítulo). Ter rótulo(s) "`entry`" e/ou "`exit`" não caracteriza que um estado é de atividade.

Um estado de atividade deve sugerir que algo é feito enquanto o objeto se encontra naquele estado. Os nomes de estados de atividade devem ser, portanto, verbos no gerúndio (por exemplo, `Calculando Tal Coisa`, `Gerando Tal Coisa`, `Pesquisando Tal Coisa`...).

Estados que refletem uma situação do objeto devem ter os nomes com verbos no particípio (por exemplo, `Tal Coisa Calculada`, `Tal Coisa Gerada`, `Tal coisa Pesquisada`...).

Os estados de espera devem ter nomes como `Aguardando Tal Coisa` ou `Esperando Tal Coisa` (por exemplo, o estado `Aguardando Chegada de Produto`, da Figura 7.1).

Os nomes das atividades executadas – seja na entrada, durante a permanência no estado ou na saída dele – devem ter preferencialmente verbos no infinitivo (por exemplo, `calcular tal coisa`, `gerar tal coisa`, `pesquisar tal coisa`...).

> **ATENÇÃO:** Quem estiver elaborando o modelo deve ter a preocupação de dar "bons nomes" aos estados. Os nomes devem ser de fácil entendimento por parte dos especialistas do negócio e usuários (a dica, mais uma vez, é usar o jargão do negócio), além de refletirem o tipo do estado.

7.3 Pseudoestados Iniciais

Um *pseudoestado inicial* não é exatamente um estado (daí o porquê do "pseudo" no nome), mas aponta para o estado em que o objeto se encontrará quando for criado (estado inicial) ou quando entrar em uma região que contém outros estados (subestados dessa região, portanto). Só poderá haver um único pseudoestado inicial por região.

A notação gráfica do pseudoestado inicial é ilustrada na Figura 7.5, empregada na Figura 7.1 para indicar que o estado `Verificando Disponibilidade Estoque` é o

Figura 7.5: Círculo pequeno preenchido: símbolo gráfico de estado inicial na UML (um detalhe da Figura 7.1).

Figura 7.6: "Olho de boi": símbolo gráfico de estado final na UML (um detalhe da Figura 7.1).

estado que um objeto da classe `Pedido` assume quando é criado.

7.4 Estados Finais

Há estados que os objetos atingem dos quais não há saída, ou seja, dos quais não há transições de saída para outros estados. Atingidos esses estados, os objetos permanecerão neles *para sempre*. Esses estados são chamados de *estados finais*. Imaginemos, para ilustrar, a situação de um equipamento que se danifica e para o qual não há reparo e não há a possibilidade de destruição ou venda: ele permanecerá para sempre como "Danificado" no inventário.

Uma forma de representarmos um estado final em um diagrama é simplesmente usarmos a notação para estados explicada na Seção 7.2, contanto que dele não haja qualquer transição de saída.

Há um outro tipo de notação para estados finais: o símbolo do tipo "olho de boi", ilustrado na Figura 7.6 e empregado nas saídas dos estados `Devolvido` e `Entregue` da Figura 7.1.

Há alguma controvérsia quanto ao emprego da notação da Figura 7.6. Por essa razão fazemos referência à UML ([11]), que define que o estado *olho de boi* "significa que a máquina de estados correspondente à região[3] que o contém terminou".

[3]Uma região pode ser um superestado (um estado que contém outros – já mencionamos isso na Seção sec:PseudoEstadosIniciais), parte de um estado concorrente (de que trataremos no final deste capítulo) ou um diagrama como o da Figura 7.1 (o caso em que não há nenhuma dessas regiões caracterizadas).

Os empregos do "olho de boi" em superestados e em estados concorrentes serão vistos mais adiante. Se estamos falando do emprego do "olho de boi" em um diagrama como o da Figura 7.1 (com uma única região), devemos entender que ele é um estado final, como seria qualquer outro do diagrama do qual não houvesse qualquer transição de saída. Nesse caso, estaríamos usando uma notação diferente, apenas.

O que dissemos no parágrafo anterior pode ser dito de outra forma: um estado final é um estado que, se atingido, o objeto não sai dele. O "olho de boi" em um diagrama, ou é um estado final (quando o diagrama tem uma única região), ou indica o final da máquina de estados da região (quando o diagrama possui mais de uma região – o que veremos mais adiante neste texto).

Dois equívocos bem comuns cometidos pelo pessoal de modelagem quando usa essa notação:

1. Não é incomum que alguns modeladores considerem que o símbolo de "olho de boi" não corresponde ao estado final, mas apenas indica o estado final do objeto.
2. O estado "olho de boi" também é associado por muitos outros modeladores à destruição do objeto, o que não está incorreto, porque a destruição é um dos possíveis estados finais de um objeto, mas não é a única situação final possível para um objeto. Um objeto em estado final pode ter atributos, o que significa que ele não foi necessariamente destruído ao atingi-lo.

Faço a seguir duas recomendações relacionadas à representação de estados finais em diagramas de máquina de estados:

1. evite usar "olhos de boi" em diagramas que não representem términos de regiões em estados compostos. Reserve essa notação para as situações de fim da máquina de estados, em diagramas com estados compostos e para a remoção de objetos nos demais diagramas;
2. se um objeto permanece "para sempre" em um estado importante para o negócio, ou seja, se não há transição de saída desse estado, modele esse estado como um estado normal (caixa retangular com as bordas arredondadas), e não como um "olho de boi". O fato de não haver transição de saída significa, inequivocamente, que aquele é um estado final.

7.5 Eventos, Condições e Ações em Transições

Um objeto, estando num determinado estado, pode passar para outro estado quando certas "coisas" acontecem. Transições definem os possíveis caminhos de passagem de

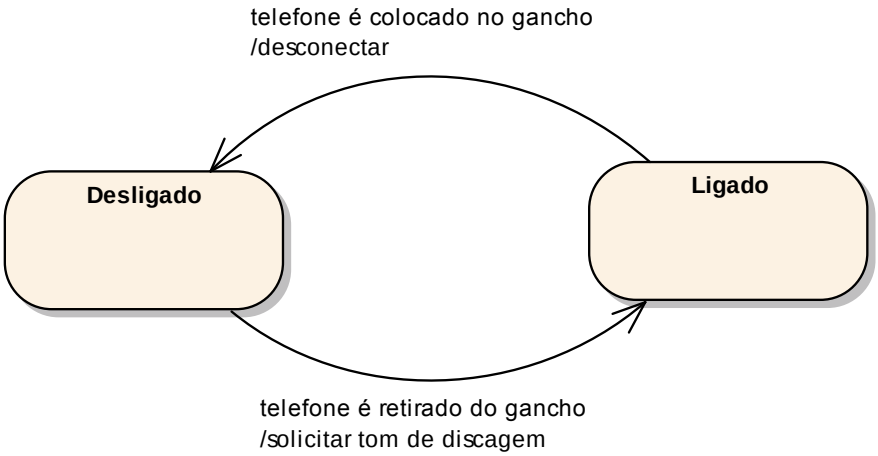


Figura 7.7: Transições mútuas entre dois estados.

um estado a outro, que se dão pela ocorrência de eventos (acontecimentos... as tais "coisas" que mencionamos), atendidas determinadas condições.

As transições são representadas na UML por setas poligonais ou curvilíneas com as pontas abertas. As setas são obrigatoriamente unidirecionais. Sendo assim, se desejamos representar caminhos de ida e de volta entre dois estados, precisamos representá-los como duas transições; uma de ida e outra (distinta) de volta.

A Figura 7.7 ilustra as transições mútuas entre dois estados usando situação aparentemente trivial do DME para objetos da classe `Telefone`.

Note que não representamos o pseudoestado inicial na Figura 7.7 porque julgamos não ser relevante, no caso, saber em qual estado um telefone se encontra quando é instanciado (a instanciação de um objeto telefone pode ser associada, por exemplo, ao registro do mesmo no sistema da operadora de telefonia). Caso isso seja necessário, podemos representar um pseudoestado inicial apontando para o estado `Desligado` ou para o estado `Ligado`, conforme queiramos.

Transições podem partir de um estado e chegar nele próprio. Transições desse tipo têm o nome especial de autotransições. A Figura 7.8 ilustra o trecho da Figura 7.1 que apresenta uma autotransição.

Como dissemos, transições são trilhadas *obrigatoriamente* pela ocorrência de eventos. Se não ocorrem eventos, os objetos permanecem nos estados em que se encontravam.

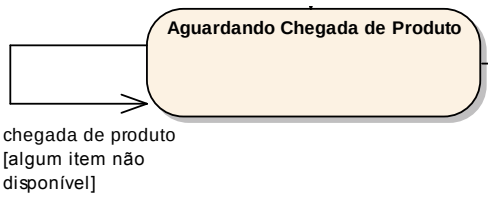


Figura 7.8: Autotransição em DMEs da UML (um detalhe da Figura 7.1).

Os eventos podem ser *externos* ou *temporais*.

Eventos externos acontecem no ambiente exterior ao que estudamos e, sendo relevantes para o sistemas, demandam que algum processo do sistema seja executado em resposta ao acontecimento. São exemplos de eventos externos a colocação de um pedido e a chegada de uma fatura em um sistema de controle de estoque.

Eventos temporais se dão de três formas:

1. pela chegada de determinada data/hora. Esses eventos são chamados de eventos temporais fixos e são normalmente especificados como *é hora de*;
2. pela passagem de uma quantidade de tempo relativa à entrada em um estado. Eventos desse tipo são chamados de eventos temporais relativos. Eles são usualmente especificados como *após tanto tempo*;
3. términos de atividades que estavam sendo executadas por um objeto quando em um estado como, por exemplo, o cálculo que estava sendo realizado enquanto o objeto estava no estado `Calculando Tal Coisa`.

Pela ocorrência de um evento, um objeto pode transitar do estado em que se encontra para outro. Em alguns casos, precisa ser atendida alguma condição adicional para que, dada a ocorrência do evento, o objeto de fato mude de estado. Pode ser necessário que, adicionalmente, um objeto execute alguma ação durante a transição.

Pelo fato de precisarmos especificar o evento, a condição e a ação, as transições são rotuladas com o nome do evento com a condição que precisa ser avaliada para se saber se o objeto transita ou não e com a ação a ser executada durante a transição. Essas três informações formam, na ordem, os chamados "rótulos ECA" (Evento, Condição e Ação) das transições.

Segundo a UML, a sintaxe dos rótulos das transições é

$$evento[condição]/ação$$

Um exemplo de um rótulo completo para uma transição entre os estados, suponhamos, `Encaminhado` e `Entregue` de um pedido é:

$$entrega[prazosuperou24horas]/aplicarmultaporatraso$$

Eventos, sendo acontecimentos, devem ter nomes condizentes: `entrega`, para significar que a entrega ocorreu, `alocação do técnico`, para indicar que um técnico foi alocado à ordem de serviço. Outros exemplos de bons nomes para eventos são `entrega recusada`, para significar que ocorreu a recusa do recebimento pelo cliente e `pedido devolvido`, para significar que o pedido foi devolvido pelo cliente.

Os colchetes devem conter a expressão a ser avaliada (a condição para que, na ocorrência do evento, a transição seja trilhada). Por exemplo: `[prazo superou 24 horas]`.

As ações, sendo processos executados pelo sistema, devem ser especificadas com verbos no infinitivo. Exemplo: `/aplicar multa por atraso`. A barra (`"/"`) precede necessariamente a ação.

Assim, no exemplo completo para um rótulo de evento acima, na ocorrência da entrega, se o prazo superou 24 horas, o objeto passará de um estado a outro (de `Encaminhado` para `Entregue`) executando a ação aplicar multa por atraso. Se o prazo não superou 24 horas, o objeto não transitará de um estado a outro, mesmo que a entrega ocorra (nesse caso, provavelmente outra transição a ser especificada no modelo será trilhada).

Eventos, condições e ações podem ser omitidos dos rótulos das transições, conforme você pode ver a seguir.

Se o evento é omitido, significa que o evento[4] corresponde ao final da atividade executada no estado do qual o objeto está saindo. Eventos só podem ser omitidos, portanto, quando o estado do qual o objeto está saindo é um estado de atividade e queremos especificar que o evento é o fim da atividade. Na Figura 7.9, quando termina a verificação da disponibilidade em estoque dos itens do pedido (termina a atividade `organizarPedido`), o pedido vai transitar para o estado `Aguardando Chegada de Produto` ou `Sendo Embalado`, dependendo se há ou não algum item do pedido faltante no estoque.

Se a condição é omitida (nesse caso, os colchetes também devem ser omitidos), a transição será trilhada incondicionalmente. No caso ilustrado pela Figura 7.10, a entrega ocorre incondicionalmente.

Se a ação é omitida, a transição ocorrerá sem que uma ação seja executada. Ainda no caso ilustrado pela Figura 7.10, a entrega é trilhada sem que nenhuma ação do sistema seja executada.

[4] Lembre-se de que é obrigatório que um evento ocorra para que a transição possa ser trilhada.

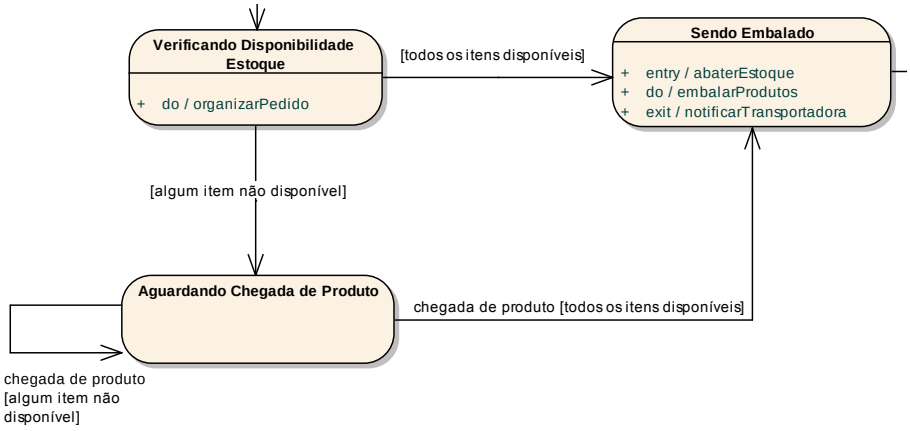


Figura 7.9: Nome do evento omitido, significando o fim da atividade do estado de origem (um detalhe da Figura 7.1).

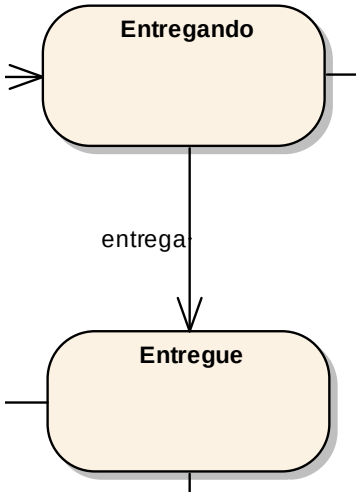


Figura 7.10: Condição omitida, significando que a transição ocorre incondicionalmente (um detalhe da Figura 7.1).

Caso haja mais de uma transição de saída de um estado, não pode haver dúvida a respeito de qual transição será trilhada; ou os eventos são diferentes ou, sendo o mesmo evento, as condições são diferentes. Isso caracteriza o que se chama de *exclusão mútua*; múltiplas transições de saída de um estado são *mutuamente excludentes*, o que nos leva a concluir que um objeto só pode estar em um só estado em determinado instante. No caso das duas transições de saída do estado `Verificando Disponibilidade Estoque`, ilustrado na Figura 7.9, ambas ocorrem pelo evento de fim da atividade `organizarPedido` porque, como vimos, não há rotulo de evento. No entanto, apenas uma transição será trilhada, porque as condições `[todos os itens disponíveis]` e `[algum item não disponível]` nunca serão avaliadas como verdadeiras ao mesmo tempo.

7.6 Estados Compostos

Um estado pode conter outros estados, concorrentes ou independentes. Esses estados são chamados *estados compostos*.

Quando verificamos que um estado pode ser subdividido em outros (sub) estados, precisamos desenvolver um diagrama para mostrar seus detalhes. No exemplo da Figura 7.7, o estado `Desligado` é trivial pois, enquanto está desligado, um telefone permanece aguardando uma ligação ou a retirada do gancho. No entanto, enquanto estiver no estado `Ligado`, um telefone pode estar pronto para fazer uma ligação, emitindo o sinal de discagem; pode estar emitindo o sinal de ocupado; pode estar no meio da entrada do número a ser chamado; pode estar falando; pode estar tentando estabelecer a ligação etc. Sendo assim, dizemos que o estado `Ligado` é um superestado e os estados `Pronto`, `Discando`, `Número Inválido`, `Conectando` etc. são subestados daquele estado. A Figura 7.11 ilustra esse conceito, mostrando os subestados representados dentro do superestado `Ligado`. O diagrama da Figura 7.11 é, portanto, um detalhamento do diagrama da Figura 7.7, que era aparentemente trivial; talvez você até tenha se perguntado por que eu o desenvolvi, já que ele era trivial.

Note na Figura 7.11 três aspectos que merecem destaque:

1. o pseudoestado inicial apontando para o subestado `Pronto` indica que este estado é o subestado em que o objeto se encontrará assim que entrar no (super)estado `Ligado`;

2. o objeto `telefone` pode passar imediatamente de qualquer subestado do estado `Ligado` para o estado `Desligado` pela simples colocação do telefone no gancho, ou seja, a saída do estado `Ligado` pela ocorrência do desligamento provoca a saída do subestado onde o telefone se encontra;

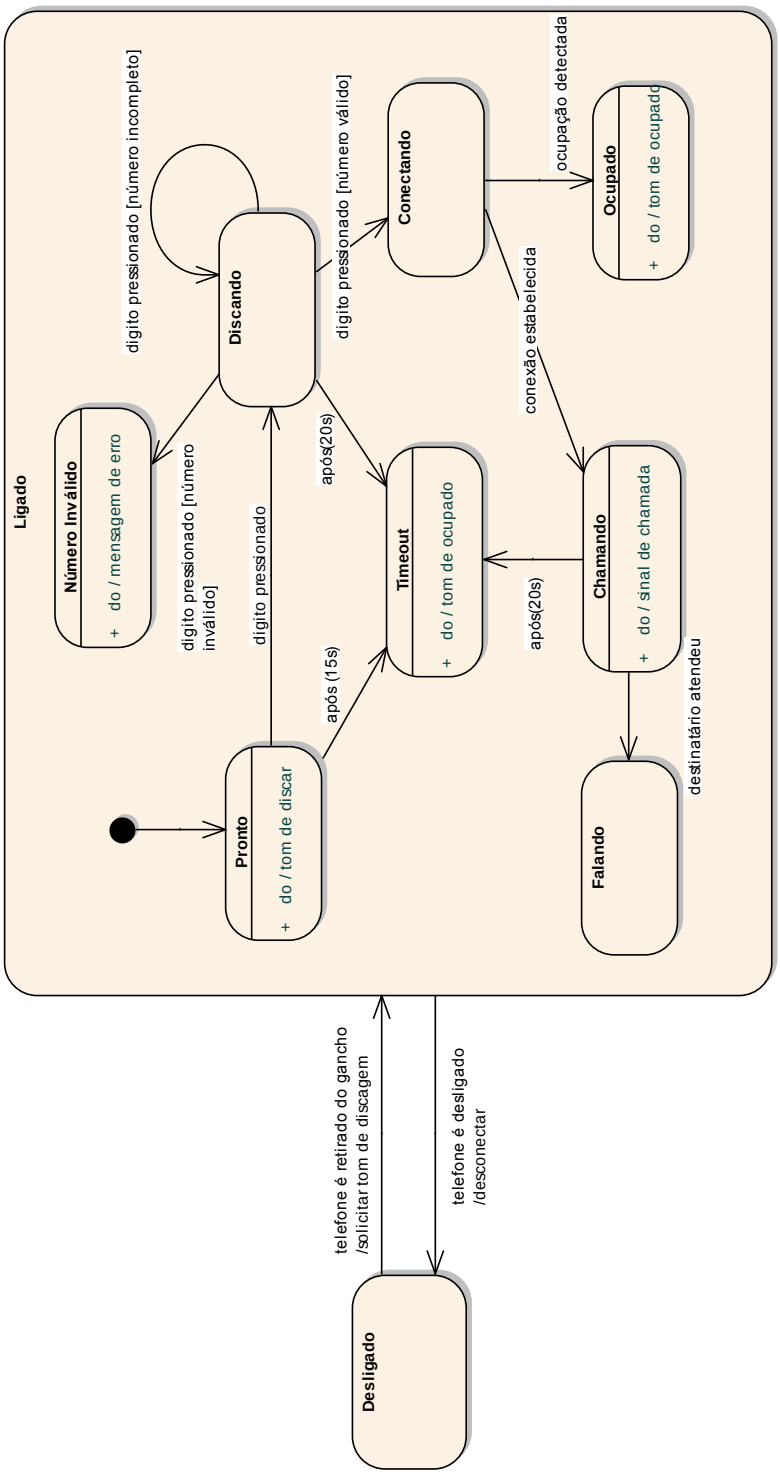


Figura 7.11: Estados e seus subestados: um detalhamento da Figura 7.7.

3. a situação que acontece quando tiramos o telefone do gancho e nada fazemos por quinze segundos é ilustrada na figura pela passagem do estado `Pronto` ou do estado `Discando` para o estado `Timeout` (tempo esgotado): o sinal de ocupado é iniciado, sinalizando que o telefone "cansou de esperar" uma ação do operador. O evento de tempo esgotado é a passagem de quinze segundos sem que nenhum dígito tenha sido teclado, representado por `após (15s)`.

Caso seja relevante especificar em qual estado um telefone se encontra quando é instanciado, devemos representar no diagrama da Figura 7.11 um pseudoestado inicial apontando para o estado `Desligado` ou para o estado `Ligado`. Com isso, teremos dois pseudoestados iniciais no mesmo diagrama. Essa situação é um exemplo de contextos distintos a que nos referimos anteriormente: o primeiro deles no nível dos estados Ligado-Desligado e o outro no nível dos subestados do estado `Ligado`.

De volta à nossa empresa ZYX: além dos estados relacionados ao processo de entrega de um pedido, é esperado que os estados relacionados ao processo de pagamento pelo pedido sejam também relevantes; usualmente, um pedido só é expedido se o pagamento por ele for feito ou, pelo menos, agendado. Atributos relacionados ao pagamento (data de pagamento, valor, descontos, impostos etc.) e atributos relacionados à composição e entrega do pedido (o conjunto de itens, as datas de expedição e entrega, o local de entrega etc.) formam subconjuntos distintos e independentes do conjunto de atributos de um pedido, dando origem a dois estados ditos *concorrentes*. Estados concorrentes são estados independentes entre si, relativos a aspectos distintos, mas que acontecem ao mesmo tempo.

É importante ressaltar que não estamos contradizendo o que escrevemos antes (que um objeto só pode estar em um único estado em um dado instante), e sim acrescentando um conceito novo: o de estados concorrentes.

A Figura 7.12 apresenta outros aspectos além dos da Figura 7.1; também são considerados os estados relacionados ao pagamento pelo pedido.

Com isso, um pedido estará, ao mesmo tempo, em um estado da parte superior e em outro da parte inferior do estado concorrente `Preparando Pedido`: ele poderá estar sendo organizado e com o pagamento autorizado, sendo embalado e com o pagamento sendo efetuado etc. Cada uma das partes separadas pelo segmento de reta tracejado corresponde a uma região (ver Seção 7.4).

Destacamos alguns aspectos importantes da Figura 7.12:

- a separação das regiões é feita por meio de segmento de reta tracejado;
- o emprego do pseudoestado inicial em cada região, significando o ponto de partida em cada uma delas;

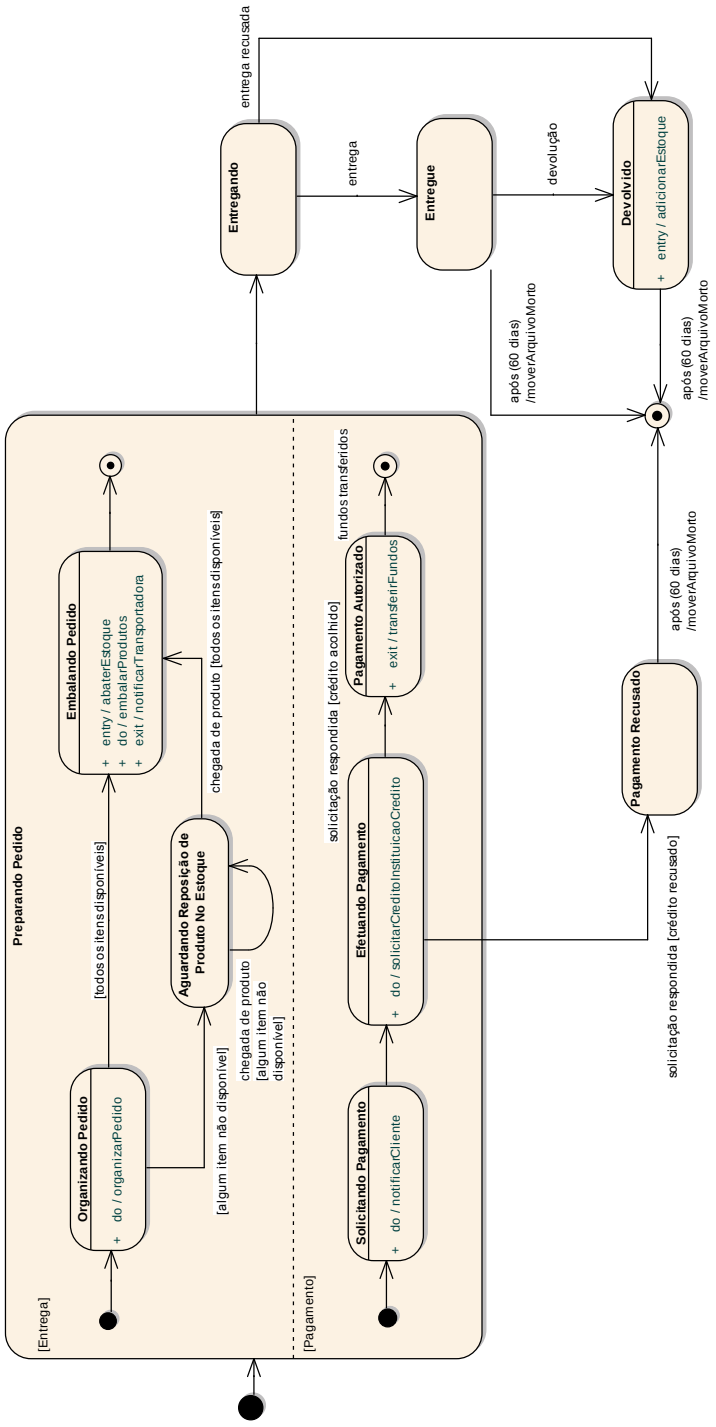


Figura 7.12: Estados concorrentes dos objetos da classe `Pedido` da ZYX.

Tabela 7.1: Atividades e ações em DMEs da UML.

	Podem durar "para sempre"?	**Contexto de execução**	**Pode ser interrompida?**	**Longa ou curta duração?**
Atividade	*Sim*	*Dentro dos estados*	*Sim*	*Longa*
Ação	*Não*	*Durante as transições*	*Não*	*Curta*

- o emprego do "olho de boi" em cada região, especificando que a máquina de estados da região se encerra. Só quando ambas as máquinas de estado se encerram o objeto transita do estado concorrente `Preparando Pedido` para o estado `Entregando`. Este aspecto, por sinal, ilustra bem nossas explicações sobre o uso da notação de "olho de boi" (ver Seção 7.4);
- a saída do estado `Preparando Pedido` para o estado `Pagamento Recusado`, por conta da recusa do pedido de crédito.

7.7 Atividades e ações em DMEs

Vimos que um estado de atividade é um estado em que um objeto se encontra executando uma atividade que termina em algum momento ou permanece executando para sempre. Vimos também que objetos podem executar ações enquanto transitam de estado para estado. Atividades e ações em DMEs são, por definição, coisas que o objeto executa em contextos diferentes; enquanto está em um estado e enquanto transita entre estados, respectivamente. Atividades podem ser interrompidas pela ocorrência de eventos, enquanto ações não podem ser interrompidas, ou seja, quando um objeto inicia uma transição, esta não pode ser interrompida, nem a ação associada a ela. Atividades são, usualmente, de mais longa duração que as ações. Esses dois conceitos são bastante próximos, e suas diferenças são subjetivas. Resumimos na Tabela 7.1 os conceitos relativos a atividades e ações.

> **ATENÇÃO:** As atividades e as ações especificadas nos estados e nas transições dos DMEs são operações que os objetos enfocados realizam. DMEs, portanto, ajudam a compor o compartimento das operações dos diagramas de classes.

7.8 DMEs e Diagramas de Casos de Uso

Diagramas de máquina de estado nos ajudam a descobrir casos de uso. Isso ocorre porque as mudanças de estado dos objetos em um sistema estão, em boa parte das

vezes, associadas a execuções de casos de uso. Por exemplo, no detalhe ilustrado na Figura 7.10 o evento de entrega é gerado com o registro da entrega no sistema, ou seja, em alguma parte de um caso de uso que especificaremos para registro das entregas no sistema. Analogamente, a transição para o estado `Devolvido` estará associada à execução de uma alguma funcionalidade (um caso de uso) para registro das devoluções de pedidos feitas à ZYX.

Portanto, uma boa dica para um rápido exame da completude do diagrama de casos de uso é verificar os eventos que provocam as transições entre estados nos DMEs. Essa técnica, embora ajude a elucidar casos de uso esquecidos, não garante que todos os casos de uso foram relacionados, inclusive porque não desenvolvemos DMEs para todas as classes do conceito.

Curiosidade: Como ilustração do que acabei de expor, com base em uma experiência que vivi no passado, no início da fase de análise de um sistema, da qual participei como arquiteto, um colega de equipe me apresentou um esboço do diagrama de casos de uso que continha cerca de trinta casos de uso. Um pouco mais adiante, tive acesso a um complexo diagrama de máquina de estados de uma entidade do domínio que outro colega havia desenvolvido. Segundo esse diagrama, a entidade modelada possuía cerca de vinte estados, havendo cerca de trinta transições entre eles. Achei estranho haver tantas transições (isso em um só DME) e tão poucos casos de uso. Em um rápido exame das transições, descobrimos mais cerca de vinte casos de uso que não haviam sido identificados no diagrama de casos de uso. Você pode imaginar o impacto que isso causou na expectativa do cliente com relação ao custo do projeto.

Diagramas de máquina de estados são, portanto, grandes aliados na compreensão e especificação corretas do negócio e do sistema que iremos desenvolver. Muitos analistas de sistemas acham, no entanto, que esses diagramas são para serem desenvolvidos na fase de projeto so sistema, somente.

7.9 Resumo do Capítulo

Por vezes, uma determinada classe possui instâncias que atravessam um número significativo de estados. Podemos tentar descrever os estados e as transições usando textos em linguagem coloquial; mas, conforme o número de estados, eventos e transições aumenta e as condições para as transições se tornam mais complexas, o texto possivelmente se tornaria bastante longo, confuso e pleno de ambiguidades. Por isso desenvolvemos DMEs da UML. DMEs especificam não só os estados pelos quais

os objetos podem passar e as possíveis transições entre os estados como também as reações dos objetos aos eventos que os atingem.

Diagramas de máquina de estados são desenvolvidos somente para classes de objetos que possuem comportamento dinâmico relevante, passando por vários estados e respondendo a diversos eventos.

Cada estado em que um objeto se encontra reflete a situação dos atributos e dos relacionamentos desse objeto. Qualquer estado é um estado de espera ou de atingimento de uma situação ou de atividade.

Um objeto em um estado pode executar uma atividade assim que entra no estado, imediatamente antes de sair ou enquanto permanece no estado.

Transições (passagem de um estado a outro) se dão exclusivamente a partir de eventos e dentro de determinadas condições. Eventualmente objetos executam ações enquanto transitam. O trio *evento, condição e ação* (ECA) rotula as transições.

Estados podem conter outros estados (os subestados), concorrentes ou independentes. Objetos não podem estar em mais de um estado ao mesmo tempo, a menos que sejam estados concorrentes.

Um diagrama pode conter mais de um pseudoestado inicial, desde que os pseudoestados estejam em contextos diferentes. Como não deve haver confusão por qual estado uma máquina de estados irá começar, o uso de múltiplos pseudoestados iniciais em um mesmo diagrama deve ser feito com atenção. Estados finais são estados dos quais os objetos não podem sair. "Olhos de boi" também representam estados finais, mas devem ser usados com atenção nesse caso, pois representam também encerramentos de regiões em estados compostos.

Atividades e ações têm pequenas diferenças conceituais, muito subjetivas: atividades são executadas nos estados, são normalmente de longa duração e podem ser interrompidas; ações são executadas durante as transições, são normalmente de curta duração e não podem ser interrompidas. Atividades e ações especificadas nos estados e nas transições são operações que os objetos enfocados realizam e, portanto, devem compor o rol de operações que eles executam.

7.10 Exercícios Propostos

1. Se, porventura, você tivesse as classes `Interruptor` e `Ar Condicionado` em um modelo, para qual ou quais delas você investiria seu tempo desenvolvendo um diagrama de máquina de estados? Justifique sua resposta. Use sua experiência sobre os estados em que os objetos de cada uma dessas classes podem estar.

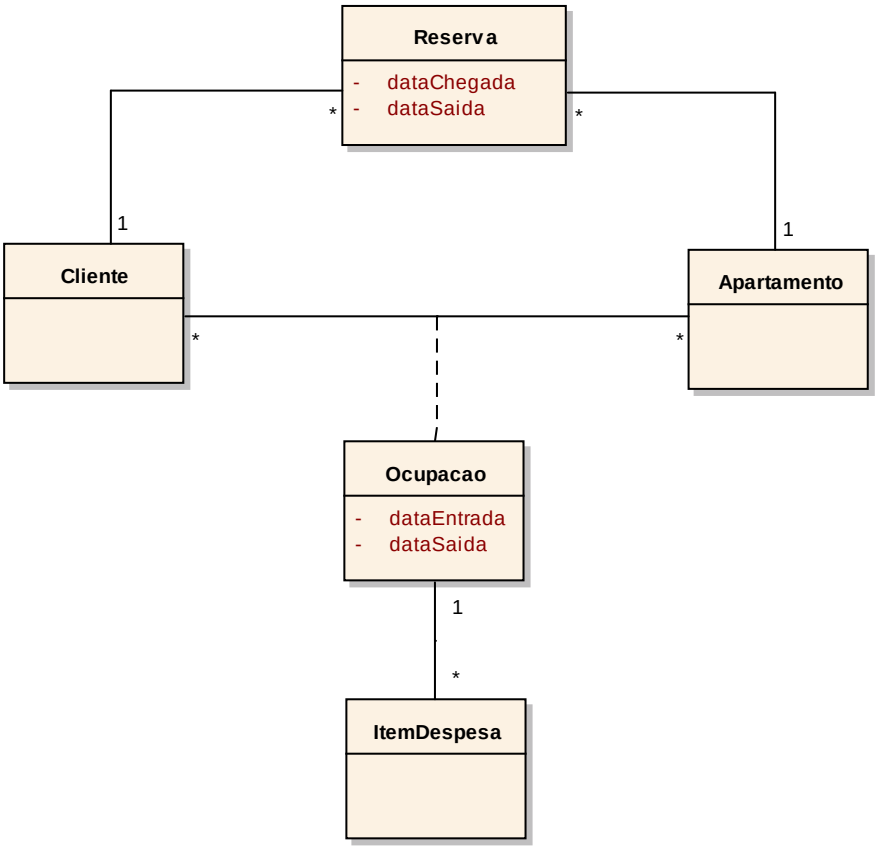


Figura 7.13: Diagrama de classes de conceito do Hotel Cincoestrelas.

2. No contexto de um pequeno hotel, há conceitos como `apartamento`, `reserva`, `cliente`, `ocupação` e `despesa`. O diagrama de classes da Figura 7.13 ilustra como esses conceitos se relacionam.

   Desenvolva o diagrama de máquina de estados para os objetos da classe `Apartamento`, considerando, em um primeiro momento, que um apartamento pode estar reservado, ocupado ou livre. A gerência do hotel entende que um apartamento reservado é aquele que não está ocupado, mas que não está livre, ou seja, está bloqueado no dia, aguardando a chegada do cliente. É importante não confundir com um apartamento que possua reservas para outros dias. Adicione, no local mais apropriado, a ação de impressão da fatura com as despesas do cliente quando ele deixa um apartamento. Use sua experiência pessoal, seu bom senso e o jargão da área hoteleira.

3. Com base no que foi descrito no Exercício 2, considere agora que um apartamento também pode estar interditado para obras, estado para o qual um apartamento pode passar a qualquer momento pela ocorrência de um problema sério e duradouro nas instalações, por exemplo. Sendo assim, a interdição para obras corresponde a uma situação distinta das demais, não se tratando de (pequeno) reparo que pode ser feito rapidamente, durante as ausências dos clientes dos apartamentos. Verifique a conveniência da adoção de superestados e/ou estados concorrentes na sua solução. Refaça o diagrama sa solução do Exercício 2 para refletir essas considerações.

As soluções encontram-se a partir da Página 218.

CAPÍTULO 8

DIAGRAMAS DE ATIVIDADE

> Great things are done by a series
> of small things brought together.
>
> Vincent van Gogh

Quando estudamos descrições de casos de uso, vimos que a especificação da interação entre usuários e sistema segue uma sequência de passos correspondendo a ações dos usuários (preenchendo campos de dados, fazendo opções, pressionando botões etc.) e a respostas (ações) do sistema. Vimos que a elaboração de especificações eficazes não é uma atividade trivial porque muitas vezes temos que lidar com situações complexas (muitas ações e opções oferecidas aos usuários, por exemplo), mantendo a concisão, mas garantindo a completude.

Quando estudamos diagramas de máquina de estados, vimos que há estados em que os objetos executam atividades, como, por exemplo, `organizarPedido`, realizada enquanto um pedido se encontra no estado `Verificando Disponibilidade Estoque` (ver Figura 5.2). Aproveitando o exemplo, se pensar um pouquinho sobre quais são os passos que compõem a atividade `organizarPedido`, possivelmente você vai se surpreender com a quantidade deles e a complexidade como eles se estruturam na atividade. Imagine, agora, a quantidade e a complexidade com que se estruturam as ações necessárias para a construção de um avião ou um navio transatlântico...

Diagramas de atividade (DAs) enfocam o fluxo de controle entre ações que compõem um processo qualquer. Esses diagramas especificam a ordem (no tempo)

de execução das ações, cobrindo, portanto, parte da dimensão temporal do modelo de um sistema; DMEs e diagramas de interação são os outros diagramas da UML que complementam os DAs na cobertura dessa dimensão.

Com os elementos de notação gráfica que incorporaram nos últimos anos para tratamento de paralelismo, partições hierarquizadas e em duas dimensões, pinos etc., os diagramas de atividade da UML vêm sendo usados em inúmeras outras aplicações, além das tradicionais especificações de algoritmos complexos. Daí sua grande relevância em modelagem de sistemas. DAs também são muito úteis para a descrição de processos compostos por ações executadas em paralelo, como tipicamente ocorre em processos de engenharia e de negócios, os chamados fluxos de trabalho – *workflows*, em inglês.

Nas versões da UML anteriores à 2.0, DAs eram associados aos DMEs como variantes destes; DAs poderiam ser entendidos com DMEs onde todos os estados eram estados de atividade. Isso permitia um entendimento rápido dos conceitos e da notação envolvidos em DAs a partir dos conceitos envolvidos em DMEs. Essa pequena variação de um diagrama para outro, no entanto, já era suficiente para acarretar grande variação no emprego dos dois diagramas.

Na versão 2.0 da UML, DAs sofreram inúmeros acréscimos e modificações na notação usada, estendendo significativamente suas expressividades. Nessa versão, mesmo não mais mencionando a associação entre DAs e DMEs, a UML adotou a mesma notação gráfica básica dos DMEs nos DAs. O fato é que o entendimento anterior à UML 2.0 continua sendo possível na maioria dos casos.

Nas seções a seguir mostraremos os elementos gráficos dos diagramas de atividade que são mais usados em modelagem de sistemas em nível conceitual. Faremos uma abordagem sob a ótica da associação com os DMEs, aproveitando, portanto, boa parte dos conceitos tratados no Capítulo 7.

Para ilustrar de forma completa a apresentação dos conceitos e da notação, daremos alguns exemplos no contexto de negócios e, no final do capítulo, empregaremos DAs na especificação gráfica de casos de uso.

8.1 Atividades e Ações em DAs

Segundo a especificação da superestrutura da UML ([11]), "ação é a unidade fundamental para a especificação de um comportamento". Entendemos, daí, que comportamento é um conjunto de ações. Uma ação recebe um conjunto de entradas e as converte em um conjunto de saídas, embora ambos os conjuntos possam ser vazios. O conjunto de entradas para uma ação pode ser resultado das saídas de uma ou mais ações executadas anteriormente. Algumas dessas ações alteram o estado do ambiente

no qual a ação é executada, alterando valores de atributos de objetos desse ambiente.

Ações são contidas em atividades. Estas proveem o contexto, o controle e a estrutura de execução (encadeamento, repetições, decisões, paralelismo) das ações. Um DA é, portanto, formado pelas ações que compõem a atividade que está sendo modelada, estruturadas de alguma forma.

A Figura 8.1 ilustra o modelo UML da atividade `Organizar Pedido`, executada enquanto um objeto da classe `Pedido` se encontra no estado `Verificando Disponibilidade Estoque` (ver DME da Figura 7.1). Os elementos do diagrama da figura estão comentados nos itens anotacionais da UML (textos dentro de retângulos com pontas dobradas ligadas aos elementos comentados por meio das linhas tracejadas).

Na Figura 8.1, o contêiner externo ("retângulo com os vértices arredondados" que contém os demais elementos do modelo) corresponde à atividade. Nesse contêiner estão as ações que compõem a atividade, estruturadas conforme a sequência definida pelos sentidos das setas de transições.

A primeira ação a ser executada é indicada pelo nó inicial (o pseudoestado inicial, segundo a terminologia usada nos DMEs). O símbolo de final de atividade ("olho de boi") indica o ponto onde a atividade se encerra.

Em nossa associação com os DMEs, as ações, como podemos perceber por seus identificadores, são estados de atividade que, quando terminam, provocam as saídas para os estados seguintes, conforme as transições de saída.

8.2 Nós de Decisão (Desvios) e Qualificação de Fluxos

Até aqui falamos em transições, mas elas, nos DAs, se chamam *fluxos*. O conceito é o mesmo e, por isso, elas possuem a mesma notação gráfica das transições nos DMEs: segmentos de reta, poligonais ou curvas com seta aberta em uma das pontas.

Os fluxos podem ser *qualificados* ou *não qualificados*, ou seja, podem possuir ou não rótulos; no caso de possuir, eles especificam somente as condições em que são trilhados[1]. Nesses casos, as condições, também chamadas de guardas, são colocadas obrigatoriamente entre colchetes ("[" e "]"), tal qual nos DMEs. Em outras palavras: um fluxo não qualificado de saída de uma ação significa que ele é trilhado incondicionalmente pela ocorrência do evento de fim da ação da qual ele parte.

Fluxos qualificados partem dos chamados *nós de decisão* (ou *desvios*), representados graficamente como losangos, como está ilustrado na Figura 8.1.

[1] Não são mencionados eventos nem ações, até porque os eventos são os de final de execução e as ações podem estar especificadas como a ação seguinte, no fluxo.

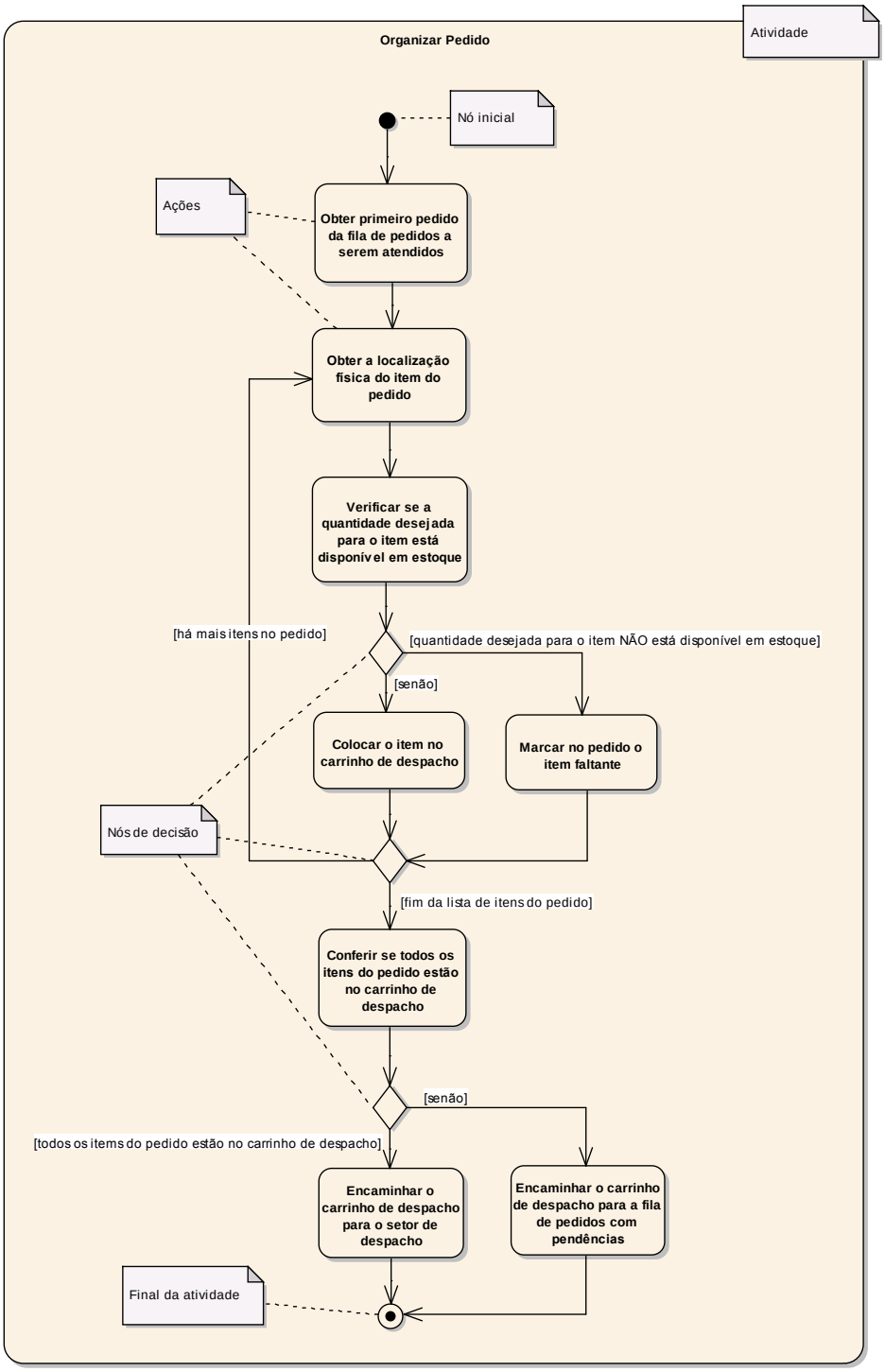


Figura 8.1: DA da atividade `Organizar Pedido`. A atividade do diagrama é executada pelo estoquista da ZYX, nossa empresa fictícia.

Segundo a UML, um nó de decisão é um nó de controle que escolhe entre fluxos de saída, obrigatoriamente guardados com as respectivas condições de saída. Podemos representar quantos fluxos de saída necessitarmos, desde que as guardas especifiquem condições lógicas mutuamente excludentes, ou seja, só é possível sair por um único fluxo de saída de um nó de decisão. Com isso, podemos modelar os *cases* ou *switches* das linguagens de programação. Para facilitar a formulação das expressões, é usual especificar as condições usando expressões lógicas e complementá-las usando "*else*" ou "*senão*".

A contrapartida de um nó de decisão é um nó de *intercalação*. A intercalação recebe os vários fluxos de entrada que correspondem aos desvios e tem um único fluxo de saída, ou seja, uma intercalação define o ponto onde os desvios correspondentes às decisões terminam e o fluxo retorna a um único caminho. Uma intercalação tem que ser colocada sempre que colocamos um nó de decisão, a menos que o fluxo termine ou que outra decisão precise ser tomada. Adotamos o mesmo símbolo de nós de decisão para os nós de intercalação: o losango.

Conforme ilustrado na Figura 8.2 (que é um detalhe da Figura 8.1), após a obtenção, por parte do estoquista, da localização física do item no estoque, é feita a verificação se a quantidade desejada do item está ou não disponível em estoque. Caso a quantidade desejada do item não esteja disponível em estoque, o estoquista marca no pedido o item faltante. Caso a quantidade desejada do item esteja disponível em estoque, o estoquista coloca o item no carrinho de despacho. Logo após uma ou outra dessas alternativas ter sido trilhada, outra decisão é tomada: se há mais itens no pedido (e aí prossegue para mais uma iteração) ou se o pedido é composto de apenas um item.

8.3 Separações e Junções

Outros elementos importantes da notação gráfica usada em DAs são as separações e as junções (*forks* e *joins*, respectivamente, em inglês). Esses elementos dotam os DAs de mecanismos para modelagem de ações que ocorrem concomitantemente, tornando-os suficientemente completos para modelagem de processamento paralelo e processos de negócio.

Separações marcam pontos da atividade em que se iniciam fluxos de ações que ocorrem em paralelo, compreendendo um fluxo de entrada e dois ou mais fluxos de saída. Ao contrário dos nós de decisão, em que a atividade executa um desvio, tomando um caminho ou (exclusive) outro, dependendo das condições avaliadas, nas separações todos os fluxos de saída são trilhados ao mesmo tempo, caracterizando o que chamamos de múltiplos fios de execução (*multithreading* em inglês).

A Figura 8.3 especifica o processo de tratamento do pedido na ZYX, imaginando

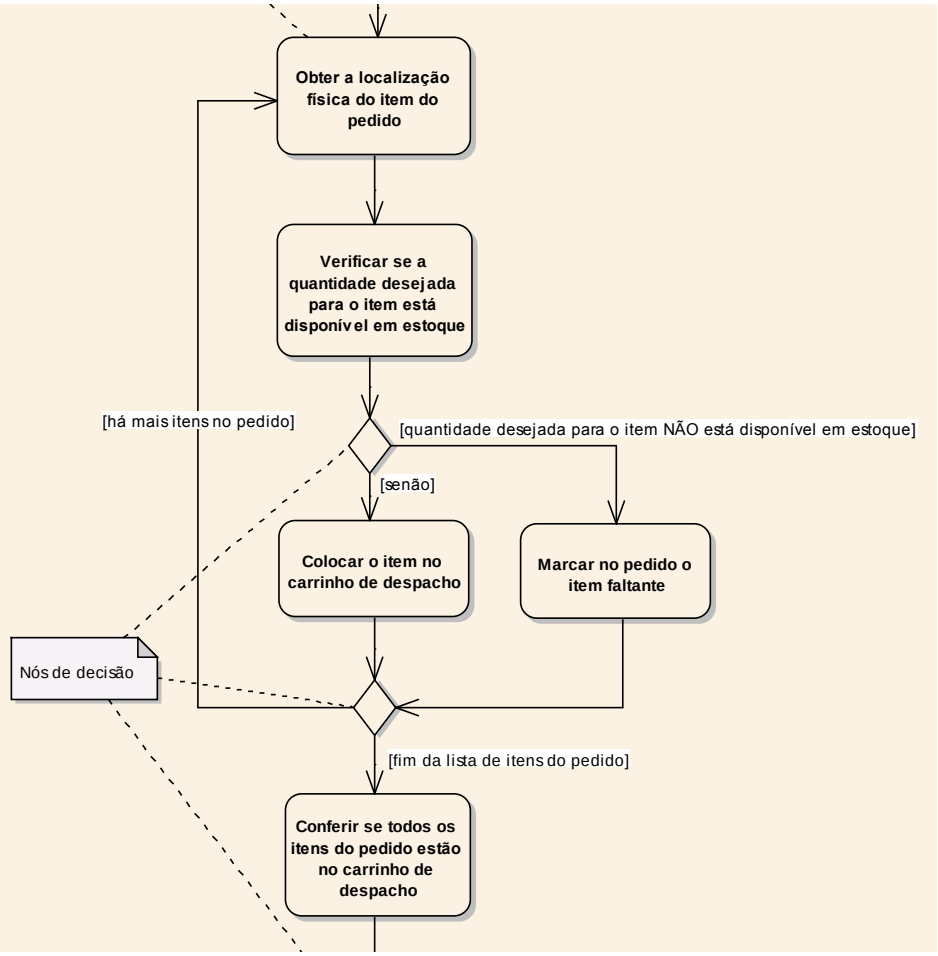


Figura 8.2: Nós de decisão em diagramas de atividade (um detalhe da Figura 8.1).

que três setores estão envolvidos no seu processamento: o setor financeiro, para tratar das questões relativas ao pagamento pelo pedido, o setor de atendimento, que trata da recepção do pedido e da elaboração da fatura, e o setor de processamento, que executa as atividades referentes à separação dos itens, embalagem e despacho dos pedidos.

Segundo o diagrama da Figura 8.3, a partir do momento em que o setor de atendimento da ZYX recebe o pedido de um cliente, ele notifica o setor financeiro para que tome as providências para a cobrança do pedido e encaminha uma cópia do pedido para o setor de processamento, para que seus itens sejam separados e embalados para entrega. Concomitantemente, a fatura deve ser preparada pelo atendimento, pois deverá ser anexada ao pedido.

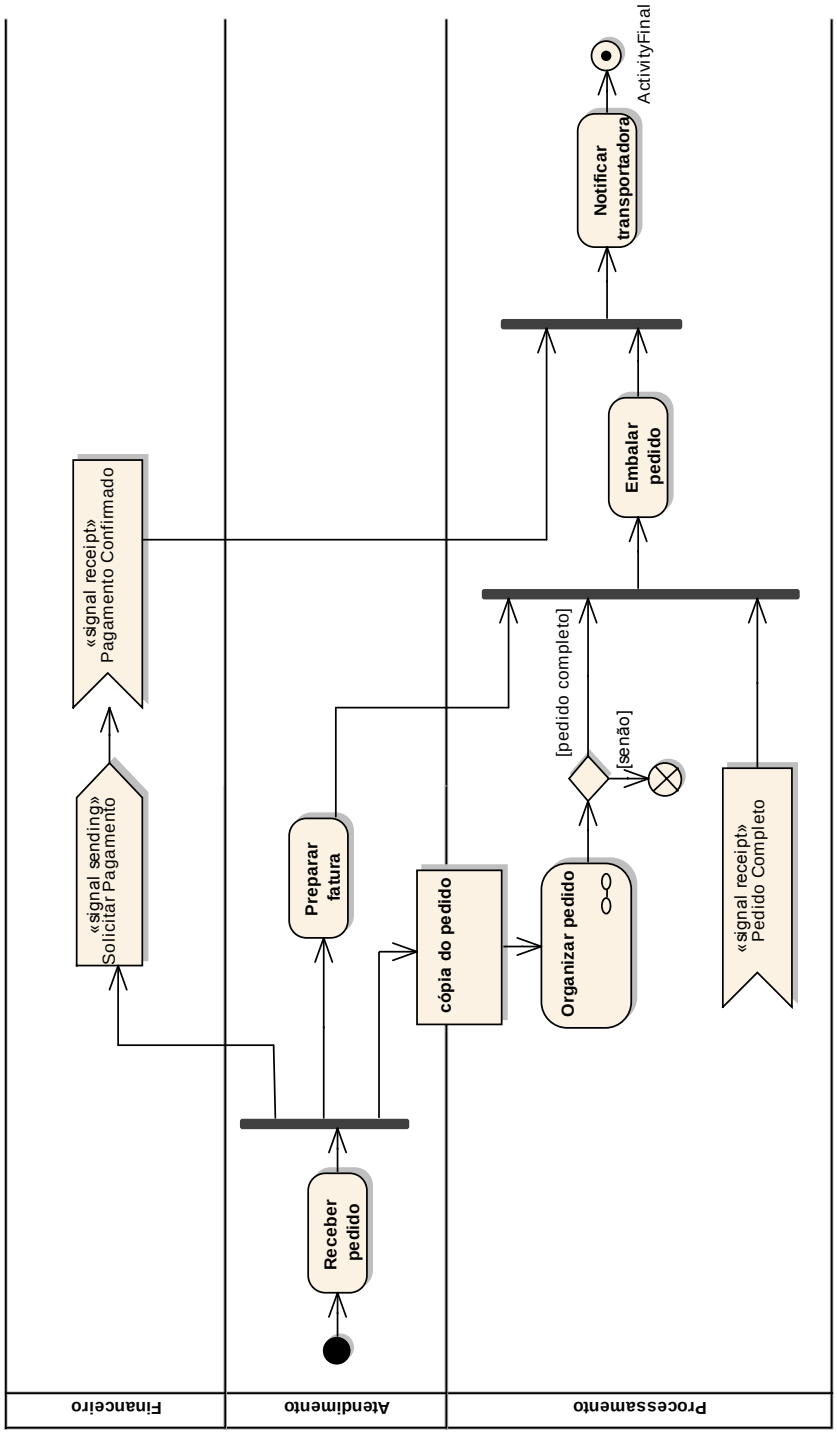


Figura 8.3: O processamento de pedidos na empresa fictícia ZYX.

Os três setores da ZYX trabalham, a partir do recebimento do pedido, de forma independente, executando as demais ações para o despacho do pedido. Essa "separação" de responsabilidades, independência ou autonomia é denotada pela barra grossa vertical que se assemelha a um garfo (daí o nome de *fork*, que significa garfo, em inglês), com um fluxo de entrada (o que vem da ação Receber Pedido), e três fluxos de saída.

As contrapartidas das separações são as junções, que marcam pontos de sincronismo entre os diversos fios de execução criados pelas separações. O processamento só passa da junção quando todos os fios de execução que nela convergem são terminados, como se a junção fosse um "ponto de encontro".

Na Figura 8.3, `Embalar Pedido` só ocorre após a fatura ter sido preparada e o pedido ter sido organizado (os itens terem sido separados no carrinho de entrega). Isso pode ser devido, por exemplo, à necessidade de a fatura ser embalada junto com os itens que compõem o pedido. De forma análoga, a transportadora só pode ser notificada (ação `Notificar Transportadora`) de que uma entrega deve ser feita se o pedido tiver sido embalado e o pagamento tiver sido informado à ZYX.

Junções não são obrigatórias após uma separação. Há fios de execução que podem permanecer em execução "para sempre", sem a necessidade de sincronismo com qualquer outro fluxo. Os fluxos também podem se encerrar antes que a junção ocorra, mas deixamos para tratar disso em detalhes quanto falarmos de fins de fluxos, mais adiante, ainda neste capítulo.

Múltiplos fluxos de entrada em uma ação caracterizam o que a UML chama de *junção implícita*; ou seja, quando vários fluxos convergem diretamente em uma ação, isso significa que a ação só é executada quanto terminam todas as ações que originaram esses fluxos.

8.4 Partições

Partições são usadas quando há necessidade de indicar quem executa as ações; as ações colocadas em uma determinada partição são executadas pelo ator/setor indicado na partição. A Figura 8.3 ilustra as partições correspondentes aos três setores da ZYX que participam do processamento de pedidos: Processamento, Atendimento e Financeiro. Partições também são chamadas *raias de natação*.

Partições podem ser hierarquizadas (veja a Figura 8.4) e podem representar mais de uma dimensão (veja Figura 8.5).

A Figura 8.4 especifica que os subsetores de `Estoque` e de `Despacho` pertencem ao setor de `Processamento`. A Figura 8.5 especifica que os subsetores `B1` e `B2`

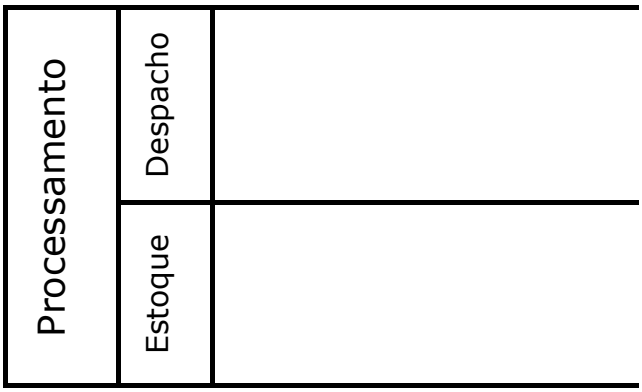


Figura 8.4: Partições hierarquizadas.

pertencem ao setor `B` e os subsetores `A1` e `A2` pertencem ao setor `A`. Além disso, uma ação colocada na "célula" `A1-B1`, por exemplo, especifica que ela é executada de forma colaborativa pelos setores `A1` e `B1`.

8.5 Fluxos de Objetos

Durante a execução das ações em uma atividade é comum que objetos sejam passados entre ações. Por vezes precisamos representar esse tráfego de objetos. Por exemplo: na Figura 8.3, queremos especificar que o setor de atendimento passa ao setor de processamento uma cópia do pedido para que o encarregado separe os itens que compõem o pedido, colocando-os no carrinho de entrega. Objetos são denotados por retângulos que se situam no fluxo entre as ações que os passam e as que os recebem. O nome do objeto deve ser colocado no interior do retângulo na forma `objeto:classe`, em que a referência à instância ou à classe (uma ou outra) é opcional. Por exemplo, as formas a seguir são aceitáveis: `copiaFatura:Fatura`, `copiaFatura` e `:Fatura`.

8.6 Atividades Aninhadas

Além de ações, as atividades podem conter, recursivamente, outras atividades. É possível representar graficamente uma atividade aninhada (a que está dentro de outra)

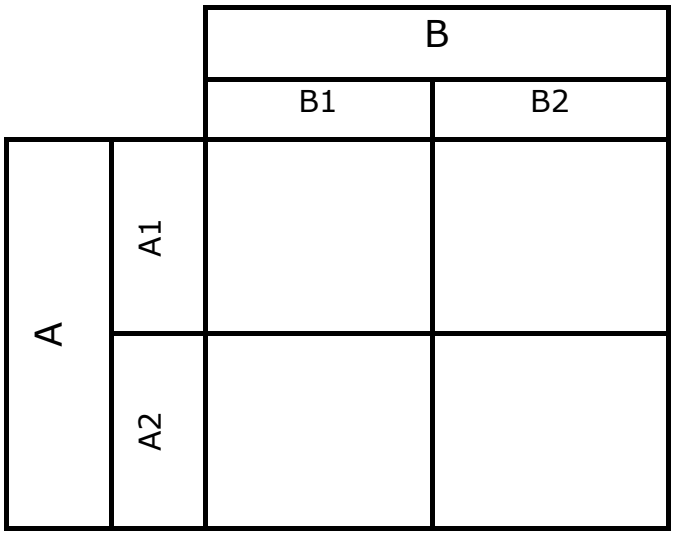


Figura 8.5: Partições hierarquizadas e em duas dimensões.

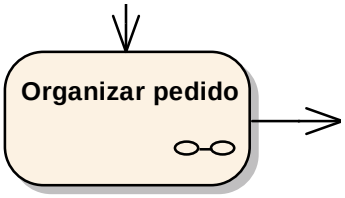


Figura 8.6: Atividade aninhada em outra (um detalhe da Figura 8.3).

como uma ação em um diagrama, marcando-a com um pequeno símbolo no canto inferior direito (ver `Organizar Pedido`, na Figura 8.3, que é detalhada na Figura 8.6).

As atividades aninhadas são detalhadas em termos de suas ações e/ou atividades em outro diagrama do modelo.

ATENÇÃO: O símbolo oficial da UML para atividade aninhada é um pequeno tridente no canto inferior direito, não mais as duas caixas conectadas da Figura 8.6, usadas nas versões anteriores à 2.0. Os CASEs, como é o caso do que usei para elaborar as ilustrações, demoram algum tempo para se adaptar às mudanças da

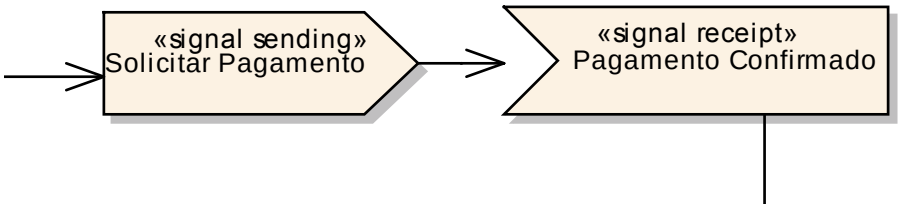


Figura 8.7: Envio e recepção de sinais (um detalhe da Figura 8.3).

notação que ocorrem entre versões da linguagem.

8.7 Sinais e Eventos Temporais

A UML dispõe de notação gráfica para a representação de envio e de recepção de sinais. Segundo a Figura 8.7 (que também é um detalhe da Figura 8.3), o setor financeiro solicita o pagamento (enviando um sinal, que pode representar uma ligação telefônica ou um e-mail) ao cliente da ZYX e aguarda a confirmação desse pagamento, possivelmente por parte da operadora do cartão de crédito, para que possa dar prosseguimento ao processamento do pedido. O envio e a recepção de sinais são ações especiais rotuladas, respectivamente, com as palavras-chave `«signal sending»` e `«signal receipt»`.

O envio de um sinal em um modelo de sistema pode estar associado ao lançamento de uma interrupção, e a recepção pode estar associada à execução da ação de tratamento dessa interrupção.

Um evento temporal é algo que ocorre em um instante determinado ou após determinado tempo medido com relação à ocorrência de outro evento (ver Seção 7.5). Um evento temporal pode dar origem a um fluxo. Na UML, eventos temporais possuem a notação gráfica ilustrada na Figura 8.8, que especifica a situação em que a ação de fechamento do mês contábil é iniciada pela ocorrência do (evento de) fim de mês.

8.8 Fim de Fluxo ou Cancelamento

Após uma separação, quando um fluxo de entrada dá origem a dois ou mais fluxos de saída (fios de execução ou *threads*, em inglês), pode ser necessário terminar um ou

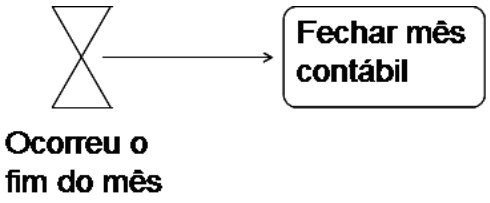


Figura 8.8: Representação de eventos temporais na UML.

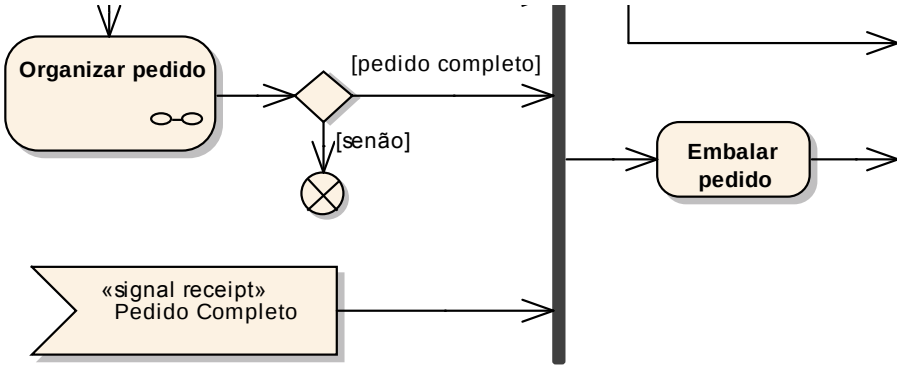


Figura 8.9: Exemplo de fim de fluxo e recepção de sinal (detalhes da Figura 8.3.

mais desses fluxos de saída antes que ocorra uma junção (que, já vimos, nem sempre é necessária), sem que isso provoque o fim de toda a atividade. Esse término de um fluxo (também chamado de *cancelamento*) é representado por um círculo com um "X" inscrito nele. A Figura 8.9 ilustra o trecho da Figura 8.3 que representa o fim do fluxo, caso um pedido seja constatado como incompleto após a sua organização. O trecho do DA ilustrado na Figura 8.9 é particularmente interessante porque especifica que a embalagem do pedido só ocorre após a preparação da fatura e do pedido estar completa, algo que pode ser verificado após a organização ou sinalizado de alguma forma um tempo depois.

Outro exemplo de cancelamento é ilustrado na Figura 8.10. No caso, as ações `B`, `C` e `D` são executadas em paralelo. Ao final da ação `B`, se a condição `c1` é verdadeira, também os finais das ações `C` e `D` são aguardados na junção para que a ação `E` seja iniciada. Se a condição `c2` é verdadeira, o fluxo de saída de `B` é cancelado e `E` é iniciada somente após os finais das ações `C` e `D`.

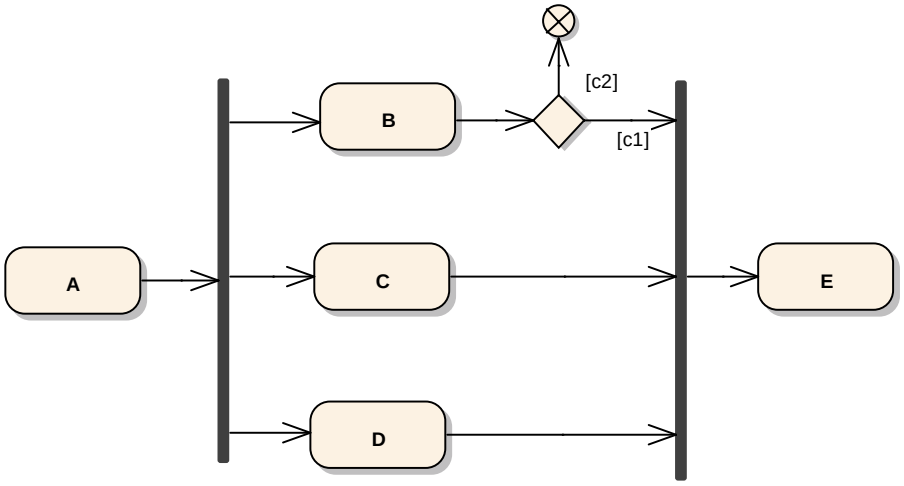


Figura 8.10: Outro exemplo de fim de fluxo.

8.9 Conectores

Os *conectores* são usados para evitar longas setas de fluxos e em quebras de páginas. As Figuras 8.11a e 8.11b representam as mesmas situações, ilustrando suficientemente bem a utilidade de conectores.

8.10 Pinos, Transformações e Regiões de Expansão

Uma ação pode receber parâmetros (objetos, valores, listas de objetos ou de valores etc.) da ação anterior e pode passar parâmetros para a ação seguinte na sequência. Caso seja importante especificar esses parâmetros, usamos a notação de *pinos* da UML.

Um pino é denotado por um pequeno retângulo junto à caixa de ação, rotulado com o nome do parâmetro que representa. Por exemplo, caso um pedido seja preenchido durante o recebimento de uma fatura e seja necessário passá-lo para que a rotina de recebimento do pagamento seja executada, podemos usar a notação de pinos, como ilustra a Figura 8.12a, ou a notação equivalente, na Figura 8.12b, que ilustra o objeto Pedido como parte do fluxo entre ações.

Há situações em que o parâmetro de entrada de uma ação deve ser o resultado

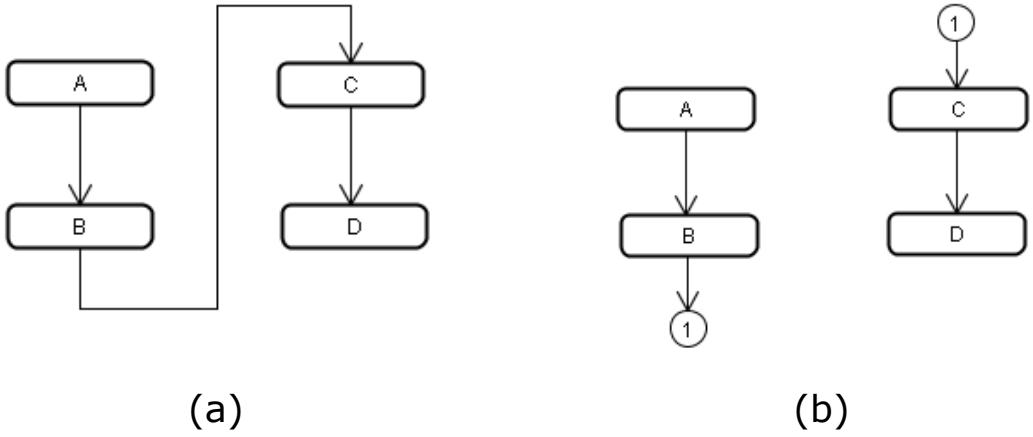


Figura 8.11: Modelos com significados idênticos, sem conectores (a) e usando conectores (b).

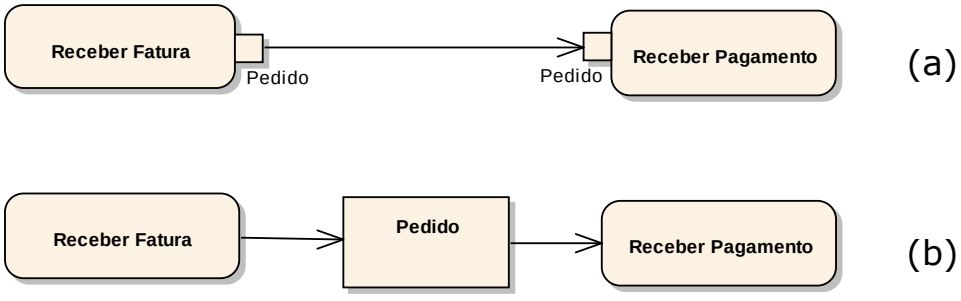


Figura 8.12: Pinos como forma de passagem de parâmetros entre ações (a) e a notação equivalente de fluxo de objetos (b).

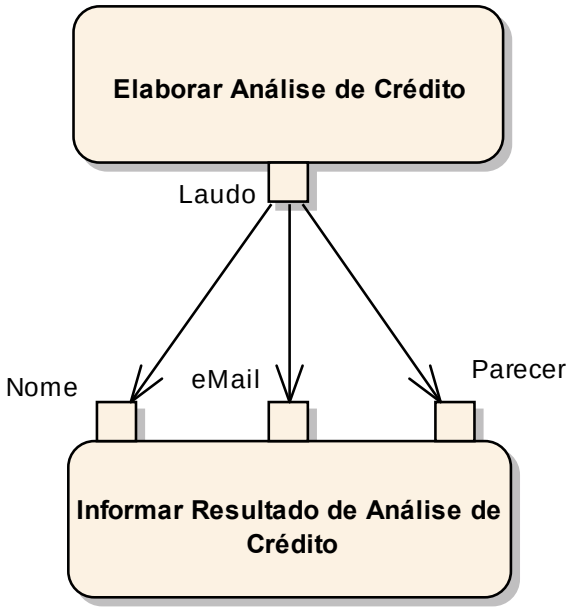


Figura 8.13: Transformação aplicada sobre parâmetro de saída de uma ação.

de uma operação a ser aplicada no parâmetro que sai da ação anterior na sequência. Nesse caso, aplica-se sobre o fluxo que parte de uma ação para a outra o que a UML chama de *transformação*, representada no diagrama por dois ou mais fluxos distintos partindo do pino de saída da ação anterior na sequência.

As transformações devem ser livres de efeitos colaterais; ou seja, quando aplicadas, não podem alterar o estado do ambiente, mudando valor de qualquer atributo de qualquer objeto. Transformações são, por exemplo, operações de consulta aplicadas sobre os parâmetros, como seleções de ocorrências em uma lista, escolha de campos específicos de um formulário etc.

A Figura 8.13 ilustra a situação em que um laudo é produzido por uma atividade de análise de crédito, mas apenas o nome, o endereço de e-mail e o parecer, que são campos selecionados do laudo, são os parâmetros necessários para a atividade de informação do resultado da análise ao solicitante do empréstimo.

Há ainda situações em que o parâmetro de saída de uma ação é composto por uma lista de itens que iniciam múltiplos fios de execução de um conjunto de ações. Esse conjunto de ações passa a ser executado concorrentemente para cada item na lista, caracterizando o que se chama de *região de expansão*. A Figura 8.14 ilustra a seguinte situação: a ação `Aplicar Provas` produz uma pilha de provas de alunos a

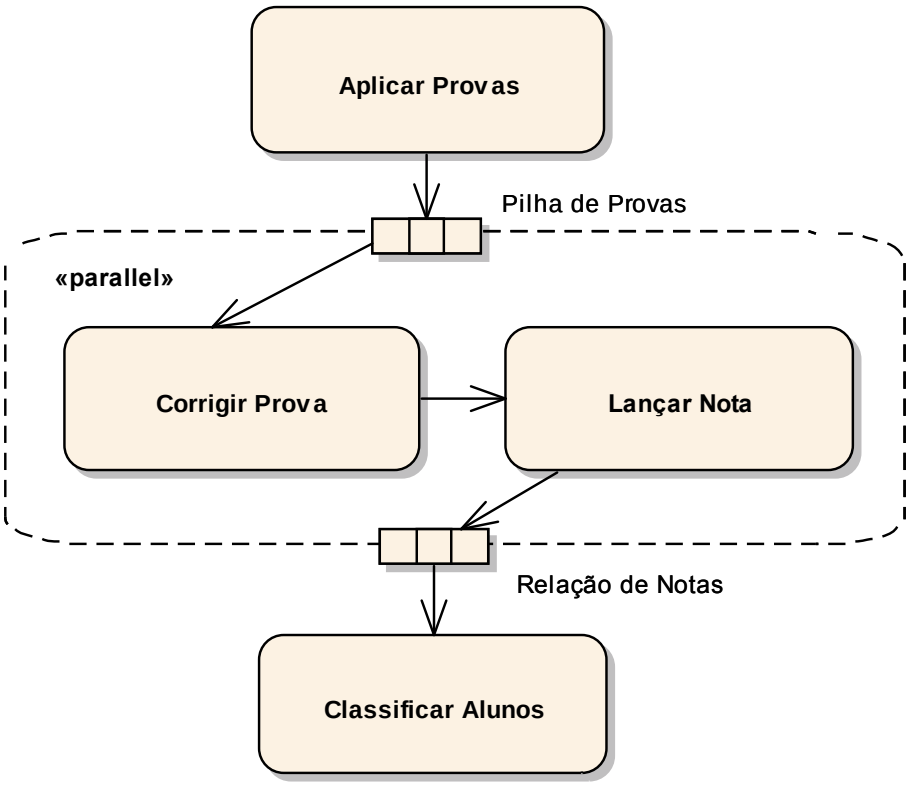


Figura 8.14: Região de expansão para execução de sequência de ações em paralelo.

serem corrigidas por um *pool* de professores. Em seguida, cada professor desse *pool* obtém a próxima prova da pilha, corrigindo e lançando a nota. A sequência de ações `Corrigir Prova` e `Lançar Nota` é executada em paralelo pelos professores. Só após as correções e o lançamento de notas de todas as provas é que a ação `Classificar Alunos` é executada. A região tracejada que contém as ações de correção e de atribuição de notas é a região de expansão. Os dois conjuntos de pinos de entrada e de saída dessa região representam conjuntos ou listas (a pilha de provas e a lista de notas, no caso).

8.11 Especificando Graficamente Casos de Uso com DAs

Vimos no Capítulo 4 que os diversos passos que compreendem a interação usuário-sistema em um caso de uso devem ser especificados de forma padronizada. A forma

lá descrita é bem aceita porque não requer, por parte do cliente que homologa a especificação, habilidades especiais além do conhecimento a respeito do negócio. No entanto, ela é sujeita a erros por algumas razões:

- não é concisa. Embora o analista precise manter o foco na concisão, isso muitas vezes não é possível ou facilmente obtenível. Uma especificação completa quase invariavelmente é enfadonha para quem a formula e para quem a lê. A busca pela concisão pode provocar falta de completude;
- dificilmente é completa. Não é só a busca pela concisão que pode ser a causa de incompletude; muitas vezes acontecem esquecimentos de certos "caminhos" dentro dos fluxos, pela dificuldade de ter uma visão mais em alto nível quando estamos formulando a especificação textual;
- pouco manutenível. As alterações nos fluxos são penosas, seja pela extensão que quase invariavelmente possuem, seja pela não disponibilidade de uma ferramenta adequada, já que frequentemente são usados documentos padronizados do MS-Word com tabelas e links estáticos. Assim sendo, alterações podem facilmente gerar inconsistências na documentação.

As especificações feitas de forma textual precisam ainda ser interpretadas e convertidas em diagramas de sequência pelos projetistas (essa tarefa se chama realização dos casos de uso) na fase de elaboração ou em linguagem de programação pelos programadores na fase de construção do sistema, quando se usa um processo ágil. É nesse ponto que a incompletude da especificação é evidenciada, demandando o retrabalho de especificação dos casos de uso e, quase sempre, novas entrevistas com os usuários.

Felizmente, complementarmente às descrições textuais ou alternativamente a elas, podemos elaborar a especificação dos casos de uso utilizando os diagramas de atividade da UML. As vantagens de usarmos os DAs para as especificações de casos de uso anulam parcialmente as desvantagens das especificações textuais. São elas:

- a concisão. DAs, como os demais modelos UML, produzem especificações concisas;
- a facilitação da busca pela completude. Isso se dá pelo fato de identificarmos facilmente, de modo visual, os pontos de desvios (as idas para fluxos alternativos) e de podermos implementá-los também com facilidade no modelo com os recursos de edição de que a ferramenta CASE dispõe;
- a manutenibilidade. As alterações nos fluxos são facilmente implementadas no modelo gráfico com a ajuda da ferramenta CASE, conforme já mencionamos.

Quando estamos tratando de modelos de sistemas, outra vantagem decorrente do uso dos DAs na especificação de casos de uso é que eles podem facilitar o trabalho dos programadores, pois a transformação do modelo em código representa uma transição bem mais suave e direta do que a transformação de uma especificação em linguagem coloquial, em código.

A necessidade do cliente de entender a notação gráfica para poder homologar a especificação é, no entanto, um óbice. Por isso, eu particularmente acho razoável que o analista inicie a especificação dos casos de uso com DAs e, com bases nesses DAs, elabore a especificação textual para homologação pelo cliente. Essa é a forma que, no meu entender, colabora muito para que se obtenham descrições textuais mais completas.

Existem basicamente duas maneiras diferentes de descrever casos de uso com o emprego de DAs: especificando unicamente as ações do sistema ou incluindo também as ações dos atores na especificação. Nesse caso, usualmente trabalhamos com partições para separar o que cada um faz; uma partição para as ações do sistema e uma para as ações de cada ator. O diagrama de atividade da Figura 8.15 descreve graficamente o caso de uso `Relés e Seus Estados`, descrito de forma textual na Tabela 4.4. Neste caso, apenas as ações do sistema correspondem a ações no DA.

Você pode notar na Figura 8.15, que a chamada ao caso de uso `Acionar Relés` do passo 7 do CT na Tabela 4.4 corresponde à atividade aninhada `Acionar Relés`. Pode notar também que representamos a decisão do usuário de pressionar o botão `Sair`, terminando a atividade, como um fluxo guardado com a condição "usuário pressiona o botão Sair. Fica também como sugestão a forma como representamos graficamente as repetições "para todos", tratando cada elemento da lista em um *loop* (laço).

O diagrama de atividade da Figura 8.16 descreve graficamente o mesmo caso de uso, porém usando partições. Nessa segunda forma, as ações do sistema são colocadas em uma partição, e as ações do supervisor de segurança são colocadas em outra partição.

Você pode notar na Figura 8.16 dois símbolos de final de atividade. Ao contrário do símbolo de início de atividade, que deve ser único, podemos ter quantos símbolos de final de atividade quisermos colocar no diagrama para torná-lo visualmente mais simples. Isso evita longos fluxos no diagrama.

É importante ressaltar que essas duas formas são meramente sugestões. Há outras formas, como, por exemplo:

- pintar as ações do sistema de uma cor e de cores diferentes as ações de cada ator, ao invés de usar partições;

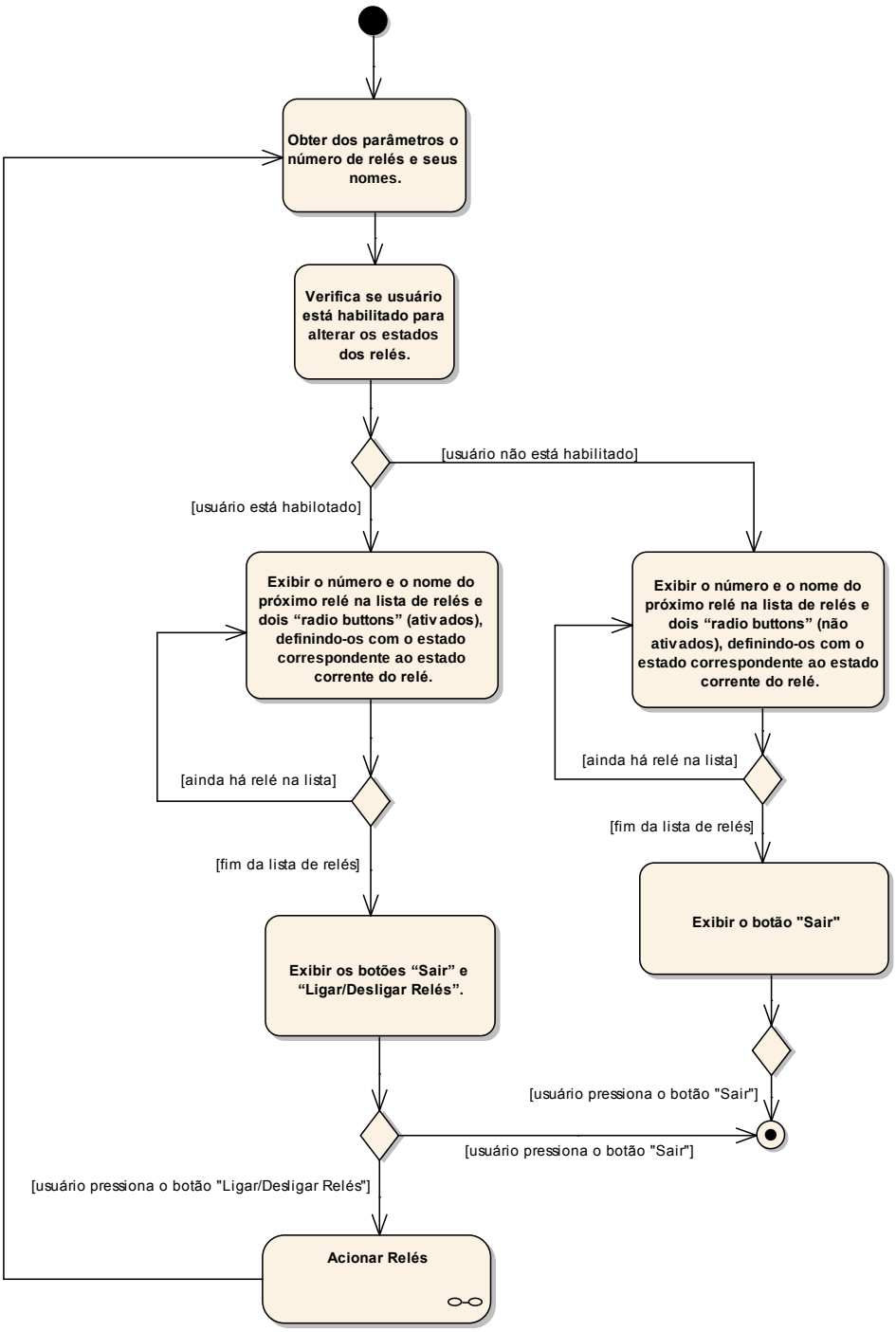


Figura 8.15: Especificação de caso de uso apenas com ações correspondendo a ações do sistema.

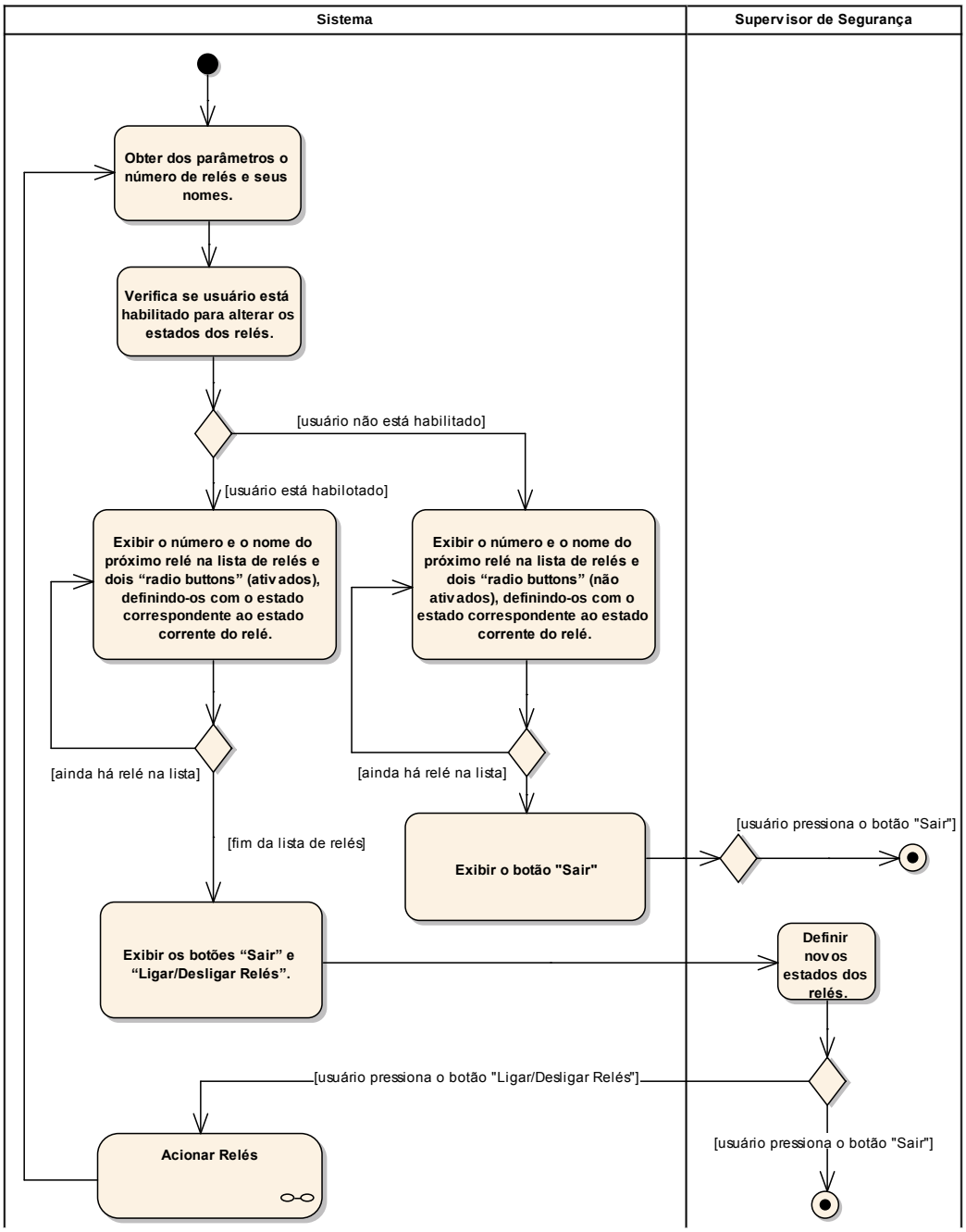


Figura 8.16: Especificação de caso de uso com ações do sistema e do usuário em partições distintas.

- representar somente as ações do sistema, porém rotulando os fluxos com eventos correspondendo às ações dos atores. Essa forma não é correta segundo a UML[2] e, por isso, não é possível adotar essa forma usando alguns CASEs. A despeito disso, essa forma é uma alternativa bastante interessante.

Tendo a capacidade de modelar processos em diversos níveis conceituais, os diagramas de atividade são um poderoso aliado de analistas de negócio, de analistas de sistemas e de projetistas, tornando-se um dos principais diagramas da UML.

8.12 Resumo do Capítulo

Diagramas de atividade – DAs – enfocam o fluxo de controle entre ações que compõem um processo qualquer, especificando a ordem de execução das ações.

Contando com os elementos de notação gráfica que incorporaram nos últimos anos, os DAs vêm sendo usados em inúmeras aplicações, dentro e fora da área de desenvolvimento de sistemas, tais como especificação de algoritmos complexos, modelagem de processos de negócio e *workflows*. DAs têm particular utilidade na especificação de casos de uso, pois as ações podem ser mais facilmente modeladas e os caminhos entre o início do processamento e os possíveis fins (os cenários) podem ser identificados visualmente e, portanto, com mais facilidade. Além disso, podemos contar com as ferramentas CASE para rápida e interativamente esboçar possíveis alternativas de realização da interação entre usuário e sistema.

Um DA modela uma atividade, que contém ações e/ou outras atividades. A ordem de execução dessas atividades é definida pelos fluxos, que apontam a passagem de ação em ação. A primeira ação a ser executada na atividade é indicada pelo nó inicial. Um ou mais nós finais indicam os finais da atividade. Sinais e eventos temporais também podem dar origem a ações em uma atividade.

Durante a execução, desvios no fluxo geral de execução podem ser feitos com base em expressões lógicas avaliadas nos nós de decisão em tempo de execução da atividade. Caso um fluxo precise dar origem a mais de um fluxo, gerando paralelismo (várias ações sendo executadas ao mesmo tempo), é usado um símbolo de separação. Caso diferentes fluxos precisem ser sincronizados (tornar-se, de novo, um só fluxo), um símbolo de junção é inserido no diagrama para representar o "ponto de encontro" dos fluxos.

Um importante elemento da notação gráfica dos DAs é a partição, que pode ser em uma ou em duas dimensões. Partições possibilitam que representemos

[2] Fluxos só podem ser rotulados com guardas, se eles partem de nós de decisão.

responsabilidades na execução de ações, bastando colocar na partição do executor as atividades pelas quais ele é responsável.

Objetos podem passar de ação em ação, representando artefatos produzidos ou consumidos por elas.

Fluxos originados em uma separação podem ser cancelados (terminados), mantendo os demais fluxos ativos ou seja, sem que a atividade seja encerrada.

Parâmetros passados de ação em ação podem ser representados por pinos. Um pino pode dar origem a mais de um fluxo, desde que seja para a mesma ação de destino, caracterizando uma transformação, ou seja, o parâmetro que o pino representa pode dar origem a outro, por uma transformação que não pode alterar o contexto (um filtro, por exemplo).

Diagramas de atividade são particularmente úteis para especificar os passos dos casos de uso, pois eles nos "forçam" a considerar todos os possíveis desvios ao longo da interação usuário-sistema e nos permitem representá-los facilmente com o uso do CASE.

8.13 Exercícios Propostos

1. Elabore um diagrama de atividade com suas ações e decisões tomadas em um dia de semana típico, do momento em que você desliga o despertador até sua chegada no trabalho ou escola. Considere a possibilidade de executar ações em paralelo.

2. Refaça o início do DA do Exercício 1, considerando a solução que demos para ele até a hora do banho, por simplicidade. Considere ainda que o despertador funciona como um gerador de eventos (os alarmes) que precisam ser "tratados" por você. Considere também a possibilidade de você usar a função "soneca" para dormir mais um pouquinho até não haver mais tempo para isso.

3. Descreva, usando DAs, o caso de uso `Registrar Compra` em um sistema para um supermercado hipotético, do qual participa o caixa registrando a compra e, eventualmente, o `Cliente`, quando o pagamento é feito por débito ou crédito no cartão e ele precisa informar a senha, além do supervisor de venda, quando é necessário retirar um ou mais itens da lista ou reimprimi-la. Use sua vivência para estabelecer os passos que compõem a descrição, mas não se esqueça de considerar as situações em que:

    - tudo dá certo;

- você não tem o dinheiro suficiente para pagar por toda a compra, podendo perceber isso durante o registro ou no final dele;
- a fita de papel da registradora acaba no meio da compra e o supervisor precisa intervir com seus "superpoderes" para comandar a reimpressão da lista desde o início;
- você discorda do preço de um item que estava em oferta e pede ao caixa que retire o item da lista. Nesse caso, o supervisor também precisa intervir;
- o código de barras não pôde ser lido pela leitora ótica e o caixa o informa pelo teclado;
- o código do item não consta do cadastro;
- você paga em cartão com chip (no débito ou no crédito) ou em dinheiro, o que é bem menos frequente naquele supermercado.

A descrição parcial na forma textual do caso de uso se encontra feita como resposta do Exercício 3 do Capítulo 4. Use-a como base.

As soluções encontram-se a partir da Página 221.

CAPÍTULO 9

DIAGRAMAS DE SEQUÊNCIA: CONCEITOS BÁSICOS

> Unity is strength... when there is teamwork and collaboration, wonderful things can be achieved.
>
> Mattie Stepanek

Em uma organização, os processos são tipicamente executados por indivíduos especializados, que exercem suas responsabilidades e atuam de forma colaborativa, trocando informações (conversas telefônicas e na copa, no cafezinho, e-mails, sinais visuais etc.) e objetos tangíveis (documentação impressa, informação em CD etc.). Essa comunicação segue uma sequência estabelecida na cultura da organização, cultura essa que é idealmente resultante de observações e análises feitas por seus gestores e por especialistas e analistas do negócio; é documentada em manuais de procedimentos da organização.

Para ilustrar, quando é hora de fechar o balanço da ZYX, alocamos contadores (instâncias da classe `Contador`) na realização dessa tarefa. Eles têm responsabilidades e, para realizá-las, executam um conjunto de operações (somar, diminuir, multiplicar, dividir, analisar lançamentos, selecionar contas etc.). Durante a tarefa, trocam mensagens e artefatos entre si e, quase invariavelmente, delegam tarefas a outros

membros da equipe, contando com a colaboração de auxiliares de contabilidade, de auxiliares administrativos e de mensageiros, dentre outros, todos instâncias das respectivas classes. Eles executam as tarefas seguindo uma sequência predefinida, programada, de execução de operações e de passagem de mensagens.

Analogamente, os programas de computador desenvolvidos segundo o paradigma de orientação a objetos são concebidos imaginando que existam objetos de classes diferentes que são instanciados na memória do computador e que executam código específico, ou seja, cada classe de objetos executa seu conjunto específico de funções. Os objetos invocam funções de outros objetos utilizando mensagens que enviam a eles segundo sequências predefinidas (os programas) criadas pelos programadores de sistemas. As chamadas às funções de outros objetos compõem o mecanismo de troca de mensagens entre objetos. Portanto, tal qual em um ambiente de trabalho, os objetos em um sistema computacional interagem e colaboram, na sequência programada, para a realização das funções do sistema.

Nessa breve introdução tratamos do que chamamos de *colaboração* entre agentes em um sistema – e uma organização pode ser vista como tal – que interagem entre si para a realização dos propósitos desse sistema. Cabe-nos definir como as colaborações podem ser especificadas. A UML dispõe de diagramas para isso. Um deles é o Diagrama de Sequência, de que trataremos neste e no próximo capítulo.

9.1 Especificando Interações por Meio de Diagramas

Para deixar bem claros os conceitos de interação e colaboração entre objetos em um sistema e para introduzir uma forma de especificação dessa interação, exploramos um pouco mais a comparação entre pessoas em um escritório e objetos em um sistema.

A Tabela 9.1 apresenta as correlações que estabelecemos entre os principais conceitos mencionados nos dois últimos parágrafos da seção anterior.

Desconsiderado o aspecto pejorativo da associação de pessoas em uma organização a objetos em um sistema, a Tabela 9.1 indica que a experiência dos administradores de empresas pode ser aplicada por analistas e projetistas de sistemas no desenvolvimento de sistemas elaborados segundo o paradigma da orientação a objetos e vice-versa.

Os diagramas de sequência (DSs) permitem especificar colaborações, descrevendo a sequência de mensagens passadas de objeto a objeto necessária para a realização de um determinado procedimento, seja ele um processo de negócio ou uma funcionalidade em um sistema.

Juntamente com os diagramas de comunicação e de visão geral da interação,

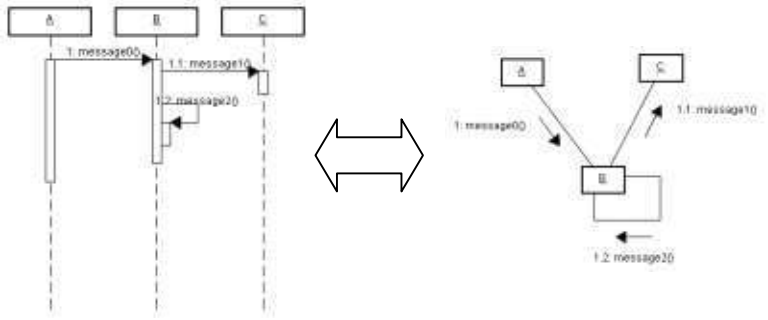
Tabela 9.1: Correlação entre os principais conceitos de processos de negócios e orientação a objetos.

Processos de Negócio	**Programas OO**
Indivíduos especializados	Objetos
Nome de um indivíduo Profissões/especialidades dos indivíduos	Identificador de um objeto Classes de objetos
Trocas de informações e sinais	Chamadas de funções
Processos de negócio	Rotinas, programas
Alocação de um profissional de uma profissão específica em uma tarefa	*Instanciação* (criação) de um objeto de uma classe específica na memória
Operações que um profissional executa para realizar suas responsabilidades	Métodos (ou operações) que compõem a programação de um objeto
Gestores e especialistas no negócio	Analistas e programadores
Manual de procedimentos da organização	Diagramas de sequência

os diagramas de sequência compõem um subconjunto importante dos diagramas da UML genericamente chamados de *diagramas de interação*, por enfatizarem a interação entre objetos.

Os diagramas de sequência e os diagramas de comunicação (estes não discutiremos em detalhes neste texto) expressam basicamente a mesma informação, mas mostram-na de forma diferente. DSs são melhores que diagramas de comunicação na apresentação das responsabilidades de cada objeto, especialmente quando o aspecto da ordenação temporal é relevante (o DS, como você verá, tem um eixo para representar a dimensão temporal).

A Figura 9.1 ilustra uma colaboração simples representada por um diagrama de sequência e por um diagrama de comunicação, em que o objeto `A` solicita algo ao objeto `B`, que por sua vez solicita algo ao objeto `C` e se autodelega a execução de algo, nessa sequência (não se preocupe agora com o que significa cada símbolo nos diagramas; apenas caminhe de objeto a objeto seguindo a ordem numérica crescente das mensagens). Como as duas notações são facilmente "intercambiáveis", algumas ferramentas CASE dispõem de funcionalidades para converter automaticamente um diagrama no outro.



(a) (b)

Figura 9.1: Diagramas de sequência (a) e de comunicação (b) especificando uma colaboração entre três objetos.

Não se preocupe, portanto, em desenvolver ambos os diagramas para uma mesma situação em um sistema. Aliás, isso não é nem um pouco recomendável, não só por ser inútil como também porque aumenta a possibilidade de inconsistência entre os diagramas do modelo, embora bons CASEs ajudem nessa questão.

Os diagramas de sequência têm grande importância na fase de projeto de sistemas computacionais, pois eles são usados para atribuir responsabilidades aos objetos do sistema e para a descoberta de operações das classes, incluindo os detalhes dos parâmetros dessas operações. Em outras palavras, DSs servem para descrever como grupos de objetos colaboram em algum comportamento do sistema, representando o seu "funcionamento", ou seja, a sua programação.

Os DSs e os demais diagramas de interação são a principal ponte entre a análise e a implementação de um sistema, sendo por isso usados principalmente na fase de projeto. Dispondo de uma boa ferramenta CASE e provendo um conjunto de diagramas de sequência e o diagrama de classes com detalhes suficientes, podemos gerar código automaticamente. A isso chamamos de engenharia direta (a partir do modelo, a ferramenta gera o código). Boas ferramentas CASE também dispõem de recursos para a execução de engenharia reversa, ou seja, a partir do código, a ferramenta gera o diagrama de sequência e o de classes.

9.2 Cenários

Mencionamos no Capítulo 3 que uma instância de um caso de uso é uma execução dele. Instâncias diferentes de um mesmo caso de uso podem produzir resultados distintos, dependendo do contexto de cada uma e das ações, que podem ser distintas, dos seus atores. Cada instância executada trilha possivelmente um caminho diferente, desde o início do caso de uso até um dos possíveis finais dele. Cada um desses caminhos diferentes caracteriza um *cenário*.

Como ilustração, imagine o caso de uso `Sacar Dinheiro no Caixa Eletrônico`. Há um início (na maioria dos bancos o processo começa com a passagem do cartão do cliente na máquina) e vários possíveis fins, como:

- o cliente tem o dinheiro e saca o que quer;
- o cliente não tem o dinheiro, mas pode usar o limite do cheque especial e saca o que quer;
- o cliente não tem o dinheiro todo e saca o que pode;
- o cliente não tem nada em conta, não tem cheque especial e nada saca;
- o cliente tem o dinheiro em conta, mas não saca o que quer porque já passou das 22h...

Existe, ainda, a possibilidade de o cliente ter esquecido a senha, o que também vai levar o processo de saque a um ou mais outros possíveis finais.

Instâncias diferentes de um mesmo caso de uso compõem *cenários* diferentes; um otimista, em que tudo dá certo; outros pessimistas, em que tudo dá errado e ainda outros nem tão lá, nem tão cá, em que nem tudo ocorre como o ator gostaria, mas mesmo assim ele fica satisfeito no final.

O principal cenário corresponde ao que chamamos no Capítulo 4 de curso (ou fluxo) típico, que é a sequência de passos que ocorre tipicamente. No caso do saque no caixa eletrônico, o curso típico deve corresponder ao cenário otimista, ou seja, ao sucesso no saque sem nenhum contratempo. Os demais cenários passam por situações alternativas (os cursos ou fluxos alternativos).

> **ATENÇÃO:** O único curso que define precisamente um cenário é o curso típico, pois as condições para que ele seja trilhado do início ao fim estão bem definidas (são as condições de guarda que tipicamente ocorrem). Cursos alternativos não definem cenários, pois um mesmo curso alternativo pode ser trilhado em vários cenários.

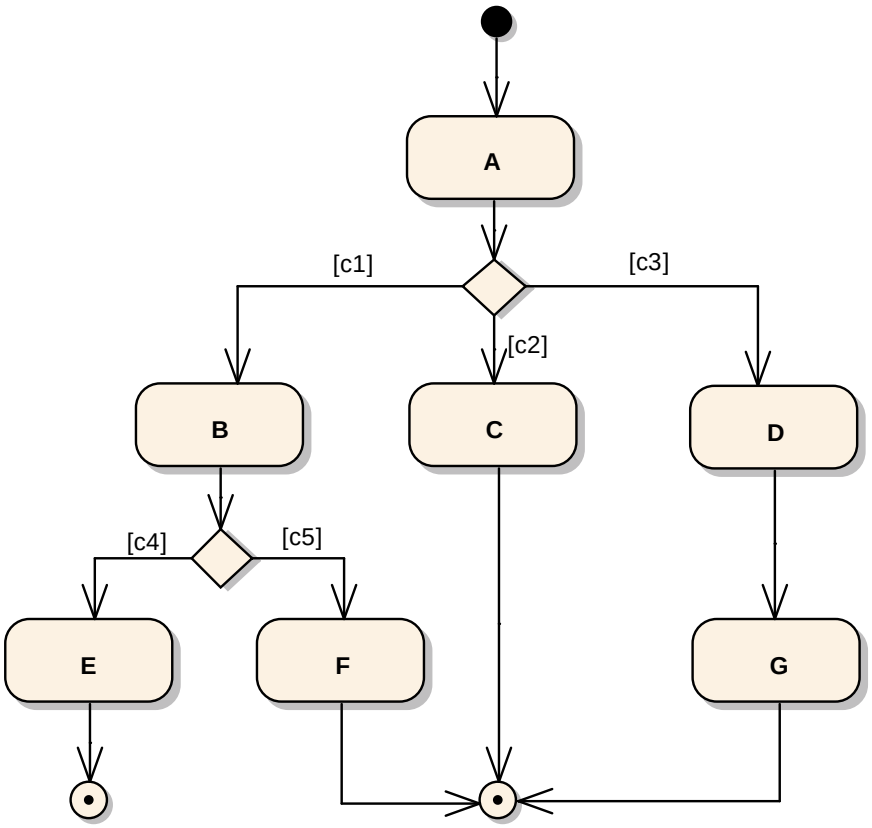


Figura 9.2: Diagrama de atividade facilitando a identificação visual de cenários. Cenário 1: ações `A` e `C`. Cenário 2: ações `A`, `D` e `G`. Cenário 3: ações `A`, `B` e `F`. Cenário 4: ações `A`, `B` e `E`.

Quando tratamos de diagramas de atividade no Capítulo 8, vimos que uma das aplicações desses diagramas é a especificação de casos de uso. Quando elaboramos a especificação de um caso de uso usando DAs, podemos visualizar bastante claramente os caminhos entre o início do caso de uso e todos os possíveis fins dele.

A Figura 9.2 mostra um diagrama de atividades para ilustrar o mecanismo de identificação visual dos possíveis cenários.

Na Figura 9.2 especificamos os cenários pelas ações executadas em cada um deles. No entanto, muitas vezes é mais efetivo especificá-los pelas condições correspondentes aos desvios feitos ao longo dos fluxos. Assim, o cenário 1 seria mais efetivamente especificado pela condição `c2` verdadeira; o cenário 2, pela condição

`c3` verdadeira; o cenário 3, pelas condições `c1` e `c5` verdadeiras; e o cenário 4, pelas condições `c1` e `c4` verdadeiras. Poderíamos especificar algo como "Cenário 1: condição `c2` verdadeira" etc.

Cenários são um conceito importante para diagramas de sequência porque são usados como orientação para as suas elaborações com vistas à diminuição da complexidade visual, conforme trataremos adiante, ainda neste capítulo.

9.3 O Ciclo de Vida dos Objetos

O ciclo de vida de um objeto compreende tudo que acontece com ele durante o tempo decorrido entre a sua instanciação e a sua destruição.

Essa definição é, em princípio, simples, mas causa muita confusão nos alunos porque há dois contextos em que ela se aplica e há dois tipos diferentes de objetos aos quais se aplica.

Os dois contextos correspondem aos dois tipos de memória em que o objeto pode se encontrar armazenado: a memória principal e a memória secundária. Os tipos de objeto são: os *persistentes* e os de *vida efêmera*.

Objetos persistentes são aqueles que precisam ser enviados à memória secundária (em disco, num banco de dados, por exemplo) para que "sobrevivam" ao desligamento do sistema, se mantendo, portanto, acessíveis após o religamento do sistema. Se eles já existem na memória secundária e precisam participar de uma colaboração, eles são trazidos (na realidade, copiados) da memória secundária para a memória principal para que possam colaborar. Se não são mais necessários na memória principal, eles são copiados de volta para a memória secundária, caso tenham sido modificados, e removidos da memória principal para que o espaço que ocuparam ali possa ser liberado para uso por outros objetos. Objetos persistentes normalmente são instâncias de classes de conceito como um determinado pedido ou um determinado cliente da ZYX.

Objetos efêmeros (ou de vida efêmera) são aqueles que são instanciados para colaborar em alguma situação, podendo ser, depois disso, descartados, ou seja, simplesmente removidos da memória, sem necessidade de serem copiados para o disco. Isso libera espaço na memória principal para a colocação de outros objetos. Normalmente esses objetos são instâncias de classes de projeto ou das chamadas *classes utilitárias* como, por exemplo, classes de manipulação de datas, de cadeias de caracteres, de coleções etc.

Independentemente do tipo de objeto, ele precisa estar na memória principal para que possa colaborar para a realização de uma funcionalidade do sistema, enviar

mensagens a outros objetos ou receber mensagens de outros objetos, executando uma ou mais operações para tratá-las. Daí, a confusão que muitos fazem é devida à necessidade de instanciar (criar um espaço na memória) um objeto da classe `Pedido`, por exemplo, para poder trazer os dados de um determinado pedido do banco de dados para a memória, achando que esse é o início do ciclo de vida desse pedido. Na realidade, o ciclo de vida do tal pedido já foi iniciado quando ele foi instanciado e, depois, armazenado em disco pelo caso de uso de criação de pedidos. O ciclo de vida só terminará quando o pedido for removido do banco de dados, feito por meio de outra funcionalidade do sistema, e não quando o objeto for eliminado da memória, para liberação de espaço, após sua persistência em disco.

Por essas razões, objetos persistentes têm normalmente ciclo de vida maior do que objetos efêmeros.

9.4 Responsabilidades, Atributos e Operações dos Objetos

Com o exemplo dos contadores e do fechamento do balanço da ZYX, falamos há pouco de responsabilidades e das operações necessárias para realizá-las. Ensinar como programadores escolhem os objetos que vão usar em uma colaboração é, presumo eu, igual a ensinar um recrutador de um setor de Recursos Humanos a escolher pessoas para contratação.

As responsabilidades que atribuímos a um objeto e, portanto, aos demais objetos da mesma classe, devem ser baseadas na capacidade desse objeto de cumpri-las; não vejo, por exemplo, algum sentido em contratar um médico para fechar os balanços da ZYX. Essa capacidade é, em primeira análise, baseada nos atributos desse objeto. Uma dica que dou aos projetistas é, mais uma vez, baseada em uma lógica que me parece óbvia para administração de recursos humanos:

> Ao designarmos alguém para executar algo, se temos de passar muita informação e detalhes a essa pessoa para que ela possa cumprir a tarefa, esse é um sinal de que não estamos designando a pessoa certa para a execução da tarefa.

Trazendo essa lógica para nosso contexto, a escolha de um objeto para cumprir uma responsabilidade em uma colaboração deve ser feita com base nos atributos desse objeto, incluindo seus relacionamentos com outros objetos. Por exemplo, se precisamos obter o nome de um cliente da ZYX por meio dos seu código de cliente, nada melhor do que atribuir à *coleção de clientes* da ZYX a responsabilidade de

responder a consultas desse tipo, na medida em que a instância dessa classe possui associação com todas as instâncias da classe `Cliente`, que são as que efetivamente têm os valores dos atributos `codigoCliente` e `nomeCliente`.

Há, portanto, situações em que um objeto não possui os atributos necessários para o cumprimento de determinada responsabilidade, mas possui uma associação com um ou mais objetos que possuem esses atributos. Nesse caso, é válido atribuir ao primeiro objeto a tal responsabilidade, que a delega parcial ou totalmente aos objetos associados que possuem esses atributos.

Não sei se você percebeu, mas quando falamos que a coleção de clientes deveria ter a responsabilidade de obter o nome de um cliente dado o seu código, acabamos de descobrir a necessidade de uma nova classe no diagrama: a classe `ColecaoDeClientes` (`Clientela` também é um bom nome), cuja instância "aponta" para todos os clientes da ZYX, ou seja, possui associações com todas as instâncias da classe `Cliente`. Como cada instância da classe `Cliente` (típica e idealmente) não conhece as demais, podemos atribuir mais uma responsabilidade à classe `ColecaoDeClientes`: organizar a lista de (indexar) clientes da ZYX. Provavelmente outras responsabilidades serão atribuídas a essa classe conforme o projeto avança.

Enumeramos, então, algumas das responsabilidades da classe `ColecaoDeClientes` que são tipicamente encontradas em classes que representam coleções:

- indexar os clientes da ZYX para um acesso eficiente à lista;
- pesquisar clientes na lista de clientes;
- atualizar no banco de dados qualquer alteração feita em qualquer cliente da ZYX;
- manter-se atualizada caso alguma alteração na lista seja feita por outro usuário; etc.

Para efetuar uma busca de cliente por seu código, por exemplo, solicitamos o cliente ao objeto da classe `ColecaoDeClientes` passando seu código como parâmetro de busca, e esse objeto trata de consultar cada objeto da classe `Cliente` com o qual ele se relaciona, perguntando seu código e, caso o código seja o da consulta, retornando o cliente.

É importante, nesse momento, voltarmos ao diagrama de classes e atualizá-lo, adicionando no diagrama a nova classe que acabamos de descobrir associada à classe `Cliente`. O trecho alterado do diagrama de classes (aquele da Figura 5.2) ficará como na Figura 9.3.

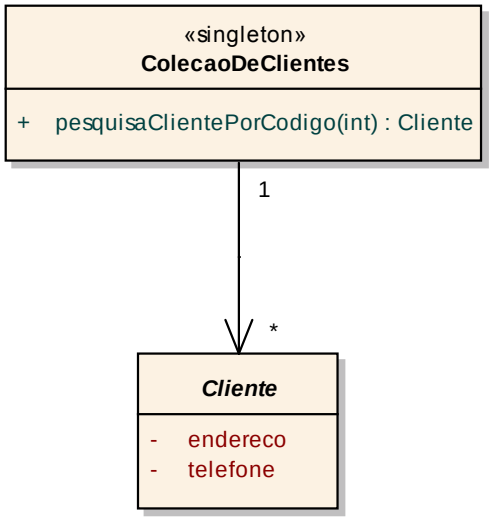


Figura 9.3: Incorporação de classe de projeto ao diagrama conceitual.

Repare que essa nova classe não é uma classe conceitual, porque não faz parte do conceito. Ela foi criada para auxiliar nas situações que enumeramos. Ela é, portanto, uma classe de projeto. Com isso, o diagrama de classes de conceito (ou conceitual) é promovido a diagrama de classes de projeto (ou de análise e projeto, como preferem alguns projetistas) pela adição dessa classe no modelo.

Repare também que essa classe recém-descoberta deve ter as seguintes características adicionais:

- deve possuir uma única instância, já que todo índice deve ser único. Essa é a razão pela qual marcamos a classe como sendo `«singleton»`, que é um padrão de projeto que garante que a classe só será instanciada uma única vez, mesmo que invoquemos seu construtor inúmeras vezes. Caso queira mais informações sobre padrões de projeto, consulte a referência clássica sobre o assunto: *Design patterns: elements of reusable object-oriented software*, de Erich Gamma, Richard Helm, Ralph Johnson e John Vlissides ([7]);

- o objeto não precisa ir para o banco de dados (não é uma classe persistente), pois pode ser reconstruído sempre que o sistema é iniciado.

Portanto, as operações necessárias ao cumprimento das responsabilidades de um objeto, se ainda não o foram, são descobertas na hora em que o colocamos para

colaborar, passando a constar, nesse momento, do compartimento de operações da classe do objeto no diagrama de classes.

Eventualmente durante esse processo descobrimos outros atributos (notadamente os derivados) e associações que se mostram necessários para tornar o trabalho de programação mais simples.

9.5 O Tripé da Análise

Os DSs (aliás, os diagramas de interação, de maneira geral) completam o que chamamos *tripé da análise*, cujas outras "pernas" são os casos de uso (a parte preponderantemente funcional), com suas descrições, e os diagramas de classes (a parte preponderantemente estrutural).

Costumamos dizer que os casos de uso em um sistema especificam *o que é para ser feito*; as classes do diagrama de classes relacionam *com quem faremos o que é para ser feito*; e os DSs são *como faremos o que é para ser feito, considerando com quem faremos* o sistema.

Essa conexão entre os três conjuntos de diagramas em um sistema se dá conforme ilustrado na Figura 9.4, que prezo como a ilustração de uma das etapas de maior importância em todo o processo de desenvolvimento de um sistema.

De forma geral, o processo de construção (codificação) de sistemas com orientação a objetos segue sequência de passos mostrada na Figura 9.4 para cada cenário de cada caso de uso. Essa figura ilustra a sequência de aplicação dos passos (conforme numeração entre parênteses), descritos a seguir:

1. tomamos como base o que temos a fazer (isso não é lá tanta novidade, convenhamos!). Isso é feito por meio do cenário extraído do caso de uso, ou seja, com base na sequência de ações que o sistema precisa executar para tratar (processar) corretamente o cenário enfocado;

2. para cada ação que compõe o cenário e com base no diagrama de classes, procuramos obter uma relação dos objetos que deverão colaborar para a realização da ação. Essa relação é feita com base nas responsabilidades que atribuímos a cada objeto (reveja o que falamos sobre responsabilidades, atributos e operações de objetos);

3. estabelecemos a sequência de mensagens (chamadas de operações e parâmetros das operações) que um precisa passar a outro para que as ações sejam executadas;

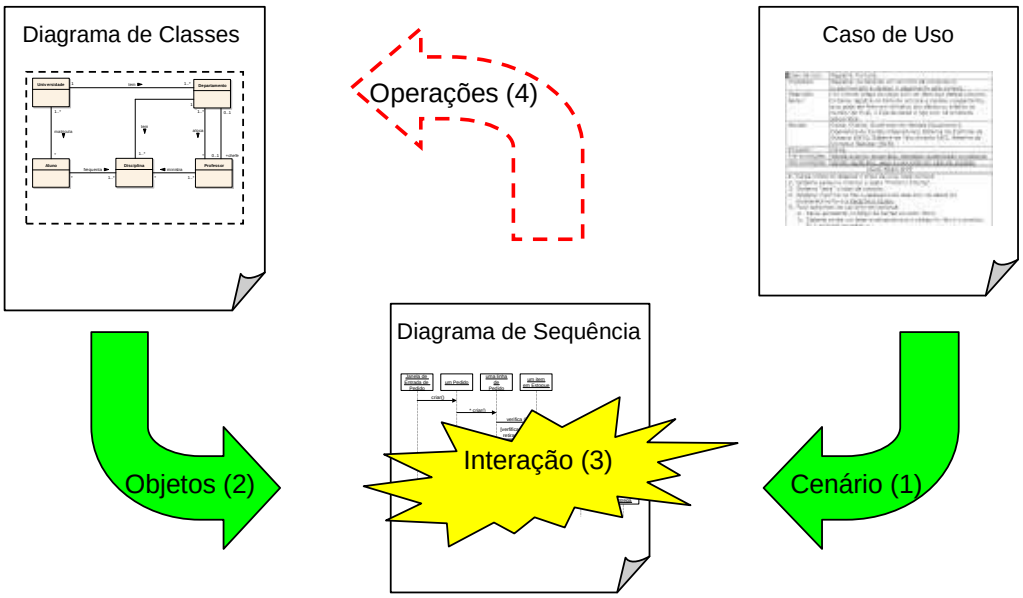


Figura 9.4: Diagramas de sequência na formação de modelos completos de sistemas computacionais. Os números entre parênteses correspondem à ordem segundo a qual cada informação é tratada no processo de construção do código para o sistema.

4. relacionamos, nos compartimentos de operações das classes dos objetos envolvidos, as operações e os parâmetros descobertos durante esse processo.

Como você pode constatar, os diagramas de sequência tendem a "puxar" o projetista para níveis mais baixos de abstração, conduzindo-o aos detalhes (nomes de operações, parâmetros, atributos de projeto e visibilidades). Dessa forma, elaborar DSs mantendo-se no nível conceitual não é uma tarefa simples (e na maioria das vezes isso não é desejado), pois, como já mencionamos, DSs propiciam a representação dos detalhes.

9.6 As Dimensões dos Diagramas de Sequência

Ao contrário dos diagramas até então apresentados, os DSs possuem eixos nas duas dimensões: a horizontal e a vertical. De forma bem geral, na dimensão horizontal relacionamos os objetos que participarão da colaboração que estamos modelando.

A dimensão vertical representa a passagem do tempo, especificando as mensagens trocadas entre os objetos de forma ordenada com respeito ao tempo.

Os detalhes de como apresentamos os elementos da notação gráfica nessas duas dimensões são apresentados nas duas seções a seguir.

A Dimensão Horizontal dos Diagramas de Sequência

Objetos e usuários (instâncias das classes e dos atores, respectivamente) compõem a dimensão horizontal de um diagrama de sequência. A ordem da colocação dos objetos nessa dimensão não tem significado; normalmente é usada a ordem em que vamos precisando da colaboração dos objetos, ou seja, conforme vamos adicionando os objetos na colaboração durante o processo de modelagem. De tempos em tempos, durante esse processo, damos uma "arrumada" na ordem dos objetos, arrastando-os para a esquerda ou direita, objetivando uma melhor apresentação visual do modelo, sem qualquer alteração em seu significado.

No modelo ilustrado na Figura 9.1a, poderíamos, por exemplo, arrastar o objeto `B` para a esquerda do objeto `A` (resultando, portanto, na ordem `B`, `A` e `C` dos objetos, da esquerda para a direita na dimensão horizontal), sem qualquer alteração no significado do modelo.

Objetos são representados por retângulos contendo seus identificadores, formulados tipicamente conforme uma das seguintes maneiras:

- `obj:classe` (por exemplo, `Manuel:Supervisor`, significando o "objeto" `Manuel` da classe `Supervisor`);
- `:classe` (por exemplo, `:Supervisor`, significando um determinado objeto da classe `Supervisor`, não sendo importante, no entanto, qual é esse objeto); ou
- `um(a)Classe` (por exemplo, `umSupervisor`, significando a mesma coisa que no item acima).

Quando, para a realização de uma atividade, são necessários, por exemplo, dois objetos da classe `Funcionario`, uma sugestão é que os identifiquemos no diagrama como `umFuncionario` e `outroFuncionario` ou `primeiroFuncionario` e `segundoFuncionario`.

Habitualmente sublinhamos os identificadores para denotar que nos referimos a instâncias (e não a classes). O sublinhado para denotar objetos em diagramas de sequência não é mais obrigatório a partir da UML 2.0.

A Dimensão Vertical dos Diagramas de Sequência

A dimensão vertical representa o tempo, que evolui de cima para baixo nos DSs, ou seja, o que está abaixo no diagrama acontece depois do que está acima.

As linhas de vida dos objetos compõem a dimensão vertical, especificando o que acontece com eles, representando como interagem entre si. Representam também o foco de controle, ou seja, a ativação e a desativação de objetos quando eles executam algo ou delegam a algum outro objeto a execução de algo por meio de mensagens que enviam. Representam tudo que acontece com os objetos durante seus ciclos de vida.

A linha de vida é denotada por um segmento de reta vertical tracejado, iniciando na base do retângulo de identificação do objeto e indo até o final do diagrama ou, se for o caso, até sua remoção.

Por sobre as linhas de vida podemos colocar o que chamamos de caixas de ativação, que denotam que o objeto tem o foco de controle, significando que ele está executando algo diretamente ou delegando a algum outro objeto a execução de algo.

Caixas de ativação são opcionais pela UML. Sem elas, os diagramas ficam mais fáceis de desenhar a mão livre, mas a ausência dificulta a leitura do diagrama em situações em que há mais de um fio de execução (paralelismo).

Caixas de ativação podem se empilhar para especificar *autodelegações*, que significam que um objeto solicita a ele mesmo que execute algo.

A Figura 9.5 ilustra os conceitos que foram apresentados considerando o ciclo de vida de um objeto.

Na Figura 9.5 ilustramos as caixas de ativação, a passagem de mensagens de criação e de destruição de objetos e uma autodelegação (um objeto invoca uma operação de si mesmo). Não é necessário, nesse momento, entender todos os aspectos da notação que constam da figura. Os detalhes ficam para o próximo capítulo.

As linhas de vida podem se subdividir em duas ou mais linhas de vida para especificar situações concorrentes ou alternativas, correspondendo a cenários diferentes de um mesmo caso de uso. As linhas podem se unir novamente mais adiante no diagrama. Com isso, o diagrama pode ficar confuso, daí a ideia de usar um diagrama para cada cenário do caso de uso[1] ou usar os recursos na notação que foram introduzidos na versão 2.0 da UML e de que também trataremos no próximo capítulo.

[1] Os diagramas de sequência muito detalhados, especificando situações alternativas (desvios e intercalações) e paralelismo (separações e junções) são usualmente muito complexos visualmente e, como consequência, perdem boa parte da utilidade. Por essas razões, recomenda-se usar um diagrama para cada cenário do caso de uso.

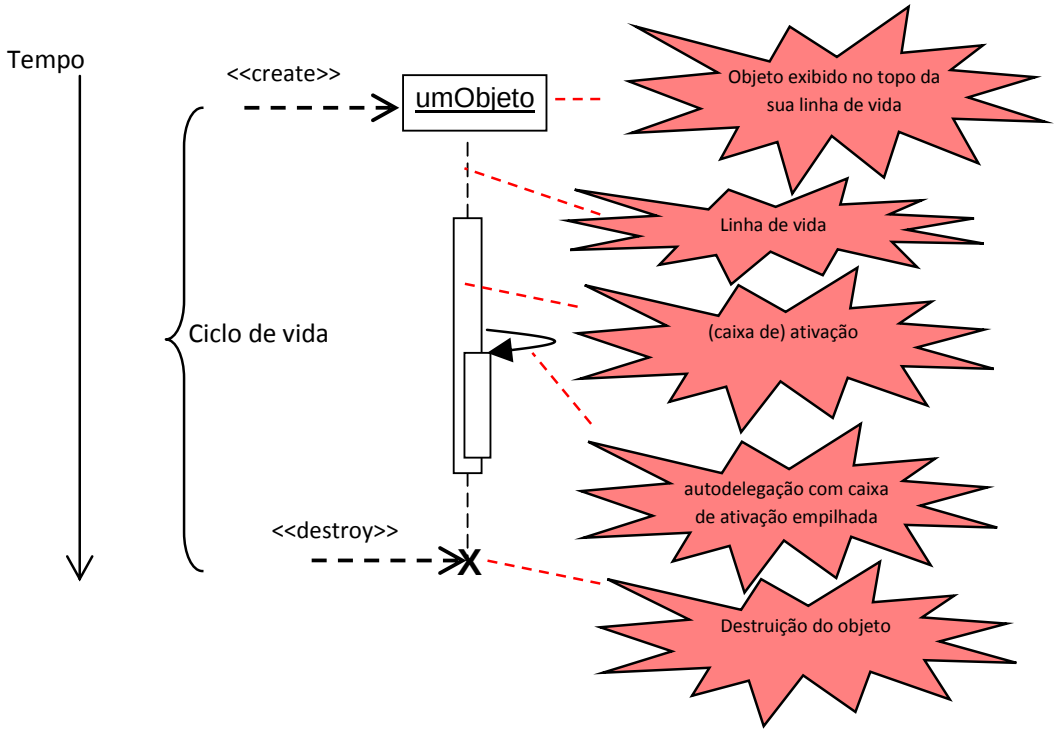


Figura 9.5: O ciclo de vida de um objeto representado em um diagrama de sequência.

9.7 Nível de Detalhamento dos Diagramas de Sequência

Um aspecto importante a ser considerado na elaboração de DSs é a finalidade à qual se prestarão: se para documentação, apenas, ou se para a geração de código. Essas duas finalidades são antagônicas com respeito ao nível de detalhamento dos diagramas.

Para a documentação, ou seja, para que sejam compreensíveis por humanos, os DSs não devem ser complexos e, portanto, não devem ser detalhados. Para que se prestem à geração automatizada de código, os DSs precisam ser detalhados e, portanto visualmente muito complexos, até para o pessoal que os elabora e os mantém.

Por essa razão, os DSs são pouquíssimo desenvolvidos para comporem documentos de análise e mais frequentemente usados pelos projetistas, que elaboram DSs até certo nível de detalhes e os entregam aos programadores, que geram o código usando a engenharia direta e, daí por diante, trabalham unicamente com o código.

9.8 Resumo do Capítulo

De forma análoga aos processos executados por indivíduos em uma organização, programas de computador desenvolvidos segundo o paradigma de orientação a objetos são concebidos com base na colaboração para a realização das funções do sistema.

Os diagramas de sequência (DSs) permitem especificar colaborações entre objetos, descrevendo a sequência de mensagens passadas de objeto a objeto necessária para a realização de uma funcionalidade em um sistema. Por enfatizar a interação entre objetos, os diagramas de sequência compõem um subconjunto importante dos diagramas da UML genericamente chamados de diagramas de interação.

Os diagramas de sequência têm grande importância na fase de projeto de sistemas computacionais, pois são usados para atribuir responsabilidades aos objetos do sistema e para a descoberta de operações das classes, incluindo os detalhes dos parâmetros dessas operações.

É possível que cada instância do caso de uso executada trilhe um caminho diferente, desde o início do caso de uso até um dos possíveis finais dele. Cada um desses caminhos diferentes caracteriza um cenário. O principal cenário corresponde ao que chamamos de curso típico.

O ciclo de vida de um objeto compreende tudo que acontece com ele durante o tempo decorrido entre sua instanciação e sua destruição. O ciclo de vida de um objeto persistente começa quando ele é instanciado e armazenado em disco e só termina quando o pedido é removido do banco de dados – e não quando o objeto é eliminado da memória para liberar espaço para outro objeto. Objetos persistentes têm normalmente ciclo de vida maior do que objetos efêmeros.

As responsabilidades que atribuímos a um objeto são realizadas pelas operações que ele implementa e devem ser baseadas na capacidade desse objeto de cumpri-las. Essa capacidade é, em primeira análise, baseada nos atributos desse objeto, incluindo seus relacionamentos com os demais objetos, pois há situações em que um objeto não possui os atributos necessários para o cumprimento de determinada responsabilidade mas possui uma associação com um ou mais objetos que possuem esses atributos.

Os DSs completam o *tripé da análise*, cujas outras "pernas" são os casos de uso com suas descrições e os diagramas de classes. O processo de codificação de um sistema com orientação a objetos segue uma sequência de passos que considera os cenários dos casos de uso e as classes do diagrama de classes como "entradas" para esse processo.

Diagramas de sequência possuem duas dimensões: horizontal e vertical. Na

dimensão horizontal relacionamos os objetos que participarão da colaboração que estamos modelando; na dimensão vertical, que representa o tempo, representamos, de forma ordenada com respeito ao tempo, as mensagens trocadas entre os objetos para que se dê a colaboração entre eles.

No próximo capítulo prosseguiremos com o estudo dos diagramas de sequência, tratando dos demais aspectos da notação gráfica e das técnicas e dicas necessárias para a interpretação e elaboração dos DSs.

9.9 Exercícios Propostos

1. Especifique, por meio das condições correspondentes aos desvios feitos ao longo dos fluxos, os cenários identificados no DA da Figura 8.1.

2. Dado o diagrama de classes da ZYX da Figura 9.6, enumere os passos da colaboração necessária para a impressão de um determinado pedido, incluindo os dados do cliente, a quantidade, as descrições, os preços unitários, os preços totais de cada item e o do pedido. Ao final, relacione as responsabilidades de cada objeto envolvido na colaboração. Leve em consideração o seguinte aspecto: as navegabilidades das associações que não foram indicadas são bidirecionais. Fique à vontade para criar classes cujos objetos você julgue necessários para a tarefa.

3. Com base no que discutimos a respeito das dimensões horizontal e vertical dos DSs, esboce o diagrama de sequência que especifica a seguinte colaboração:

   João, presidente da ZYX, solicita a Paulo, diretor financeiro, o valor do lucro líquido da Empresa. Paulo, por sua vez, desconhecedor dos valores das receitas e das despesas, solicita a Maria, gerente de faturamento, o valor das receitas. Após receber o valor das receitas, Paulo solicita a Pedro, gerente de operações, o valor das despesas. De posse desses dois valores, Paulo calcula o lucro líquido (diferença entre receitas e despesas) e o envia a João como resposta à solicitação inicial.

   Imagine que João, Paulo, Maria e Pedro sejam objetos instanciados de suas respectivas classes e todos se encontram instanciados quando a colaboração se inicia.

   Use uma das convenções de nomes de objetos que mencionamos, atente para a ordem da passagem das mensagens, mas não se preocupe com a notação rigorosa da UML com relação às mensagens. Se você usar um CASE para resolver o exercício, também não se preocupe com as caixas de ativação. Seria ideal, no caso, se pudéssemos desabilitá-las, mas eu não conheço um CASE disponível hoje em dia em que isso seja possível.

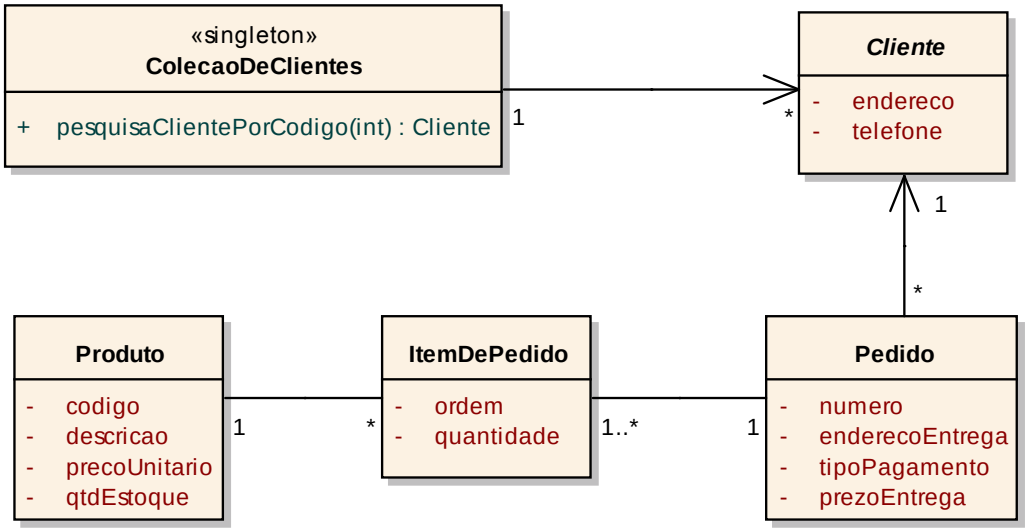


Figura 9.6: Trecho do diagrama da ZYX no nível de especificação.

As soluções encontram-se a partir da Página 223.

CAPÍTULO 10

DIAGRAMAS DE SEQUÊNCIA: MENSAGENS, QUADROS DE INTERAÇÃO, CONTROLADORES E INTERFACES

> Everything requires time. It is the only truly universal condition. All work takes place in time and uses up time. Yet most people take for granted this unique, irreplaceable, and necessary resource. Nothing else, perhaps, distinguishes effective executives as much as their tender loving care of time.
>
> Peter Drucker

Vimos no Capítulo 9 que objetos colaboram para a realização das funcionalidades do sistema que estamos construindo. Essa colaboração considera as responsabilidades atribuídas a cada objeto e é coordenada por uma sequência programada de ações e mensagens que os objetos trocam entre si.

Os diagramas de sequência da UML ajudam a atribuir responsabilidades a objetos e permitem que especifiquemos quais objetos participam das colaborações e

quais são as sequências de ações executadas e de mensagens passadas entre os objetos.

Neste capítulo prosseguiremos o estudo desses diagramas, tratando dos demais aspectos da notação gráfica e das técnicas e dicas necessárias para a interpretação e elaboração dos DSs.

O primeiro assunto a ser abordado diz respeito aos tipos de mensagens que podem ser passadas entre os objetos.

10.1 As Mensagens de Chamada

As chamadas correspondem à invocação de alguma operação de um objeto feita por outro objeto ou, eventualmente, de um objeto por ele mesmo.

As chamadas são representadas por setas com pontas fechadas que partem da linha de vida do objeto que envia a mensagem, solicitando a execução de algo, para a linha de vida do objeto alvo, aquele que executa o que é solicitado.

As setas são rotuladas com, pelo menos, o nome da função e os valores ou expressões de seus parâmetros, ou seja, os dados necessários para que o objeto invocado execute o que foi solicitado.

Os rótulos podem conter também as condições em que as mensagens são enviadas, especificadas entre "[" e "]". As condições de guarda prefixando as mensagens, embora tenham sido substituídas por quadros (frames) com rótulos opt ou alt (oriundos do inglês optional e alternative, respectivamente), ainda são amplamente usadas. Os quadros serão tratados mais adiante neste capítulo.

A Figura 10.1 ilustra a situação em que o objeto `A` solicita ao objeto `B` a execução da operação op1 passando os parâmetros `p1` e `p2`. A operação `op1` faz parte, portanto, do rol de operações públicas de `B`, pois o objeto `A` tem de "enxergar" e poder executar essa operação do objeto `B`, daí a necessidade de a operação `op1` ser pública.

Como já mencionamos, um objeto pode solicitar a si próprio a execução de algo, o que caracteriza o que se chama *autochamada* ou *autodelegação* (o objeto delega a si próprio a execução de algo). Suponhamos que, durante a execução da operação `op1` por `B`, este objeto precise executar a operação `op2`, que consta de seu rol de operações, passando os parâmetros `p3` e `p4`. A Figura 10.2 ilustra essa situação.

Repare que, nesse caso, a operação `op2` não precisa ser pública se nenhum outro objeto, além de `B`, precisar executá-la. Na realidade, se esse for mesmo o caso, a operação `op2`, embora podendo ser pública, deverá ser tornada privada por questões de boa prática (encapsulamento).

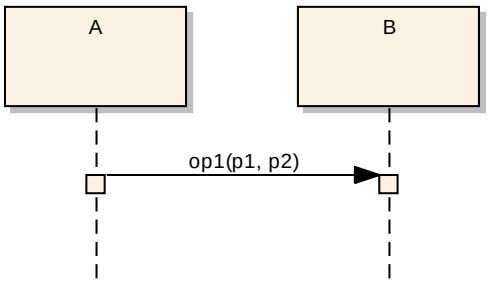


Figura 10.1: Mensagem de chamada entre dois objetos.

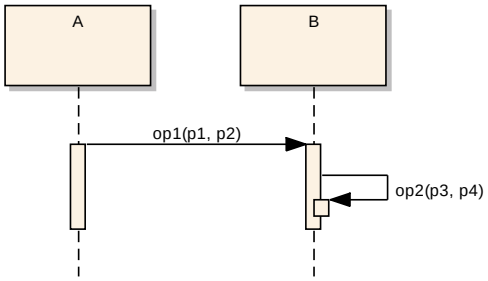


Figura 10.2: Autodelegação da operação `op2` pelo objeto `B`.

10.2 Mensagens de Criação e Destruição de Objetos

É possível representarmos a criação e a destruição de objetos, conforme você verá a seguir.

A criação (instanciação), que corresponde à execução do método construtor do objeto, é denotada por uma seta tracejada com a ponta aberta partindo da linha de vida do objeto que solicita a criação para o ponto onde se inicia a linha de vida do objeto que será criado. Nesse ponto da dimensão vertical, que representa o início do ciclo de vida do objeto sendo criado, coloca-se a caixa de identificação do objeto (o retângulo com o nome do objeto).

A Figura 10.3 ilustra a situação em que um objeto `A` solicita a criação de um objeto `B` (em outras palavras, `A` executa o método construtor do objeto `B`) que, por sua vez, solicita a criação do objeto `C`.

Há situações em que objetos não são criados, sendo apenas "convocados" para

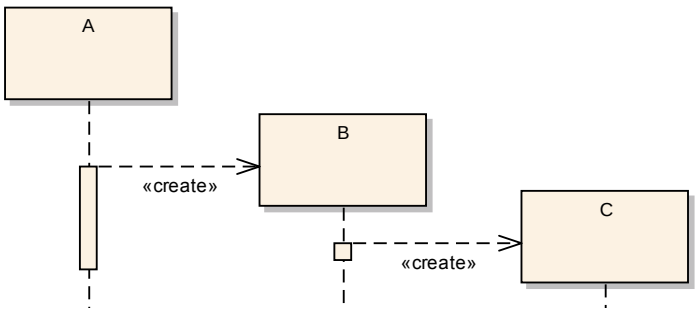


Figura 10.3: Mensagens de solicitação de criação de objetos ilustrando as caixas de identificação dos objetos alinhadas com as mensagens.

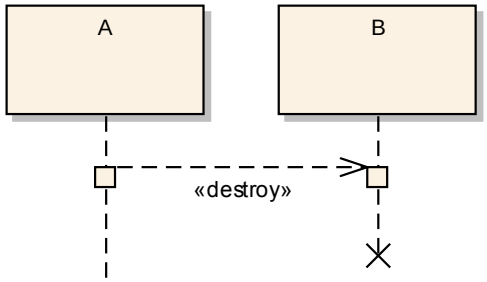


Figura 10.4: Mensagens de solicitação de criação de objetos ilustrando as caixas de identificação dos objetos alinhadas com as mensagens.

colaborarem um caso de uso do sistema. Objetos que já existem quando o caso de uso se inicia têm seus identificadores (retângulos com seus nomes) colocados no topo do diagrama. Os que são criados ao longo do caso de uso têm seus identificadores colocados mais abaixo dos demais, conforme são criados (ver Figura 10.3).

A destruição de um objeto pode ser comandada por outro objeto, por meio do envio de mensagem de destruição (é também uma seta tracejada de ponta aberta), ou seja, pela execução do método destrutor do objeto. A "morte" do objeto é denotada por um "X" em negrito. Nesse ponto, a linha de vida do objeto se encerra, pois, em se tratando de objetos, não há vida após a morte! A Figura 10.4 ilustra a situação em que o objeto `A` solicita a destruição do objeto `B`.

Um objeto pode "sobreviver" à execução de um cenário de um caso de uso, ou seja, não necessariamente precisa ser destruído quando o caso de uso se encerra.

Por exemplo, quando executamos com sucesso o caso de uso `Cadastrar Cliente`, é esperado que os dados do novo cliente permaneçam no cadastro após o final da execução do caso de uso.

Vale a pena comentarmos um erro, de certa forma comum, que os alunos cometem: terminar um diagrama de sequência colocando sistematicamente um "X" no final das linhas de vida de todos os objetos relacionados no diagrama, confundindo fim de diagrama com fim de linha de vida (destruição) de objeto.

É importante não confundir, no entanto, a destruição de um objeto persistente com a necessidade de o objeto que armazena seus dados na memória ser destruído após os dados serem enviados ao banco de dados para liberação do espaço ocupado. Nesse ponto, vale a pena rever a Seção 9.3.

10.3 Mensagens de Retorno

Outro tipo de mensagem é a de retorno da resposta de uma chamada, com consequente retorno do controle de execução. Esta mensagem, além de possivelmente retornar um valor para o objeto que fez a chamada, passa também o controle de volta a ele, pois a ação solicitada foi concluída pelo objeto chamado. As caixas de ativação do objeto chamado, quando representadas, se encerram com as mensagens de retorno.

Uma mensagem de retorno é representada por uma seta tracejada de ponta aberta que vai do objeto chamado para o objeto chamador, ou seja, no sentido oposto à chamada. Ela pode ser rotulada com a informação retornada.

Assim como as caixas de ativação, segundo a UML, as mensagens de retorno podem ser omitidas. Em chamadas síncronas (sem paralelismo, ou seja, quando o objeto que chama aguarda o fim da execução da operação pelo objeto chamado, conforme vimos tratando até agora em nosso texto), os retornos de controle podem ser facilmente inferidos por simples observação; além disso, suas omissões tornam os diagramas visualmente mais simples. Veja o exemplo da Figura 10.5.

Na Figura 10.5, o objeto `A` envia a mensagem `m1` para o objeto `B`, que, nesse momento, fica ativado, passando a deter o controle de execução. O objeto `B`, então, envia a mensagem `m2` ao objeto `C`, que a processa e, ao final, retorna o controle de volta a `B`, que retorna o controle de volta ao objeto `A`. Sabemos disso porque `A` envia a mensagem `m3` ao objeto `B`, só podendo fazer isso se tiver obtido o controle de volta (um objeto só envia uma mensagem se tiver o controle do processamento). `B` processa a mensagem `m3` e retorna o controle de volta a `A`, que envia a mensagem `m4` ao objeto `C`, que no final retorna o controle para o objeto `A`.

Releia a explicação que acabamos de dar. Se, mesmo relendo o texto que

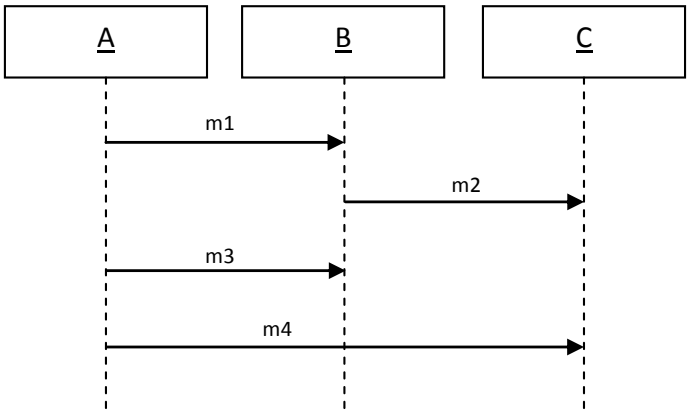


Figura 10.5: Omissão das mensagens de retorno e das caixas de ativação, sem prejuízo de expressividade e de precisão.

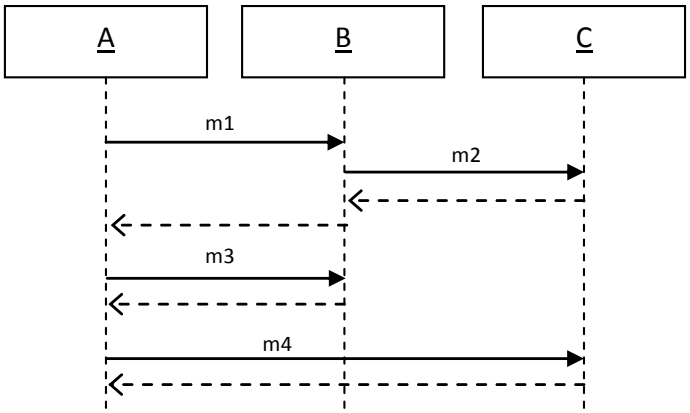


Figura 10.6: Diagrama equivalente ao da Figura 10.5, agora com as mensagens de retorno representadas.

explica a Figura 10.5, ele ainda parecer confuso, tente relê-lo olhando para o diagrama ilustrado na Figura 10.6, que tem o mesmo significado do diagrama da Figura 10.5 mas mostra as mensagens de retorno.

Se agora adicionarmos as caixas de ativação na sequência da Figura 10.6, a compreensão talvez se torne ainda mais fácil. Faremos isso conforme ilustra a Figura

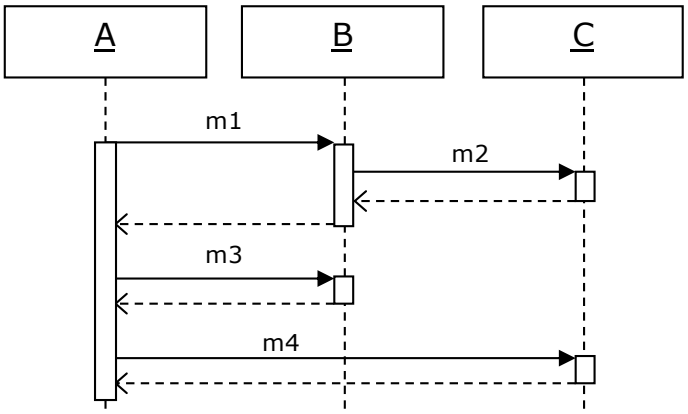


Figura 10.7: Diagrama equivalente aos das Figuras 10.6 e 10.5, agora com as caixas de ativação representadas além das mensagens de retorno.

10.7, quem tem o mesmo significado das Figuras 10.6 e 10.5.

Repare na Figura 10.7: quando um objeto solicita algo a outro objeto, ele permanece ativo pelo tempo em que aguarda a execução do que foi solicitado. A caixa de ativação termina no objeto chamado porque o controle volta ao objeto chamador.

Quando já se tem alguma prática, retornos e caixas de ativação não ajudam muito na interpretação dos diagramas quando estamos tratando de chamadas síncronas, que podem ser omitidos a bem da simplicidade visual.

Caso estejamos lidando com concorrência, enviando *mensagens assíncronas* (com múltiplos fios de execução – `multithreading`), os retornos e as caixas de ativação podem ser vitais para o entendimento correto de um diagrama. Trataremos de mensagens assíncronas a seguir.

> **ATENÇÃO:** O retorno de uma chamada *sempre* acontece para o objeto que fez a chamada, *nunca* para outro objeto.

10.4 Chamadas Assíncronas

Até então tratamos de mensagens síncronas entre objetos em DSs; nessas chamadas, o objeto que chama espera pelo término do processamento da mensagem pelo objeto

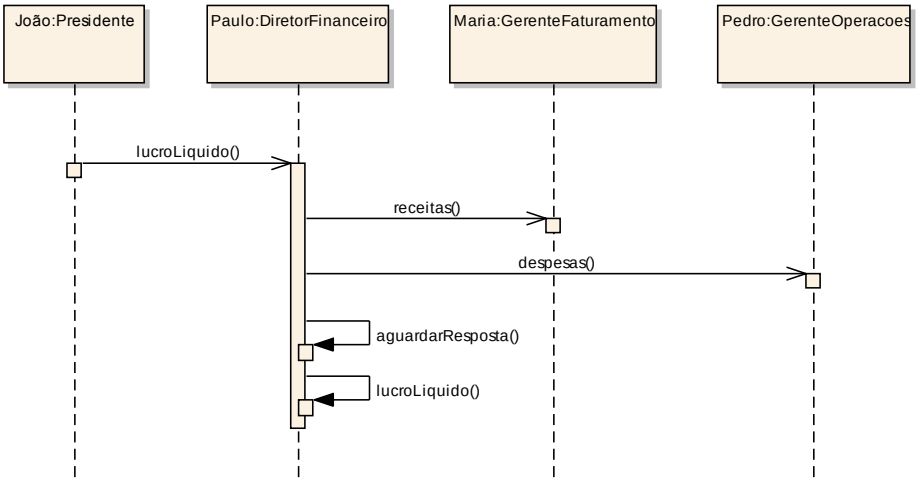


Figura 10.8: Especificação de uma colaboração usando chamadas assíncronas.

chamado. O fato é que um objeto pode delegar a outro a execução de algo utilizando também outro tipo de chamada: as mensagens assíncronas.

Nas chamadas assíncronas, o objeto que chama não aguarda o término do processamento da mensagem pelo objeto chamado para poder fazer outra chamada a outro objeto ou, eventualmente, a si próprio. Em outras palavras: o controle é mantido pelo objeto chamador, mas é passado *também* ao objeto chamado. Isso dá origem a outro fio de execução (*thread*, em inglês).

Enquanto chamadas síncronas são representadas por setas sólidas (não tracejadas) de pontas fechadas, as chamadas assíncronas são representadas por setas sólidas de pontas abertas.

Retornando à situação descrita no último exercício do Capítulo 9, a solução dada levou em consideração o enunciado da atividade à risca, em que Paulo aguarda Maria informar o valor das receitas para só então solicitar o valor das despesas a Pedro. Daí o uso de chamadas síncronas. Isso, no entanto, não acontece na vida real em uma organização em que tipicamente os processos ocorrem em paralelo. Paulo, o diretor financeiro da ZYX, não precisa aguardar o valor das receitas, que solicitou à Maria, para que possa pedir ao Pedro o valor das despesas.

A Figura 10.8 ilustra a passagem das mensagens de forma assíncrona.

Nesse caso, Paulo delega a responsabilidade de calcular as receitas à Maria e, sem esperar pela resposta, delega a responsabilidade de calcular as despesas a Pedro. Em seguida, podemos imaginar que Paulo fica no estado de "espera ocupada" (*busy wait*,

em inglês), aguardando que Maria e Pedro reportem os valores das receitas e despesas para que ele então possa calcular o valor do lucro líquido. Para tal, adicionamos a autodelegação de Paulo da ação de espera pelos resultados. Essa autodelegação é síncrona porque Paulo precisa aguardar seu final. A ação de espera pode ser entendida como a especificação do mecanismo de junção (diagramas de atividades) dos dois fios de execução disparados para cálculo das receitas e das despesas. A autodelegação de cálculo do lucro já estava na solução dada para o exercício.

10.5 Parâmetros das Chamadas

As mensagens que um objeto passa a outro são, em grande geral[1], chamadas que ele faz às operações públicas do outro objeto. A assinatura de uma operação, além de seu nome, consiste do tipo de retorno e dos parâmetros. Estes precisam ser especificados nas chamadas representadas no DS (além, é claro, de nos compartimentos de operações das classes, no modelo de classes de projeto).

Os tipos de retorno e os parâmetros das operações, incluindo seus tipos, são especificados nas assinaturas das operações e, portanto, nos rótulos das chamadas. Os rótulos das setas de chamadas ficam, com isso, muito longos, o que torna os DSs ainda mais complexos visualmente.

Com isso, durante a elaboração de um DS ficamos boa parte do tempo cuidando dos aspectos visuais do diagrama, aumentando ou diminuindo o espaçamento entre as linhas de vida dos objetos em função da extensão do rótulo das chamadas... Mas não há outro jeito.

Quando tratamos de responsabilidades, atributos e operações, mencionamos que, se designarmos alguém para executar uma tarefa e tivermos de falar muito, explicando como fazer e dando muitos detalhes do que queremos que seja feito, isso é uma possível evidência de que escolhemos a pessoa errada para fazer a tarefa. O "falar muito" e os "detalhes" a que nos referimos estão associados, no contexto de projeto de sistemas, ao número de chamadas e ao número de parâmetros presentes em cada chamada. Em outras palavras: se as assinaturas das operações são muito longas, você deve pensar na possibilidade de estar escolhendo os objetos errados para colaborar.

[1] Objetos podem se comunicar por meio de outros mecanismos, como lançamento e tratamento de interrupções, que não veremos nesse texto.

10.6 Quadros de Interação

Com os elementos da notação gráfica apresentados até aqui, você tem condições de elaborar diagramas de sequência próximos da realidade de desenvolvimento de sistemas computacionais. Mais recentemente, no entanto, a UML incorporou o conceito de *quadros*, adicionando recursos importantes que facilitam a especificação de colaborações.

Quadros de interação (*frames*, em inglês) definem uma ou mais regiões de um diagrama de sequência da UML onde representamos iterações (as repetições, laços ou loops), trechos concorrentes, trechos opcionais ou trechos alternativos de colaborações, dentre outros.

Quadros de interação foram introduzidos na UML na versão 2.0 e simplificaram bastante a especificação de repetições, condições (desvios) e concorrência, tornando os diagramas mais expressivos.

Com isso, DSs podem ficar mais completos, pois diminui a necessidade de representar um número menor de cenários por diagrama (você se lembra da recomendação que demos, de representar apenas um cenário por diagrama?).

Os quadros são rotulados com nomes de operadores, colocados no pequeno compartimento acima e à esquerda no quadro. As regiões que compõem os quadros, quando existe mais de uma, chamam-se fragmentos.

A Tabela 10.1 relaciona os operadores comumente usados em quadros de interação.

Se quisermos, por exemplo, representar que as mensagens em determinado trecho de uma colaboração sejam repetidas um certo número de vezes, envolvemos (*emolduramos*) o conjunto de mensagens por um quadro `loop`, conforme ilustrado na Figura 10.9. Nesse caso, a chamada `m1` e a autodelegação `m2` são repetidas *n* vezes.

Caso queiramos especificar uma condição, usamos quadros `opt` para uma única condição (um único fragmento), ou `alt` para múltiplas condições alternativas (mais de um fragmento), conforme a necessidade. A Figura 10.10 ilustra a situação em que a chamada `m1` é feita se a condição `c1` for verdadeira.

A Figura 10.11 ilustra a situação em que, caso a condição `c1` seja verdadeira, a chamada `m1` é enviada; caso contrário, a chamada `m2` é enviada.

Você pode notar que a condição que controla o número de vezes que a colaboração que está no interior do quadro é executada (Figura 10.9) e as condições que especificam a colaboração opcional e as colaborações alternativas (Figuras 10.10 e 10.11, respectivamente) são colocadas entre "`[`" e "`]`".

Os quadros são contêineres da UML. Sendo assim, a partir do momento em

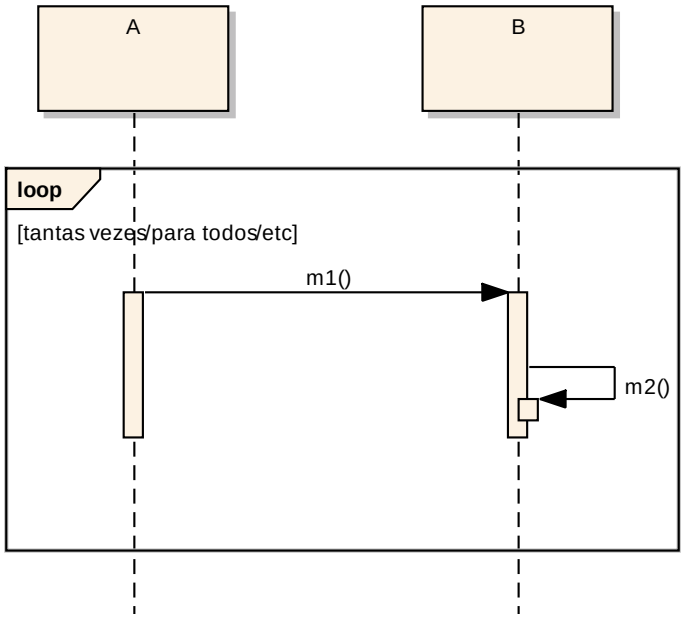


Figura 10.9: Quadro `loop` para especificar repetições.

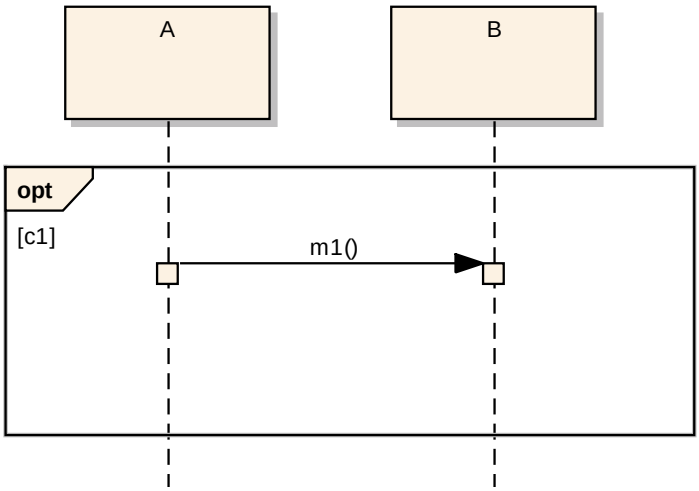


Figura 10.10: Colaboração opcional especificada por um quadro `opt`.

Tabela 10.1: Operadores comumente usados em quadros de interação.

Operador	**Significado**
`alt`	Especifica múltiplos fragmentos alternativos em um quadro. A condição em que um fragmento é executado é colocada no topo do fragmento entre colchetes. Os fragmentos do quadro são separados por linhas tracejadas, com as condições em que são executados em seus topos entre colchetes.
`opt`	Especifica um quadro executado opcionalmente. O quadro corresponde a um único fragmento. A condição em que o fragmento é executado é colocada no topo do quadro, entre colchetes.
`par`	Especifica múltiplos fragmentos executados concorrentemente.
`loop`	Especifica um quadro (um único fragmento) executado iterativamente.
`ref`	Especifica um rótulo para um quadro, que pode ser referenciado em outro lugar ou em outro diagrama, como uma chamada.
`sd`	De "*sequence diagram*", que opcionalmente rotula um quadro que contém totalmente um diagrama de sequência.

que se coloca um trecho de colaboração dentro dele, todo o trecho está logicamente associado ao quadro. Sendo assim, se eliminamos o quadro do diagrama, toda a colaboração que ele contém é eliminada do modelo; se movemos o quadro, toda a colaboração em seu interior se move junto, e assim por diante. Isso é o que se espera que aconteça no CASE, mas, obviamente, dependerá de como e se a ferramenta implementa o conceito de quadros conforme especifica a UML.

A notação usando quadros tem vantagens e desvantagens em relação à versão anterior da UML, usando guardas, que relacionamos a seguir:

- os quadros agregam complexidade visual ao modelo;
- os quadros proveem maior expressividade ao modelar blocos aninhados correspondendo a iterações e condicionalidades;
- o uso de guardas resulta em menor complexidade visual, embora os rótulos das mensagens fiquem mais extensos;
- o uso de guardas provê menor expressividade ao modelar blocos aninhados correspondendo a iterações e condicionalidades.

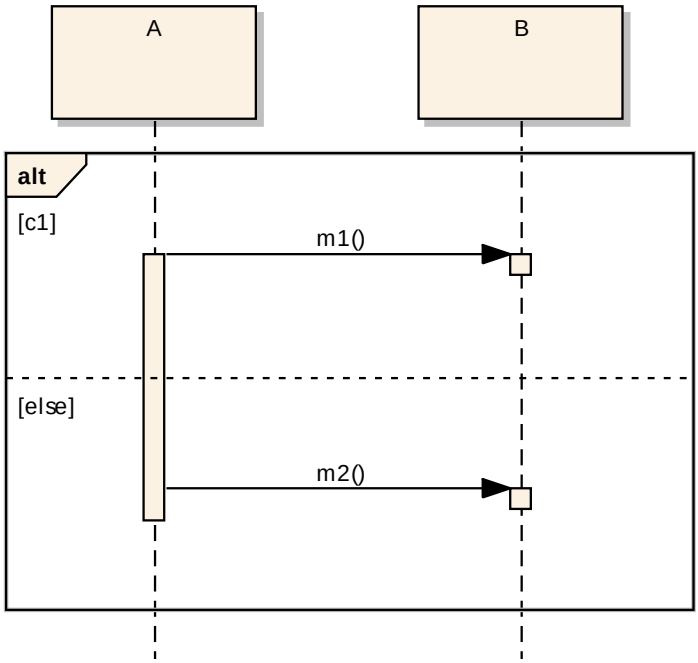


Figura 10.11: Colaborações alternativas especificadas por um quadro `alt`.

O uso de uma ou de outra notação (desconsiderando-se que as mensagens com guardas não fazem mais parte da notação da UML, apesar de ainda muito usadas) depende da situação, ficando a critério do time de modelagem.

10.7 Falando Um Pouco Mais Sobre a Criação de Objetos

Uma pergunta que frequentemente me fazem diz respeito a quando criar os objetos em uma colaboração. A resposta que costumo dar é: "o mais tarde possível". Vou justificar minha resposta usando novamente como ilustração a instanciação de um novo cliente (cadastramento) pelo caso de uso `Cadastrar Cliente`. Nesse caso, a pergunta é: quando instanciar o objeto `cliente` durante a realização do caso de uso para armazenar os dados coletados do formulário de cadastramento? A razão para instanciarmos o objeto `cliente` o mais tarde possível na colaboração é bem simples: diminuir a probabilidade de termos de destruir o objeto, caso encerremos o caso de uso em uma situação de insucesso. Criar e destruir objetos provoca o que se chama fragmentação da memória, que torna a resposta do sistema mais demorada, pode

provocar falha na criação de novos objetos e pode "despertar" o processo coletor de lixo. Esse processo rearruma o espaço livre da memória, reagrupando os fragmentos livres de memória como se fosse uma "arrumação" de casa. Certamente isso leva tempo, demanda CPU e deixa o computador mais lento para o usuário.

10.8 Objetos Controladores

Muitos projetistas optam por atribuir a um conjunto reduzido de objetos as responsabilidades de controlar o fluxo de mensagens e/ou de zelar pela obediência às regras de negócio. Esses objetos são instanciados de classes concebidas exclusivamente para isso, não fazendo parte do conceito; são, portanto, classes de projeto. Há inclusive projetistas que retiram quase todas as responsabilidades das classes conceituais, deixando para elas a atribuição exclusiva de armazenar os atributos e prover as operações *get*/*set* (as de obtenção e de definição dos atributos das classes, respectivamente).

Essas classes recebem o nome de *classes controladoras*. O objetivo de criarmos classes assim é o de concentrarmos a lógica da aplicação em poucas classes, facilitando com isso a manutenção do sistema. Eu, particularmente, acho suficiente criar uma classe controladora por caso de uso para a maioria dos casos de uso.

Quem opta pelo uso de classes controladoras desenvolve diagramas de sequência em que as mensagens partem exclusivamente desses objetos para si próprios (autodelegações) ou, na quase totalidade das demais mensagens, usando mensagens de *get*/*set* endereçadas aos objetos de classes conceituais.

As classes controladoras e seus relacionamentos devem constar do diagrama de classes de projeto.

10.9 Objetos de Interface

Outros objetos muito frequentemente usados são os objetos de interface. As interfaces usualmente proveem meios de comunicação entre ambientes distintos, como, por exemplo, entre dois sistemas, entre dois subsistemas, entre duas tecnologias diferentes e, em muitos casos, entre os usuários e os sistemas.

Neste caso se enquadram os formulários (as telas), sejam eles as telas das aplicações *web* (os navegadores ou *browsers*) ou das aplicações *desktop* (formulários Visual Basic, Delphi etc.).

As telas são instanciadas quando o caso de uso se inicia, exibindo os campos

de entrada e saída de dados e os demais controles, como botões, barras de rolagem etc. Elas são instâncias de suas respectivas classes (um formulário VB é instanciado da classe `Form` do VB, por exemplo).

Os formulários proveem, portanto, a única via de acesso possível do usuário aos objetos do sistema. Normalmente representamos nos DS os atores de um lado do formulário e os objetos do sistema do outro.

Entre os atores e os formulários são trocadas mensagens que representam os eventos externos, disparados pelos atores, e as respostas do sistema a esses eventos. Essas mensagens e respostas devem corresponder aos passos contidos nas especificações dos casos de uso.

Usualmente, do outro lado do objeto formulário colocam-se os objetos instanciados das classes controladoras, em uma posição mais próxima), das classes conceituais e das demais classes de projeto.

Objetos formulários podem instanciar outros objetos formulários (as telas *popups*, por exemplo) durante uma colaboração (uma interação) com o usuário.

Não é boa prática, no entanto, colocar regras de negócio e atribuir responsabilidade de controle de interação aos formulários. Costumo dizer que os formulários devem ser "burros", e que a "inteligência" do sistema deve ficar a cargo dos objetos controladores e dos objetos das classes conceituais (muitos projetistas diriam "quando muito" nesse ponto, porque até as classes conceituais muitos optam por mantê-las "burras"). Por essa razão, assim que um formulário recebe o controle (o usuário pressiona o botão "*Ok*", por exemplo), deve passá-lo, o mais rapidamente possível, para o objeto controlador que está ao seu lado no DS.

Objetos de interface são bastante usados para diminuir o acoplamento entre duas partes distintas de um sistema, provendo com isso maior manutenibilidade (capacidade ou facilidade de sofrer manutenção) a ambos os sistemas. Assim com as classes controladoras, as classes de interface e seus relacionamentos devem constar do diagrama de classes de projeto.

Desenvolver DSs complexos é considerado trabalho jogado fora por muitos projetistas. Por essa razão, o procedimento usado por muitos deles é iniciar a realização dos casos de uso especificando a interação em mais alto nível usando diagramas de sequência. Em seguida, geram um "código inicial" utilizando os recursos de engenharia direta disponível na ferramenta CASE. A partir daí, deixam a cargo dos experientes construtores do sistema (se não fazem, eles mesmos, a construção) a codificação "à mão" do restante do sistema.

Eventualmente, quando o sistema está pronto e testado, eles promovem a engenharia reversa do código para a recomposição/complementação do diagrama de classes.

10.10 Resumo do Capítulo

Os objetos colaboram para a realização das funcionalidades do sistema que estamos construindo. Essa colaboração é coordenada por uma sequência programada de ações e mensagens que os objetos trocam entre si. Esses são os programas.

Os diagramas de sequência permitem especificar quais objetos participam das colaborações e quais são as sequências de ações executadas e de mensagens passadas entre os objetos. As mensagens trocadas entre os objetos são dos tipos chamadas síncronas e assíncronas, de criação e de destruição de objetos e mensagens retorno.

As chamadas correspondem à invocação de alguma operação de um objeto feita por outro objeto ou, eventualmente, de um objeto por ele mesmo e são representadas por setas com pontas fechadas que partem do objeto que solicita a execução de algo e chegam ao que executa o que é solicitado.

É possível representar a criação e a destruição de objetos. A instanciação corresponde à execução do método construtor do objeto chamado e é denotada por uma seta tracejada com a ponta aberta. A destruição de um objeto pode ser comandada por outro objeto, que chama seu método destrutor. A mensagem de destruição é representada igualmente por uma seta tracejada de ponta aberta. Nesse ponto, a linha de vida do objeto destruído se encerra, pois, em se tratando de objetos, não há vida após a morte!

Outro tipo de mensagem é a de retorno da resposta de uma chamada, com consequente retorno do controle de execução. Mensagens de retorno são representadas por setas tracejadas de ponta aberta, no sentido oposto à chamada. O retorno de uma chamada sempre acontece para o objeto que fez a chamada, nunca para outro objeto qualquer.

As chamadas podem ser síncronas e assíncronas. Nas chamadas síncronas, o objeto que chama espera pelo término do processamento da mensagem pelo objeto chamado. Nas assíncronas, o objeto que chama não aguarda o término do processamento da mensagem para poder fazer outra chamada a outro objeto, dando origem a múltiplas ativações (múltiplos fios de execução). Chamadas síncronas são representadas por setas sólidas (não tracejadas) de pontas fechadas, enquanto chamadas assíncronas são representadas por setas sólidas de pontas abertas.

As chamadas podem passar parâmetros aos objetos chamados, que são especificados nas assinaturas das operações. Os quadros de interação definem uma ou mais regiões de um DS onde representamos iterações (as repetições, laços ou *loops*), trechos concorrentes, trechos opcionais ou trechos alternativos de colaborações, dentre outros, simplificando bastante suas representações em relação às formas adotadas nas versões da UML anteriores à versão 2.0. A notação usando quadros tem

vantagens e desvantagens em relação à versão anterior da UML, usando guardas. O uso de uma ou de outra notação, desconsiderando que as mensagens com guardas não fazem mais parte da notação da UML (apesar de ainda muito usadas), depende da situação, ficando a critério do time de modelagem.

Muitos projetistas optam por atribuir a um conjunto reduzido de objetos as responsabilidades de controlar o fluxo de mensagens e/ou de zelar pela obediência às regras de negócio. Esses objetos "de projeto" são instâncias de classes que recebem o nome de classes controladoras. O objetivo de criar classes assim é concentrar a lógica da aplicação em poucas classes, facilitando com isso a manutenção do sistema.

Objetos de interface proveem meios de comunicação entre ambientes distintos, como , por exemplo, entre dois sistemas, entre dois subsistemas, entre duas tecnologias diferentes e, em muitos casos, entre os usuários e os sistemas. Objetos de interface são bastante usados para diminuir o acoplamento entre duas partes distintas de um sistema, provendo, com isso, maior manutenibilidade.

Num tipo especial de objetos de interface se enquadram as telas das aplicações *web* ou os formulários das aplicações *desktop* que proveem a única via de acesso possível do usuário aos objetos do sistema. Os formulários devem ser "burros", e a "inteligência" do sistema deve ficar a cargo dos objetos controladores e dos objetos das classes conceituais.

10.11 Exercícios Propostos

1. Retorne à empresa ZYX. Quando um pedido é colocado diretamente por um cliente que usa o sítio da ZYX na Internet, ele especifica os itens que compõem o pedido, colocando-os no "carrinho de compras". É possível que o cliente adicione no carrinho itens que não estão disponíveis no estoque no momento do pedido, até porque a ZYX pode querer não trabalhar com determinados itens em estoque. Isso certamente atrasará a entrega do pedido pelo prazo demandado para a entrega do produto à ZYX pelo fornecedor. Podemos assumir que o prazo e o preço de fornecimento são atributos do relacionamento entre produto e fornecedor do produto.

   O diagrama de classes a ser considerado é o ilustrado na Figura 10.12, que é o apresentado na Figura 5.2 com a adição da classe de associação `Fornecimento` para acomodar os atributos e as operações relativas ao fornecimento de produtos à ZYX.

   Repare na Figura 10.12 que um mesmo produto pode ter qualquer número de fornecedores, mas as condições de fornecimento de um produto por um determinado fornecedor são únicas.

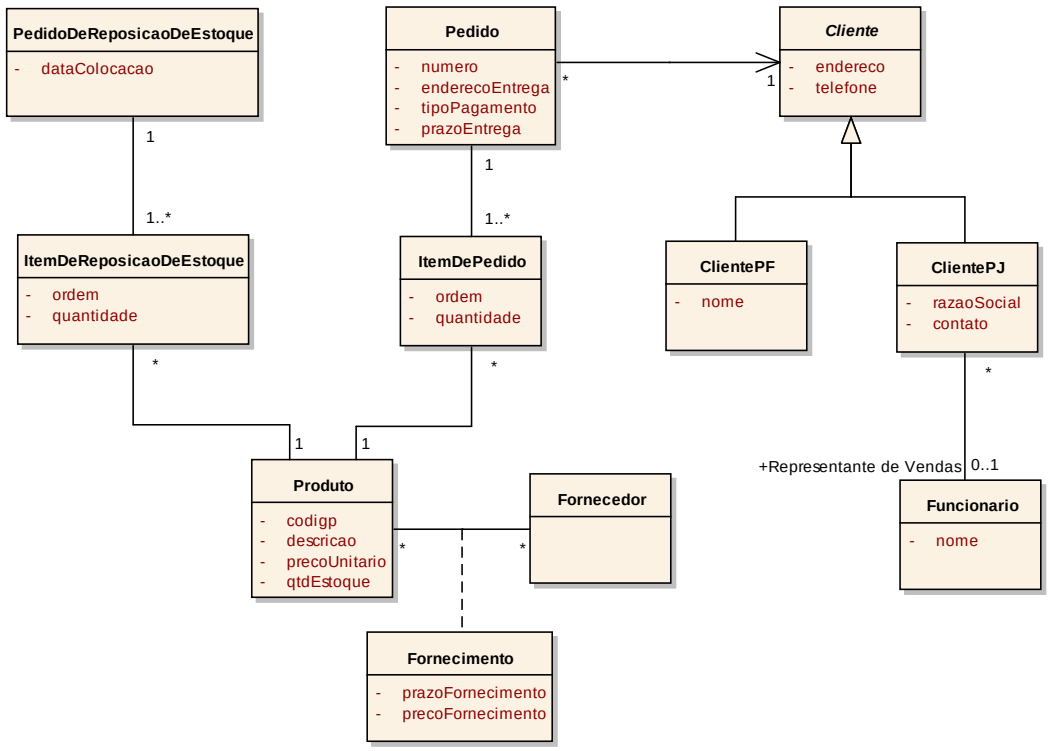


Figura 10.12: Diagrama de classes da ZYX contemplando prazo e preço de fornecimento.

Especifique a colaboração necessária para obter o prazo de entrega de um determinado item, supondo que não há quantidade suficiente em estoque e que o prazo é o do primeiro da lista dos fornecedores para o mesmo produto. Represente as operações e seus parâmetros, assim como as visibilidades das operações.

2. Estendendo o trecho do descrito no Exercício 1, vamos especificar, agora, as seguintes situações:

   - o produto se encontra no estoque. Nesse caso, o prazo de fornecimento é zero e é comandada a retirada da quantidade demandada do estoque; ou
   - precisamos localizar o fornecimento com o menor preço para a situação em que o produto não se encontra em estoque. O prazo de fornecimento é o valor do atributo `prazoFornecimento` da classe `Fornecimento` para o referido fornecedor.

Tabela 10.2: Tabela de descrição parcial do caso de uso `Efetuar Pedido`.

Caso de Uso:	Efetuar Pedido
Propósito:	Registrar pedidos feitos por cliente da ZYX via *web*.
Descrição geral:	Os clientes da ZYX que se autenticam no sistema podem efetuar seus pedidos diretamente pela internet colocando no carrinho de compras que representa o pedido os itens que deseja comprar e pagando com cartão de crédito.
Atores:	Cliente, operadora de cartão de crédito.
Pré-condições:	Cliente possui *login* e senha de acesso no sistema.
Pós-condições:	Caso sucesso: pedido e seus itens registrados no sistema.
Curso Típico (CT)	
1. Sistema exibe os campos para autenticação do cliente e o botão "Ok"; 2. Cliente fornece login e senha e pressiona o botão "Ok"; 3. Sistema autentica cliente, obtém seu nome e endereço e os exibe no topo do formulário; 4. Para todos os itens que compõem o pedido do cliente: a. Sistema exibe os campos para fornecimento do código do produto, a quantidade desejada e os botões "Adicionar ao carrinho de compras" e "Finalizar pedido"; b. Cliente fornece o código do item, quantidade e clica no botão "Adicionar ao carrinho de compras"; c. Sistema busca a descrição do produto e o preço unitário; d. Sistema exibe a descrição do produto, o preço unitário, calcula o preço total do item e o exibe; e. Sistema atualiza o valor total do pedido; 5. ...	
Cursos Alternativos (CA)	
CA 1: Passo 3 do CT – Sistema não autentica cliente	
1. Sistema informa que login/senha não constam do cadastro. ** Fim do Caso de Uso **	

3. Desenvolva o DS para os passos do caso de uso `Efetuar Pedido` especificados na Tabela 10.2. Considere que no diagrama de classes do Exercício 1 adicionamos a classe `ColecaoDeClientes` associada à classe `Cliente` para a indexação de clientes, conforme vimos no Capítulo 9.

   Use um objeto controlador para controlar os passos da interação e um ou mais objetos de interface para representar formulários. Você poderá adicionar outras classes que eventualmente julgar necessárias ao diagrama.

As soluções encontram-se a partir da Página 229.

CAPÍTULO 11

DIAGRAMAS COMPLEMENTARES

> Nothing ends nicely, that's why it ends.
>
> Tom Cruise

Dos treze diagramas oficiais da UML, até aqui apresentamos os cinco mais usados em modelagem de sistemas computacionais. Existem, no entanto, outros quatro diagramas que podem ajudar bastante na especificação de determinados aspectos das dimensões temporal e estrutural do sistema e na organização dos modelos.

Neste capítulo apresentaremos os principais conceitos e a notação gráfica dos diagramas de visão geral da interação, de pacotes, de componentes e de instalação.

É natural que você estranhe o fato de apresentarmos quatro diagramas em um único capítulo, quando precisamos de oito capítulos para apresentar cinco diagramas. Isso só é possível por duas razões que podem ocorrer concomitantemente, dependendo do diagrama:

1. os conceitos são simples;
2. os conceitos já foram vistos nos outros diagramas e se combinam, de alguma forma, nesses quatros, ou seja, os diagramas a serem apresentados compartilham os conceitos associados aos demais diagramas de que já tratamos.

Trataremos inicialmente dos diagramas de visão geral da interação, que podem ser entendidos como combinações entre diagramas de atividade e de sequência, conforme veremos a seguir.

11.1 Diagramas de Visão Geral da Interação

Como já tratamos de diagramas de atividades e de diagramas de sequência, podemos definir, de forma bastante satisfatória, diagramas de visão geral da interação como sendo uma combinação entre esses dois. Essa combinação se dá pela colocação de quadros de interação dos diagramas de sequência nos lugares das caixas de ações dos diagramas de atividades.

Um diagrama de visão geral da interação também pode ser visto como sendo uma fragmentação de diagramas de sequência, com esses fragmentos de mais baixo nível (as pequenas colaborações entre objetos contidas em quadros de interação) "costurados" com o uso dos elementos que compõem a visão de mais alto nível da estrutura de controle (fluxos, decisões, intercalações, separações e junções) dos DAs.

Os diagramas de visão geral da interação usam interações (quadros com trechos de diagramas de interação) e/ou ocorrências de interações, ou seja, referências para outros quadros com trechos de diagramas de interação dispostas segundo uma estrutura de controle de mais alto nível. Eles servem, portanto, para detalhar, em um só diagrama, os passos da colaboração entre objetos necessários para a realização de cada ação de uma atividade.

Vamos ilustrar o que acabamos de expor imaginando que temos uma determinada atividade especificada pelo DA da Figura 11.1.

Caso queiramos especificar como cada ação da atividade da Figura 11.1 é realizada por meio de colaborações entre objetos, desenvolvemos um diagrama de visão geral da interação, que pode ter a forma do diagrama ilustrado na Figura 11.2.

Note que, por um lado, podemos entender que o diagrama da Figura 11.2 é um refinamento do diagrama da Figura 11.1. Por outro, podemos ver o diagrama da Figura 11.2 como sendo uma fragmentação do diagrama de sequência necessário para especificar todos os cenários de uma atividade com vários quadros de interação colocados da sequência apropriada com o uso dos elementos de controle de fluxo dos DAs.

Podemos usar todos os elementos gráficos dos DAs, incluindo os de controle do fluxo, como decisões (já vistas nas Figuras 11.1 e 11.2), separações, junções etc., nos diagramas de visão geral da interação.

Embora estes diagramas sejam bastante recentes (passaram a fazer parte da

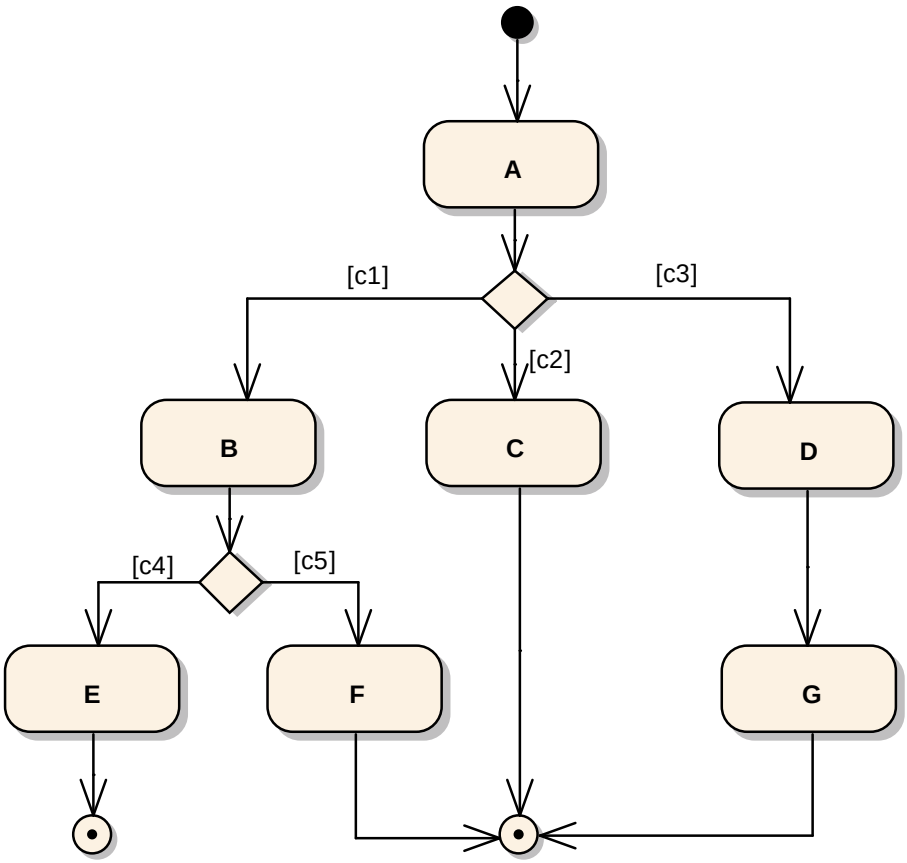


Figura 11.1: Diagrama de atividade especificando uma atividade hipotética.

especificação da UML apenas em sua versão 2.0), já podemos antever suas aplicações em projetos de software que usam geração de código com base nos modelos UML, em que apenas a utilização de diagramas de sequência é inviável, em função da complexidade visual resultante ao se modelar mais de um cenário em um mesmo DS.

Também por serem recentes, pode não haver suporte gráfico completo para esses diagramas pelo CASE que você usa. Mas isso não é um problema sério, na medida em que você pode construir um DS para cada ação do DA sendo detalhada. Aliás, essa é uma opção quando o diagrama de visão geral da interação adquire uma complexidade visual que o impede de caber em uma página. A outra opção, lembre-se, é o uso de conectores dos DAs.

A recomendação que deixamos é procurar tornar as ações tão simples quanto

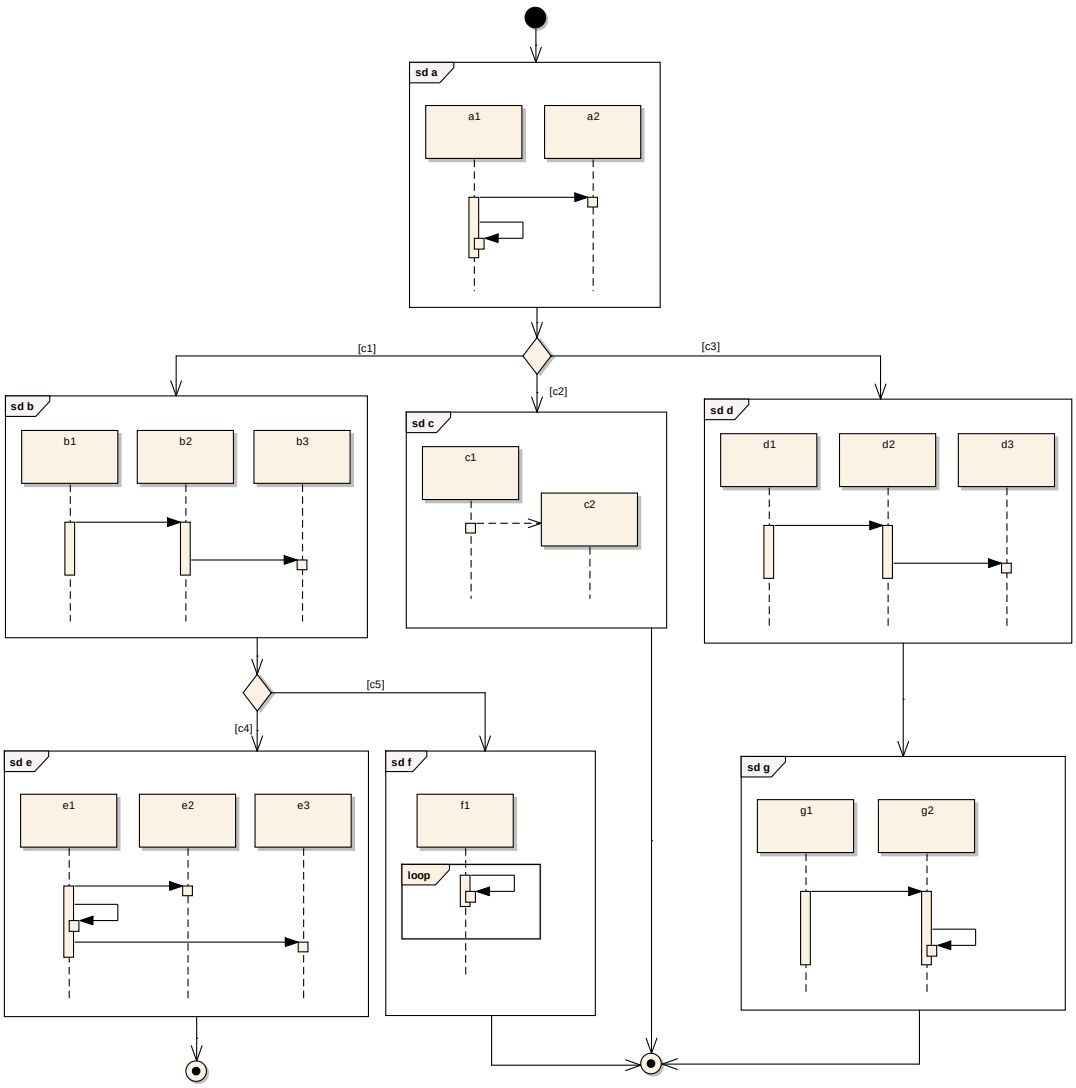


Figura 11.2: Diagrama de visão geral da interação, especificando as colaborações necessárias para a realização das ações do DA da Figura 11.1.

for possível, atômicas (indivisíveis) de preferência, objetivando obter quadros de interação visualmente simples.

11.2 Diagramas de Pacotes

Pacotes são contêineres usados para agrupar certos elementos da notação, provendo um espaço de nomes para os elementos agrupados e permitindo que organizemos o modelo, dividindo-o em partes relacionadas entre si para sua melhor visualização e compreensão.

Pacotes têm a forma gráfica de uma pasta suspensa, ou seja, um retângulo com sua etiqueta em um dos cantos. O nome do pacote é usualmente colocado no centro do retângulo, no topo ou dentro da etiqueta.

Dentro do espaço de nome, os nomes dos elementos gráficos (nomes das classes, dos atores etc.) têm de ser distintos, pois precisam identificar univocamente os elementos a que se referem dentro desse espaço.

Apenas elementos empacotáveis de um modelo podem ser agrupados em pacotes nesse modelo. A definição de *elemento empacotável* que consta na UML, acreditem, é: "elementos empacotáveis são elementos que possuem nomes e que podem ser empacotados".

Os elementos mais comumente agrupados com o uso de pacotes são classes, atores, casos de uso e outros pacotes. Pacotes contendo outros pacotes permitem formarmos hierarquias de pacotes.

Como dissemos, o uso de pacotes facilita a organização e a compreensão de um modelo, agrupando os elementos dos diagramas conforme suas afinidades semânticas e tornando os diagramas menos complexos.

Podemos, por exemplo, organizar os atores de um sistema em pacotes. Um pacote poderia ser o de atores administradores do sistema, ou seja, aqueles atores que exercem funções de administração dos usuários, de fornecimento de parâmetros de execução do sistema em cada uma das suas áreas de utilização dentro da organização. Outro pacote poderia ser o de gerentes, que tipicamente usam o sistema para a realização de consultas, extração de relatórios, para dar prosseguimento a processos que demandam decisões importantes etc. A Figura 11.3 ilustra essa situação.

Podemos, ainda, organizar os casos de uso por subsistemas, incluindo no pacote Finanças os casos de uso relacionados ao subsistema de finanças, no pacote RH os casos de uso relacionados ao subsistema de gestão de pessoal etc.

Agrupar elementos em pacotes é feito nos CASEs usualmente de duas formas:

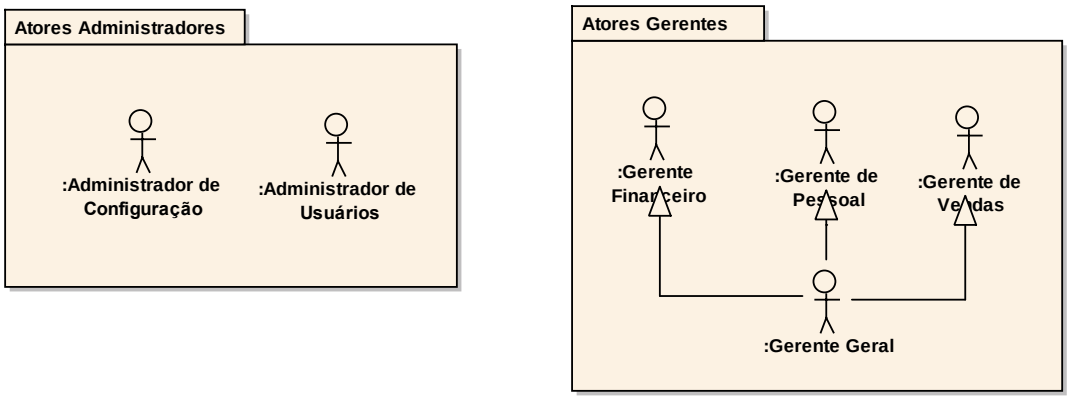


Figura 11.3: Uso de pacotes na organização dos atores de um sistema.

1. instanciar o pacote no formulário e arrastar o elemento para dentro do pacote instanciado, caso o elemento já exista; ou

2. instanciar o elemento no formulário já dentro do pacote.

É esperado que, ao mover ou deletar um pacote, todos os elementos em seu interior se movam ou sejam deletados, respectivamente. Retirar um elemento de dentro do pacote é usualmente feito arrastando-se o elemento para fora do pacote.

Após agrupar os elementos em pacotes e, possivelmente, organizar os pacotes segundo hierarquias, é sempre interessante ilustrar essa organização em um ou mais diagramas de pacotes que, colocados na documentação idealmente antes dos demais diagramas, mostram os pacotes e seus relacionamentos de dependência (relacionamentos de uso).

A Figura 11.4 ilustra um diagrama de pacotes representando, além dos dois pacotes, um relacionamento de dependência mútua entre eles. Um relacionamento de dependência (seta tracejada de ponta aberta) é lido na direção da seta como "usa" ou "depende de". O diagrama de pacotes da figura poderia representar, por exemplo, a situação em que classes de conceito contidas no pacote `Pagamentos` estão associadas com classes de conceito contidas no pacote `Finanças` e há navegabilidades e chamadas de operações em ambos os sentidos.

Como vimos no final do Capítulo 10, com o intuito de diminuir o acoplamento entre os pacotes, os acessos a recursos de outros pacotes são usualmente feitos através de classes de interface. Isso aumenta a manutenibilidade de cada subsistema

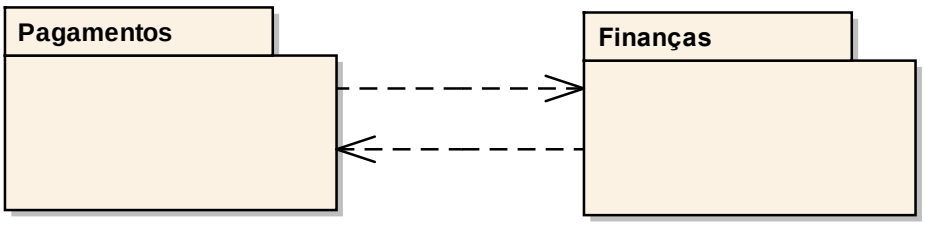


Figura 11.4: Dependência entre pacotes em um diagrama de pacotes.

e, com isso, do sistema como um todo. Sendo assim, podemos representar os relacionamentos da Figura 11.4 de forma alternativa, como é ilustrado na Figura 11.5.

É comum os projetistas incluírem cada classe de interface em seu respectivo pacote. Com isso, a classe `AcessoFinancas` faria parte do pacote `Finanças` e a classe `AcessoPagamentos` faria parte do pacote `Pagamentos`. Outra prática menos frequente, por conta de características de algumas linguagens, é criar-se um pacote (por exemplo `Interfaces`) para agrupar todas as classes de interface de um sistema.

11.3 Diagramas de Componentes

Ao contrário de um pacote, que de forma geral corresponde a um agrupamento lógico de elementos do modelo, um componente corresponde a um agrupamento físico de elementos do modelo que provê um conjunto de funcionalidades. Assim sendo, componentes são também contêineres.

Os componentes em um sistema têm a ver com como projetamos e implementamos o sistema, ou seja, com a arquitetura do sistema.

Os diagramas de componentes mostram a estrutura do modelo de implementação, sendo atemporal, portanto. Um diagrama de componentes é apresentado como um conjunto de símbolos que representam os componentes, conectados entre si por meio de relacionamentos de dependência e comumente usando classes de interface para intermediar a comunicação entre os componentes.

Um componente pode corresponder a um subsistema ou a qualquer outra porção reutilizável de um sistema, "mostrando" apenas o que é relevante do ponto de vista do restante do sistema e encapsulando o resto.

Por sinal, a possibilidade de reutilização de porções de um sistema em muitos casos orienta a *componentização* desse sistema, ou seja, seu desenvolvimento por

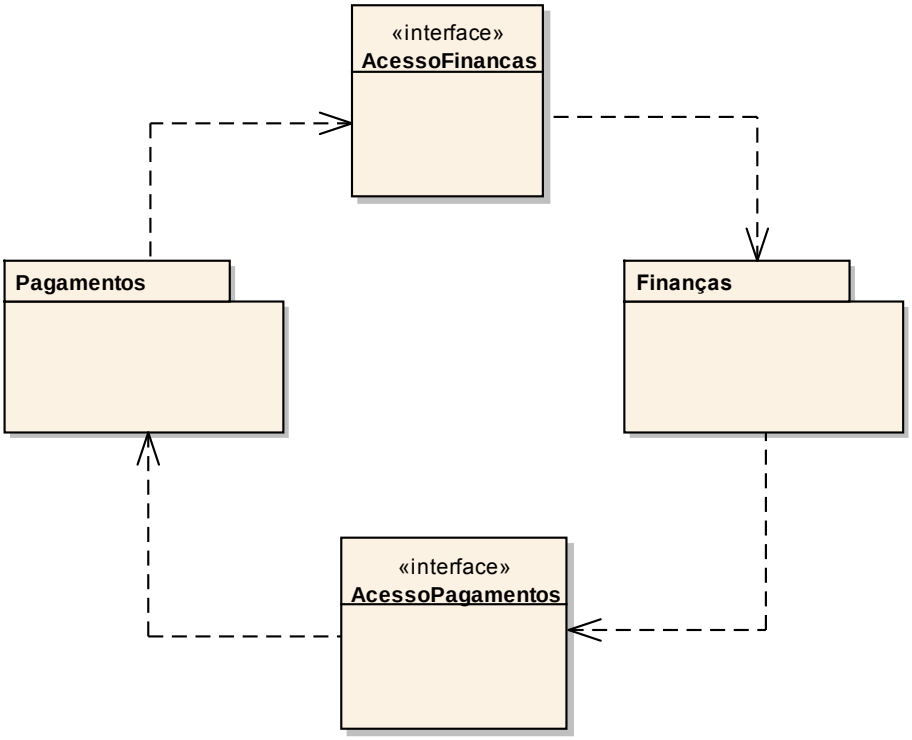


Figura 11.5: Dependência entre pacotes realizada através de classes de interface.

blocos internamente coesos e interconectados entre si (a chamada *orientação a componentes*).

As bibliotecas de vínculos dinâmicos do Windows – DLLs – são exemplos de componentes para os quais especificamos as interfaces de uso e os conjuntos de funcionalidades. Para tal, em cada DLL temos um conjunto de classes que colaboram para a realização das responsabilidades da DLL.

Em termos de aplicação, exemplos de componentes comumente construídos nas organizações são:

- componentes para autenticação de usuários;
- componentes para tratamento de datas, de cadeias de caracteres e outras utilidades;
- componentes para manipulação de dados de clientes;

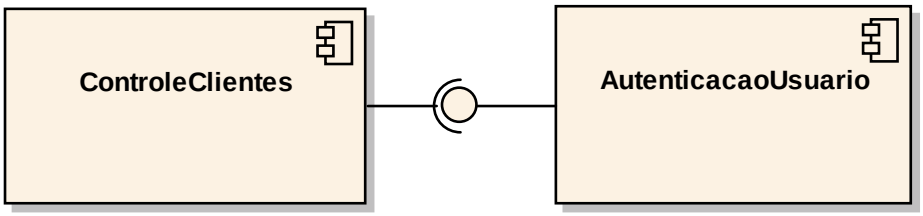


Figura 11.6: Notação gráfica de componentes, ilustrando as interfaces fornecida (pelo componente `AutenticacaoUsuario`) e exigida (pelo componente `ControleClientes`.

- componentes para manipulação de ordens de serviço;
- componentes para manipulação de faturas.

Componentes são utilizados para, basicamente:

- dividir o sistema em módulos;
- facilitar seus reúsos em outros sistemas;
- aumentar a manutenibilidade dos sistemas.

Um diagrama de componentes da UML deve especificar a organização das classes em agrupamentos físicos, podendo corresponder aos pacotes conforme foram logicamente organizadas.

A notação estabelecida pela UML 2.0 para os componentes em um diagrama de componentes é ilustrada na Figura 11.6. As classes que compõem cada componente, e eventualmente outros componentes, podem ser representadas no interior do componente.

A Figura 11.6 ilustra a comunicação entre componentes feita por meio de interfaces *fornecida* e *exigida*, ou seja, a interface que o componente provê (pequeno círculo aberto) e a interface que o outro componente usa (pequeno semicírculo externo ao da interface fornecida).

Relacionamentos de dependência nesses casos podem ser substituídos por interfaces fornecidas e exigidas conforme a Figura 11.7. Isso é devido ao fato de que o "pirulito" (é esse mesmo o nome que usualmente damos) representa a interface

Figura 11.7: Transformação de relacionamentos de dependência em notação de interfaces fornecidas e exigidas.

fornecida – portanto, a usada; o semicírculo representa a interface exigida – portanto, a que usa.

A orientação a componentes envolve a adoção de políticas específicas em uma organização, compreendendo inclusive a possibilidade de compra dos deles no mercado para agilizar o desenvolvimento de sistemas.

11.4 Diagramas de Instalação

Sistemas distribuídos são, por definição, aqueles que possuem arquitetura composta de partes de software que são processadas em nós de processamento distintos, eventualmente fisicamente distantes entre si. Para esses sistemas também necessitamos representar que partes do software processam em que partes do hardware e quais são os mecanismos de comunicação entre essas partes. Essa informação é colocada sob a forma de um ou mais diagramas de instalação da UML, usualmente em um documento do sistema chamado de *Documento de Arquitetura*.

Um diagrama de instalação consiste de um conjunto de caixas paralelepipédicas ligadas por associações de comunicação. As caixas representam os nós de processamento, e as associações usualmente indicam o protocolo usado para a comunicação. Os nós podem conter as representações dos componentes contidos neles, conforme o sistema foi/será implementado. As caixas representando os nós são, portanto, outros contêineres da UML.

A Figura 11.8 ilustra a representação da arquitetura de um sistema cliente-servidor em que o nó cliente se comunica com o nó servidor usando o protocolo HTTP.

A Figura 11.8 pode sugerir que diagramas de instalação são triviais. No entanto, em sistemas concebidos segundo arquiteturas complexas e pouco usuais (por exemplo, sistemas compostos de várias tecnologias distintas e sistemas integrando outros sistemas legados que demandam componentes de mediação entre eles), o

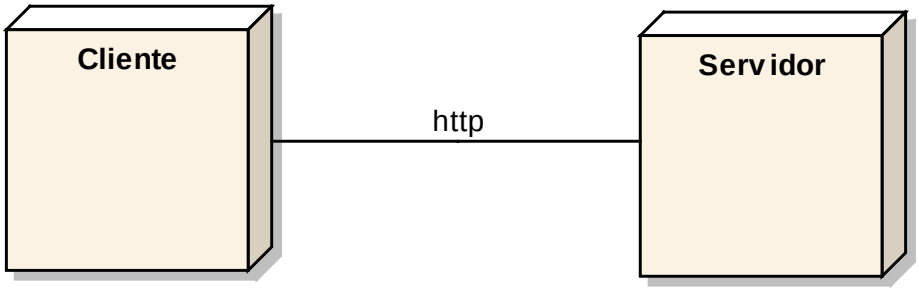


Figura 11.8: Sistema cliente-servidor com comunicação por HTTP.

diagrama pode ser bastante complexo. Nesses casos, diagramas de instalação podem ser bastante úteis para a compreensão do conjunto.

11.5 Palavras-Chave

Existe, ainda, um pequeno porém importante detalhe na notação que usamos ao longo de todo o texto, mas do qual não tratamos em nenhuma seção especificamente: as palavras-chave da UML.

A UML fornece uma notação gráfica para elementos estruturais (classes, casos de uso etc.), elementos comportamentais (interações e máquinas de estados), de agrupamento (pacotes, quadros), itens anotacionais (notas/comentários) e os blocos relacionais básicos (relacionamentos diversos), dentre outros.

Embora esse conjunto de elementos gráficos da notação seja grande, é de se esperar que a notação não atenda adequadamente a todas as necessidades de modelagem nos diversos domínios (modelagem de sistemas, modelagem de negócios etc.), sem que isso implique o crescimento sem-fim desse conjunto. Isso se dá pelo fato de esses elementos terem suas semânticas precisamente definidas, com funções específicas dentro dos grupos relacionados anteriormente. Qualquer necessidade de pequena flexibilização no significado de um símbolo impediria seu uso e, em consequência, o uso da linguagem.

Com isso em mente, o OMG definiu de antemão algo que pode ser entendido como um mecanismo de extensão da UML: as palavras-chave (*keywords*, em inglês).

Uma palavra-chave é usada quando, na construção de um modelo, se necessita de um elemento estrutural, comportamental, de agrupamento, anotacional ou de

relacionamento que não existe na UML mas que é semelhante a algo que já existe. Podemos pensar em palavras-chave como *subtipos* (especializações) que criamos ou que alguém já criou e estamos reusando para os tipos classe, associação, generalização etc. da UML. Com isso, podemos estender a semântica de um elemento da notação usando, no modelo, o elemento básico da UML que melhor se enquadra no que precisamos, rotulando-o com um texto – a palavra-chave – entre "«" e "»" que sintetize seu significado. Em seguida, precisamos definir, de forma não ambígua (preferencialmente usando a própria UML), o que significa a palavra-chave.

Por exemplo, um relacionamento básico da UML é o relacionamento de uso/dependência, representado graficamente pela seta tracejada com ponta aberta (ver seções 6.6, 11.2 e 11.3). Esse mesmo elemento gráfico pode ser rotulado, por exemplo, com «estende» e «inclui» para modificar o significado de uso ou dependência para inclusão de um caso de uso em outro de duas formas diferentes, conforme você viu na Seção 3.5 – Relacionamentos em Diagramas de Casos de Uso.

A UML já relaciona e especifica inúmeras palavras-chave que são de uso frequente da comunidade de modelagem[1]

Outras palavras-chave muito usadas (e que usamos em nossa aulas) são «ator», «entidade», «interface» e «singleton», para estender a semântica de classes em um modelo de classes, especificando respectivamente que uma dada classe é uma categoria de usuários, que uma dada classe é um conceito do negócio (é uma classe de entidade), que uma classe é de interface e, finalmente, que uma classe só pode ser instanciada uma única vez.

O conceito de palavras-chave foi separado do conceito de estereótipo (do qual não tratamos neste texto) a partir da versão 2.0 da UML. Muitos autores, no entanto, ainda se referem a palavras-chave como sendo estereótipos da UML.

11.6 Resumo do Capítulo

Existem outros quatro diagramas da UML que podem ajudar bastante na especificação de determinados aspectos das dimensões temporal e estrutural do sistema e na organização dos modelos. Os diagramas de visão geral da interação, de pacotes, de componentes e de instalação compartilham os conceitos associados aos cinco diagramas de que já tratamos.

Diagramas de visão geral da interação podem ser entendidos como combinações entre diagramas de atividade e de sequência. Essas combinações se dão pela colocação de quadros de interação dos diagramas de sequência nos lugares das caixas de ações

[1] Ver Tabela B1 no documento de superestrutura da UML

dos diagramas de atividades. Portanto, os quadros de interação que especificam as colaborações necessárias para as realizações das ações são "costurados" com o uso dos elementos que compõem a estrutura de controle dos DAs. A recomendação que damos é para tentar tornar as ações tão simples quanto possível, objetivando obter quadros de interação visualmente simples.

Pacotes são contêineres usados para agruparmos logicamente certos elementos da notação, provendo um espaço de nomes para os elementos agrupados e permitindo que organizemos o modelo, dividindo-o em partes relacionadas entre si para melhor visualização e compreensão. Os elementos mais comumente agrupados com o uso de pacotes são classes, atores, casos de uso e outros pacotes. Pacotes contendo outros pacotes permitem formar hierarquias de pacotes. O relacionamento de dependência entre pacotes deve ser representado nos diagramas e a boa prática recomenda estabelecer interfaces de comunicação entre os pacotes que se relacionam.

Um componente corresponde a um agrupamento físico de elementos do modelo que provê um conjunto de funcionalidades. Dessa forma, componentes são também contêineres. Um diagrama de componentes é apresentado como um conjunto de símbolos que representam os componentes, conectados entre si por meio de relacionamentos de dependência e comumente usando classes de interface para intermediar a comunicação entre os componentes. Os componentes em um sistema têm a ver com como projetamos e implementamos o sistema, ou seja, com a arquitetura do sistema.

Em sistemas distribuídos necessitamos representar que partes do software processam em que partes do hardware e quais são os mecanismos de comunicação entre essas partes. Essa informação é colocada sob a forma de um ou mais diagramas de instalação da UML, usualmente em um documento do sistema chamado de Documento de Arquitetura. Um diagrama de instalação consiste de um conjunto de caixas que representam os nós de processamento, ligadas por associações de comunicação que usualmente indicam o protocolo usado para a comunicação. Os nós podem conter as representações dos componentes contidos neles.

A UML fornece uma notação gráfica básica para os elementos estruturais, comportamentais, de agrupamento, itens anotacionais e os blocos relacionais básicos, dentre outros. Como a semântica de cada elemento e suas formas de emprego são precisamente definidas na especificação, a notação básica não atende adequadamente a todas as necessidades de modelagem nos diversos domínios. O OMG estabeleceu, de antemão, a possibilidade de uso das palavras-chave, que podem ser entendidas como mecanismo de extensão da UML. Uma palavra-chave é usada quando se necessita de um elemento estrutural, comportamental, de agrupamento, anotacional ou de relacionamento que não existe na UML, mas que é semelhante a algo que já existe. Você pode criar sua palavra-chave, caso necessite, mas precisa especificar, de maneira

inequívoca, seu significado e seu modo de emprego. A UML já relaciona e especifica no documento de superestrutura inúmeras palavras-chave que são de uso frequente da comunidade de modelagem.

11.7 Exercícios Propostos

1. No exercício três da Seção 10.11, apresentamos uma tabela com alguns passos da especificação do caso de uso `Efetuar Pedido` da nossa empresa ZYX.

   Desenvolva o diagrama de atividades que especifica graficamente o trecho do caso de uso e, com base nele e na colaboração representada no DS proposto como solução daquela atividade, desenvolva o diagrama de visão geral da interação para esse trecho do caso de uso. Considere, em princípio, que o diagrama de classes é o resultante da solução que demos para aquele exercício, mas você pode adicionar outras classes se julgar necessário.

2. Agrupe as classes do diagrama de classes da ZYX em sete pacotes: `Controle de Clientes`, `Controle de Pedidos`, `Controle de Fornecedores`, `Controle de Estoque`, `RH`, `Controle e Visão`. Elabore, também, um diagrama de pacotes em mais alto nível, mostrando as dependências entre os quatro pacotes e supondo que as navegabilidades não assinaladas são bidirecionais.

   O diagrama de classes é o da Figura 11.9.

3. Desenvolva um diagrama de distribuição do sistema da ZYX considerando que:

   - o sistema é *web*, ou seja, usa navegadores web como clientes do lado dos usuários e servidor de aplicação (onde os programas são executados) do outro lado;
   - a arquitetura do sistema ainda prevê um terceiro nó que processa o servidor de banco de dados (SGBD);
   - a comunicação entre os clientes e o servidor de aplicação é feita usando o protocolo HTTP;
   - a comunicação entre o servidor de páginas e de aplicação é feita usando-se *sockets*;
   - cada pacote que estabelecemos na Atividade 2 resultará na construção de um componente. Os componentes residirão no servidor de aplicação.

   Represente os componentes correspondentes aos pacotes (apenas esses pacotes, não considerando ainda as tecnologias de implementação e persistência) dentro do contêiner que representa o servidor.

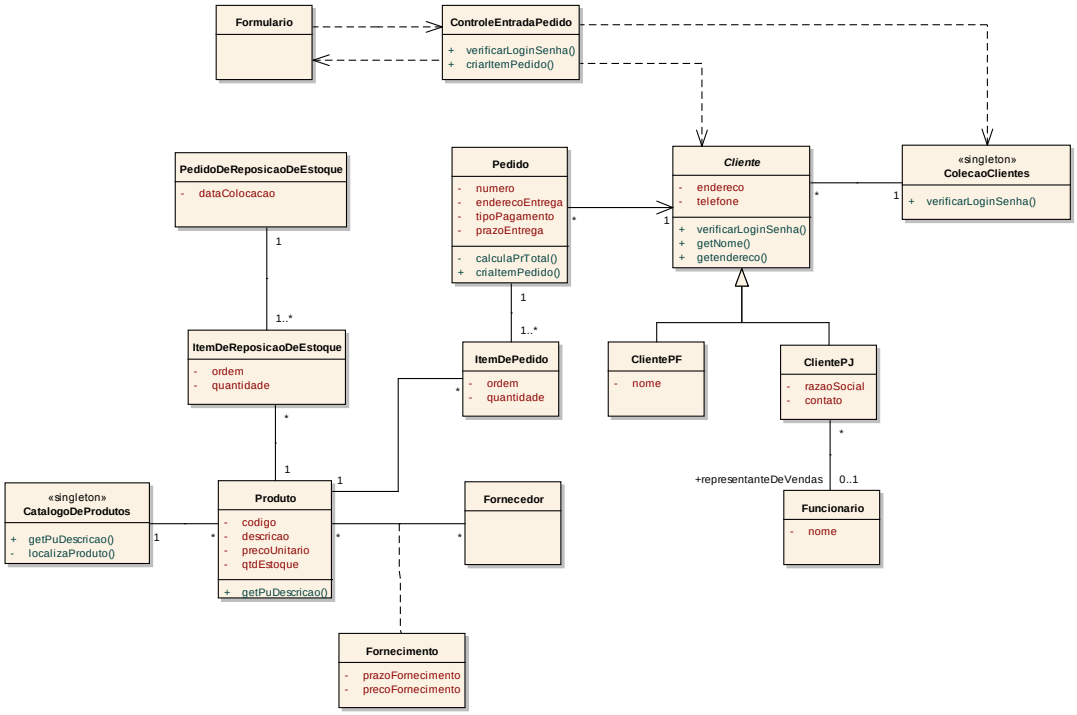


Figura 11.9: O diagrama de classes de especificação da ZYX.

As soluções encontram-se a partir da Página 235.

APÊNDICE A

SOLUÇÕES DOS EXERCÍCIOS PROPOSTOS

A.1 Exercícios do Capítulo 2, página 23:

1. Processos de software estabelecem antecipadamente as sequências das atividades, o que será produzido em cada atividade, os prazos, os custos, os pontos de controle, os níveis de qualidade e as demais variáveis associadas ao projeto de desenvolvimento de um sistema. Esses elementos servem de base para a gerência eficiente do projeto, que busca manter cada uma das variáveis dentro dos níveis de qualidade preestabelecidos para o software.

   A modelagem possibilita o estudo do sistema e a discussão de correções, modificações e validação com o cliente antes de o sistema ser construído – a um custo relativamente baixo, portanto. A modelagem facilita a comunicação entre os membros das equipes de análise e projeto e entre eles e os clientes e usuários. A modelagem permite estabelecer clara e inequivocamente o que é para ser feito e como fazer o sistema.

2. São várias as razões para usar a UML durante a análise e o projeto de sistemas OO. Citaremos algumas delas.

   A notação da UML é precisamente definida na especificação, o que significa que modelos construídos com ela não dão margem a múltiplos entendimentos, ou seja, à ambiguidade.

   A UML permite construir modelos completos, na medida em que é capaz

de capturar as dimensões funcional, estrutural e temporal que compõem os sistemas de informação.

Com o uso de software de suporte (CASE), disponíveis amplamente porque a linguagem é padronizada, podemos manipular facilmente os diversos elementos gráficos na notação, tornando a tarefa de modelagem uma atividade bastante dinâmica e efetiva.

A.2 Exercícios do Capítulo 3, página 41:

1. No primeiro caso (a), identifica-se uma única funcionalidade do sistema (abrir OS) para a qual existe um único ator: "atendente".

   No segundo caso (b), também se identifica uma única funcionalidade do sistema (Abrir OS) para qual também existe somente um ator (atendente), já que o cliente no balcão não interage com o sistema.

   O terceiro caso (c) é conceitualmente idêntico ao segundo. A única coisa que muda é a tecnologia: no caso anterior, a informação é impressa; nesse caso a informação vai por e-mail. Além disso, o supervisor não interage com o sistema que gera a OS, que é o sistema em estudo. Ele interage com outro sistema: o Outlook ou Gmail/Hotmail via Firefox ou algo no gênero.

   Para os três casos, portanto, a resposta é a mesma: o `Atendente` é o único ator.

2. No primeiro caso (a), identifica-se uma única funcionalidade do sistema: `Informar Conclusão de OS`.

   No segundo caso (b) existem duas funcionalidades. A primeira é `Acionar Alarme` (o ator é o `Sensor de Presença`) e a segunda é `Resetar Alarme` (você pode achar um nome melhor para essa função, que significa desligar a sirene e rearmar o sistema).

3. a) O Sistema de Contas a Receber (SCR) é informado (por uma mensagem eletrônica) da conclusão de uma OS. Colocamos o SCR como ator porque, em geral, sistemas precisam estar online, para que possam se comunicar. Se a comunicação for em *batch*, o SCR não participará como ator do caso de uso. Outro fator importante é que o caso de uso `Efetuar Cobrança` é caso de uso do SCR, e não do sistema que estamos modelando. Por isso ele não aparece no diagrama da Figura A.1.

   b) Já vimos casos semelhantes ao longo do texto: entrega de documentos impressos não caracterizam interação usuário-sistema. O trecho correspondente do diagrama fica como na Figura A.2.

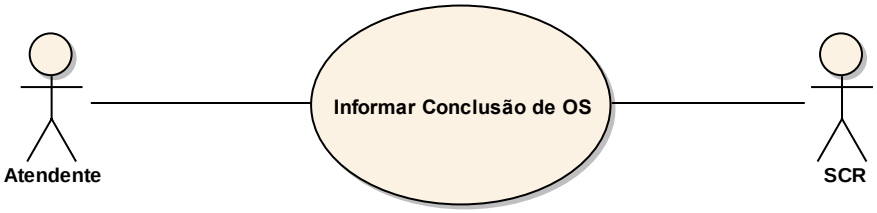


Figura A.1: O atendente, a conclusão das OS e o SCR.

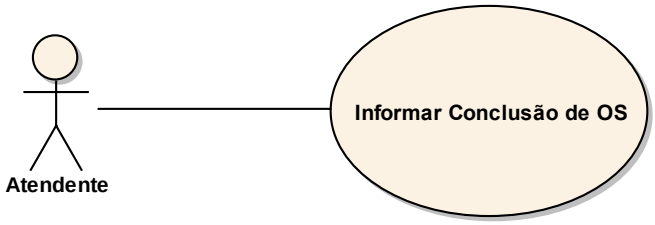


Figura A.2: A entrega de documentos impressos.

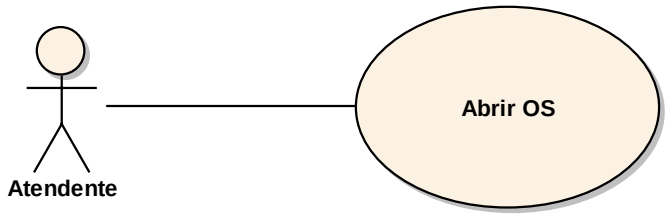


Figura A.3: Dados do cliente como passos da descrição do caso de uso.

c) A informação dos dados do cliente e do equipamento faz parte do que é feito no caso de uso, ou seja, de sua descrição. O trecho correspondente do diagrama fica como na Figura A.3.

d) A apresentação do formulário de cadastramento de um cliente e seu preenchimento pelo atendente são feitos dentro do caso de uso `Abrir`

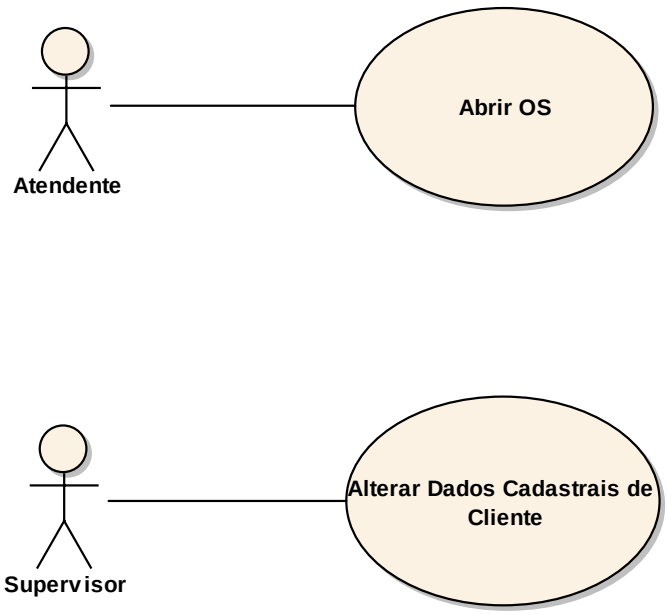


Figura A.4: Atores distintos participando de casos de uso distintos.

`OS`. O supervisor pode acessar o sistema para alterar os dados do cliente quando isso for necessário. Essa é outra funcionalidade do sistema. O diagrama fica, portanto, como o da Figura A.4.

e) A associação de um ator a um caso de uso, é importante lembrar, significa que essa funcionalidade está disponível para esse ator como uma funcionalidade do sistema. `Cadastrar Cliente` estende `Abrir OS` porque, durante a abertura de uma OS, pode ser necessário cadastrar o cliente se seus dados não constarem do cadastro. O diagrama fica conforme a Figura A.5.

f) Os dados de um cliente podem ser cadastrados pela recepcionista, como uma atividade a mais, opcional, durante a abertura de uma lista de exames. O diagrama fica conforme a Figura A.6.

g) Aqui há uma questão interessante: um sistema não pode ser ator de si mesmo, pois, afinal, não estamos interessados nos detalhes dos atores e sim do sistema em estudo (uma coisa não pode ser de interesse e de não interesse ao mesmo tempo). Eu mencionei que o sistema "dispara" a impressão do relatório de inadimplentes, mas na realidade quem faz isso é

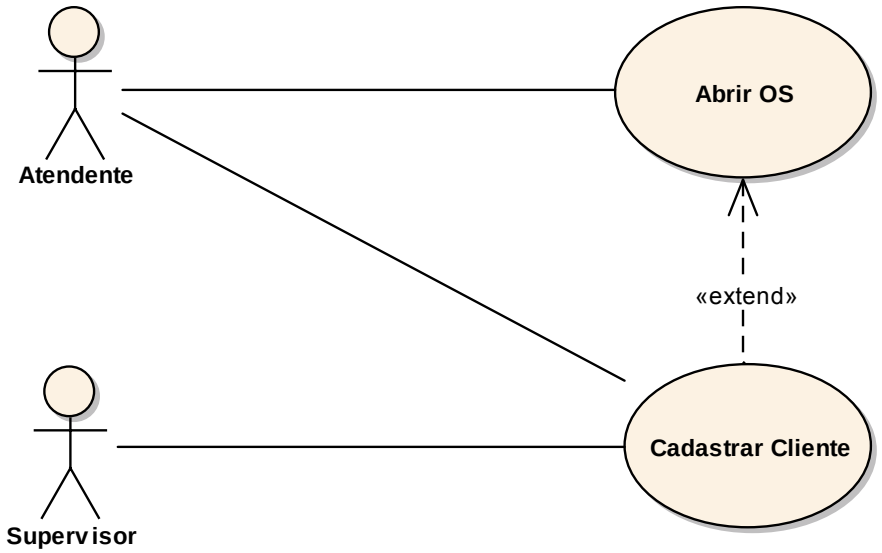


Figura A.5: Ator associado ao caso de uso base e ao que estende.

uma funcionalidade do sistema operacional implementada por um relógio interno que "sabe" a hora e o mecanismo de agendamento de ações a serem executadas nas horas agendadas. No Windows isso é feito pelo comando "AT"; no Linux, isso é o CRON/Crontab. O diagrama fica conforme a Figura A.7.

h) Essa situação é bem simples. Basta pensarmos em como precisa ser a sequência das ações do ator e do sistema nesse caso. Um usuário, ao se autenticar no sistema, faz com que o sistema verifique se ele está ou não na lista negra. Caso o usuário esteja, ainda durante a execução do caso de uso `Efetuar Logon` o sistema envia um "torpedo" ao chefe do suporte, que não participa do caso de uso como ator porque não interage com ele, mas com a funcionalidade de ler SMSs do seu celular. Esses detalhes estão dentro do caso de uso e não aparecem no diagrama, que fica conforme a Figura A.8.

A.3 Exercícios do Capítulo 4, página 57:

1. Para responder ao item, suponha um caso de uso `A` que inclui o caso de uso `B`. `A` faz a chamada a `B`, executando seus passos. Suponha agora que o caso de uso

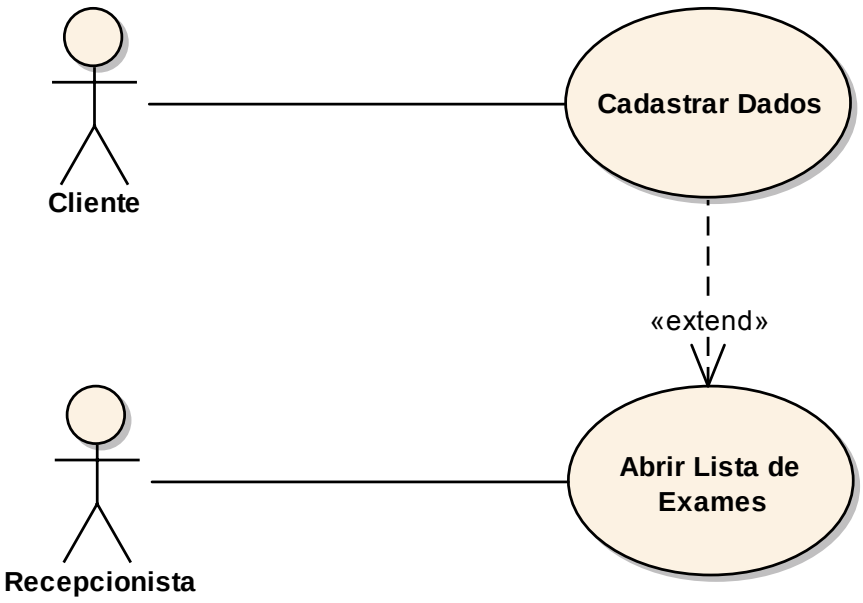


Figura A.6: Caso de uso executado por um ator ou possivelmente durante a execução de outro caso de uso por outro ator.

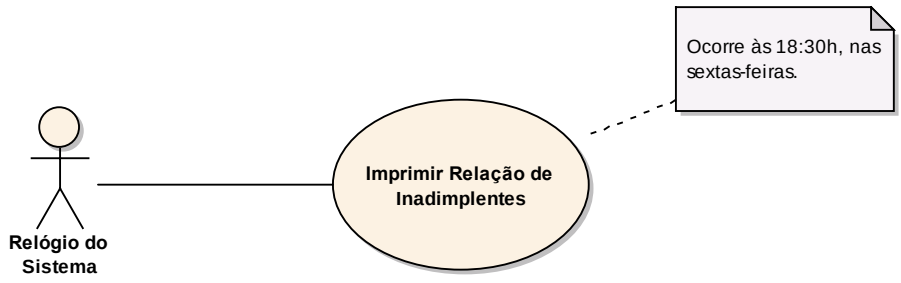


Figura A.7: Usando o mecanismo de agendamento do Sistema Operacional como ator.

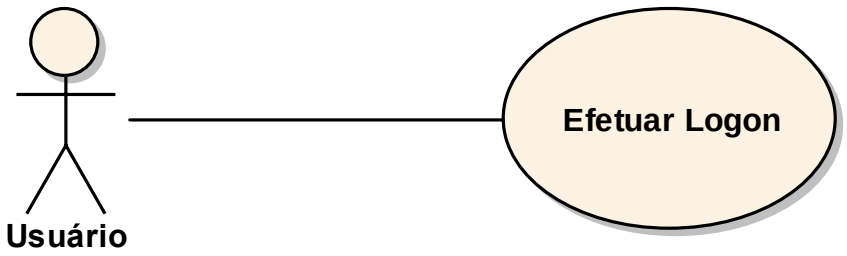


Figura A.8: Complexidade "escondida" em um caso de uso.

`B` estenda o caso de uso `A`. `B` estende (engorda, aumenta) `A` com seus passos; portanto, `A` inclui `B`, só que opcionalmente, segundo as regras de negócio.

2. O relacionamento de inclusão especifica que a chamada ao caso de uso é feita sempre segundo as regras de negócio. O sempre significa, neste caso, idealmente, tipicamente... Lembre-se do pagamento pelo almoço (Seção 3.5, Figura 3.7), que, embora obrigatório, pode não ocorrer se o cliente fugir pela janela. Já extensões são inclusões que não ocorrem sempre, portanto alternativamente, ou atipicamente, segundo as regras de negócio.

3. O diagrama fica como o da Figura A.9

   Note que mesmo que o `Supervisor`, o `Cliente` e a `Operadora do Cartão` de crédito interajam eventualmente com a funcionalidade `Registrar Compra`, o relacionamento entre eles e o caso de uso é de associação.

   A descrição abreviada pode ser algo como na Tabela A.1.

   Note também que identificamos o `Caixa` como iniciador do caso de uso porque é ele que, efetivamente, indica o início de uma nova compra no sistema. O `Cliente` seria o iniciador do caso de uso de negócio, se o estivéssemos descrevendo.

   E a descrição detalhada pode ser como nas tabelas A.2 e A.3:

   Repare que usei como cabeçalho da descrição detalhada o mesmo conteúdo da descrição abreviada. Além disso, coloquei sa palavras `Supervisor`, `Operadora`, SCE, SF e SCR entre parênteses na relação de atores, como se fossem apelidos para `Supervisor de Vendas`, `Operadora do Cartão`, `Sistema de Controle de Estoque`, `Sistema de Faturamento` e `Sistema de Contas a Receber`,

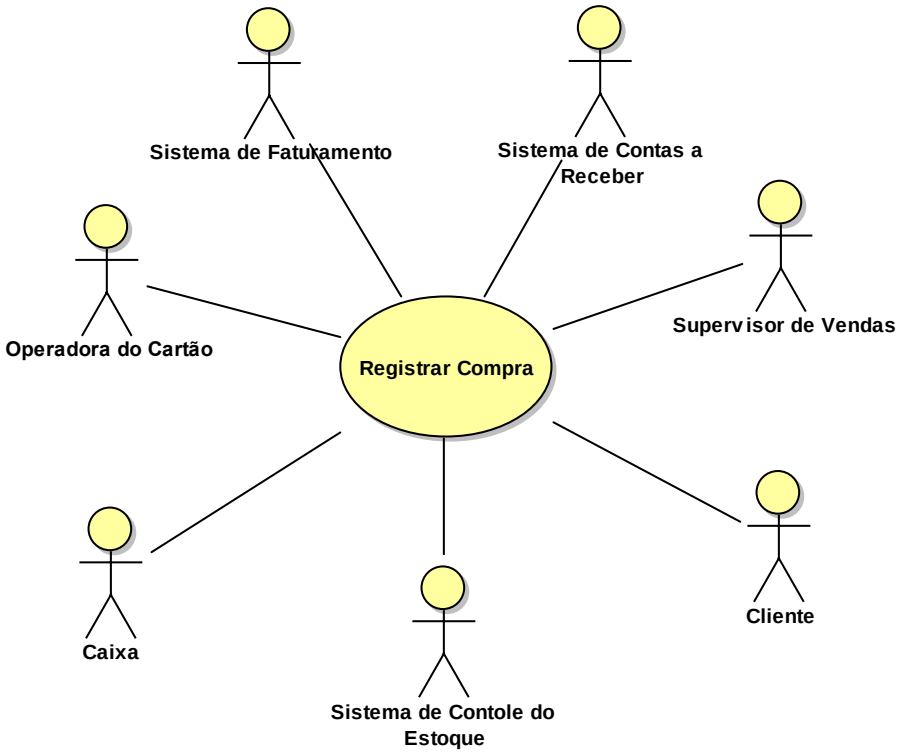


Figura A.9: O registro de compra em um supermercado.

respectivamente. Uso muito isso para dar "apelidos" mais abreviados aos atores, facilitando o trabalho de descrição do caso de uso.

Não houve muita preocupação com a precisão com respeito às técnicas usadas nesse tipo de aplicação. A completude da descrição também foi sacrificada a bem da concisão deste texto; nem todos os cursos alternativos e de exceção foram tratados. Situações como a não aprovação da operadora de cartão, seja por senha inválida do cliente ou por insuficiência de crédito, assim como falhas de comunicação com o SCE, com o SF, o SCR e com a operadora do cartão de crédito, além possibilidade de informação de senha incorreta de supervisor, deveriam ser tratados se o sistema fosse um sistema "de verdade".

Consideramos, no entanto, que o grau de detalhamento da solução apresentada é suficiente, do ponto de vista acadêmico, para os propósitos de nosso texto. Se você é um especialista nesse tipo de sistema, sinta-se à vontade para descrever

Tabela A.1: Descrição abreviada do caso de uso `Registrar Compra`.

Caso de Uso:	Registrar Compra
Propósito:	Registrar os itens de um carrinho de compras no supermercado e receber o pagamento pela compra.
Descrição geral:	Um cliente chega ao caixa com os itens que deseja comprar. O Caixa registra os itens de compra e recebe o pagamento, que pode ser feito em dinheiro, em débito ou crédito no cartão. Ao final, o Cliente deixa a loja com os produtos adquiridos.
Atores:	Caixa, Cliente, Supervisor de Vendas, Operadora do Cartão, Sistema de Controle de Estoque, Sistema de Faturamento, Sistema de Contas a Receber.
Iniciador:	Caixa.
Pré-condições:	Venda anterior encerrada, Vendedor autenticado no sistema.
Pós-condições:	Venda registrada, paga e concluída em caso de sucesso.

completamente o caso de uso e com maior precisão.

Podemos apreender deste exercício que a tarefa de especificação de casos de uso é bastante extenuante pela necessidade de ser completa e precisa, condicionada à qualidade de ser concisa.

As especificações, após lidas e aprovadas pelos usuários, seguem para o pessoal de projeto, para especificação da solução, e para e equipe de formulação dos casos de testes funcionais que irão ajudar na aprovação do sistema (homologação) para implantação.

Tabela A.2: Descrição detalhada do caso de uso `Registrar Compra` (início).

Caso de Uso:	Registrar Compra
Propósito:	Registrar os itens de um carrinho de compras no supermercado e receber o pagamento pela compra.
Descrição geral:	Um cliente chega ao caixa com os itens que deseja comprar. O Caixa registra os itens de compra e recebe o pagamento, que pode ser feito em dinheiro, em débito ou crédito no cartão. Ao final, o Cliente deixa a loja com os produtos adquiridos.
Atores:	Caixa, Cliente, Supervisor de Vendas (Supervisor), Operadora do Cartão (Operadora), Sistema de Controle de Estoque (SCE), Sistema de Faturamento (SF), Sistema de Contas a Receber (SCR).
Iniciador:	Caixa.
Pré-condições:	Venda anterior encerrada, Vendedor autenticado no sistema.
Pós-condições:	Venda registrada, paga e concluída em caso de sucesso.

Curso típico (CT)

1. Caixa indica no sistema o início de uma nova compra.
2. Sistema exibe no monitor o texto “Próximo Cliente”.
3. Sistema “zera” o total da compra.
4. Sistema imprime na fita o cabeçalho da lista com os dados do estabelecimento e a data/hora atuais.
5. Para cada item do carrinho de compras
   a. Caixa apresenta o código de barras ao leitor ótico.
   b. Sistema emite um *beep* sinalizando que o código foi lido e o produto foi localizado no estoque.
   c. Sistema busca o preço e a descrição do produto no cadastro.
   d. Sistema exibe o preço e a descrição do produto no monitor.
   e. Sistema adiciona o preço do produto ao total da compra.
   f. Sistema imprime item de compra na fita de papel.
   g. Sistema exibe o novo total da compra, sobrescrevendo o total exibido anteriormente.
6. Caixa informa ao sistema o fim da lista de compras.
7. Sistema solicita a forma de pagamento.
8. Caixa informa que o pagamento será feito por meio de cartão de crédito com *chip*.
9. Sistema instrui ao cliente a inserção do cartão.
10. Sistema aguarda a inserção do cartão.
11. Sistema lê o número do cartão.
12. Sistema exibe no visor do leitor do cartão o valor da compra e solicita a informação da senha.
13. Cliente digita a senha do cartão e pressiona tecla verde “Enviar”.
14. Sistema transmite o número do cartão, o total da compra e a senha à operadora.
15. Operadora valida a compra.
16. Sistema envia o total da compra ao SCR.
17. Sistema informa que a compra foi aprovada e solicita que cliente retire o cartão da leitora de cartões.

Tabela A.3: Descrição detalhada do caso de uso `Registrar Compra` (final).

18. Sistema abre a gaveta do caixa. 19. Sistema envia a lista de itens adquiridos ao SCE. 20. Sistema envia o total da compra ao SF. 21. Sistema exibe a mensagem “Compra encerrada com sucesso” no monitor. 22. Sistema conclui a transação de compra. ** Fim do Caso de Uso **
Cursos alternativos (CA)
CA 1: Passo 5g do CT – Cliente solicita ao Caixa que interrompa a transação de venda por indisponibilidade financeira.
** COMPLETAR, COMO EXERCÍCIO **
CA 2: Passo 8 do CT – Caixa informa que o pagamento é em dinheiro.
1. Caixa registra o valor em dinheiro dado pelo Cliente. 2. Sistema calcula o valor a ser devolvido a título de troco. 3. Volta ao passo 18 do CT.
CA 3: Passo 15 do CT – Operadora não aprova a compra.
** COMPLETAR, COMO EXERCÍCIO **
Cursos de exceção (CE)
CE 1: Passos 5b do CT – Sistema não pôde ler o código de barras.
1. Caixa digita o código manualmente. *2.* Volta ao passo 5c do CT.
CE 2: Passos 5f do CT – Final da fita de impressão.
1. Supervisor pressiona tecla de intervenção de supervisor. 2. Sistema solicita senha de supervisor. 3. Supervisor informa senha. 4. Sistema valida senha de supervisor. 5. Sistema exibe menu de supervisor. 6. Supervisor escolhe opção “Reimprimir Itens da Transação de Compra”. 7. Sistema imprime dados do estabelecimento, data e hora originais da compra e a lista de itens. 8. Supervisor se “*desloga*” do sistema, passando o controle da transação de volta ao Caixa. Sistema retorna ao passo 5g do CT.

A.4 Exercícios do Capítulo 5, página 76:

1. As entidades e as informações são:

   - `Universidade`: nome e endereço;
   - `Departamento`: nome;
   - `Professor`: nome;
   - `Disciplina`: nome, data de início e de fim, duração da disciplina;
   - `Aluno`: nome, endereço, data de nascimento etc.

   Cabe, adicionalmente, uma observação importante: quando você pensou nas informações para a identificação das entidades, muito provavelmente você incluiu alguns relacionamentos. Por exemplo, quando você identificou a classe `Professor`, pode ter pensado nas disciplinas que um professor leciona como uma informação que seria interessante armazenar para os professores. Na realidade, você corretamente identificou a associação de uma classe com outra como sendo uma informação que uma ou ambas as classes devem armazenar.

2. Aproveitando a ideia da resposta comentada da atividade 1 teremos:

   - Universidade: `nomeUniversidade`, `enderecoUniversidade`;
   - Departamento: `nomeDepartamento`;
   - Professor: `nomeProfessor`, `dataNascimentoProfessor`, `dataAdmissao`;
   - Disciplina: `nomeDisciplina`, `dataInicio`, `dataFim`, `duracaoDisciplina`;
   - Aluno: `nomeAluno`, `enderecoResidenciaAluno`, `dataNascimentoAluno` etc.

   Os rótulos completos para alguns desses atributos poderiam ser:
   ```
   +nomeUniversidade:String;
   +enderecoUniversidade:String;
   -dataInicio:Date;
   -dataFim:Date;
   -/duracaoDisciplina:int;
   ```

   As visibilidades foram definidas como públicas e privadas sem nenhum critério específico além de para exercitarmos a notação. O atributo `duracaoDisciplina` foi definido como derivado porque pode ser calculado pela diferença entre `dataFim` e `dataInicio` da disciplina.

   Cabe, aqui, um breve comentário: é usual a omissão dos elementos formadores de genitivos (do, da, dos, das) nos identificadores dos atributos para reduzir

seus tamanhos, sem perda significativa de expressividade. Assim, o identificador `nomeDaDisciplina` pode ser substituído, como foi, por `nomeDisciplina`.

3. Os nomes das associações estão em itálicos:

   a) `PedidoDeReposicaoDeEstoque` *contém* `ItemDeReposicaoDeEstoque`;

   b) `Cliente` *coloca* (ou *faz*) `Pedido`;

   c) `Fornecedor` *fornece* `Produto`;

   d) `Produto` *especifica* **ItemDePedido**;

   e) `Funcionario` *atende a* `ClientePJ`.

   Esses nomes são bons porque, se quisermos ler no sentido contrário, a leitura fica da seguinte forma:

   a) `ItemDeReposicaoDeEstoque` *é contido em* `PedidoDeReposicaoDeEstoque`;

   b) `Pedido` *é colocado por* (ou *é feito por*) `Cliente`;

   c) `Produto` *é fornecido po*r `Fornecedor`;

   d) `ItemDePedido` *é especificado por* `Produto`;

   e) `ClientePJ` *é atendido por* `Funcionario`.

   Para concluir a resposta, é importante salientar que quase todos os nomes dessas associações podem ser omitidos do diagrama por serem associações com significados um tanto óbvios. Talvez eu deixasse apenas o nome da associação do item 5, se eu não tivesse colocado o rótulo do papel que `Funcionário` assume nessa relação com `ClientePJ`. O funcionário é o representante de vendas para o `ClientePJ`.

4. - Um pedido de reposição de estoque está associado a um ou mais itens de reposição de estoque e um item de reposição de estoque está associado a um pedido de reposição de estoque;
   - Um item de reposição de estoque está associado a um produto e um produto está associado a zero ou mais (também se fala "a qualquer número de") itens de reposição de estoque;
   - Um pedido está associado a um ou mais itens de pedido e um item de pedido está associado a um pedido;
   - Um item de pedido está associado a um produto e um produto está associado a zero ou mais itens de pedido;
   - Um pedido está associado a um cliente e um cliente está associado a qualquer número de pedidos;

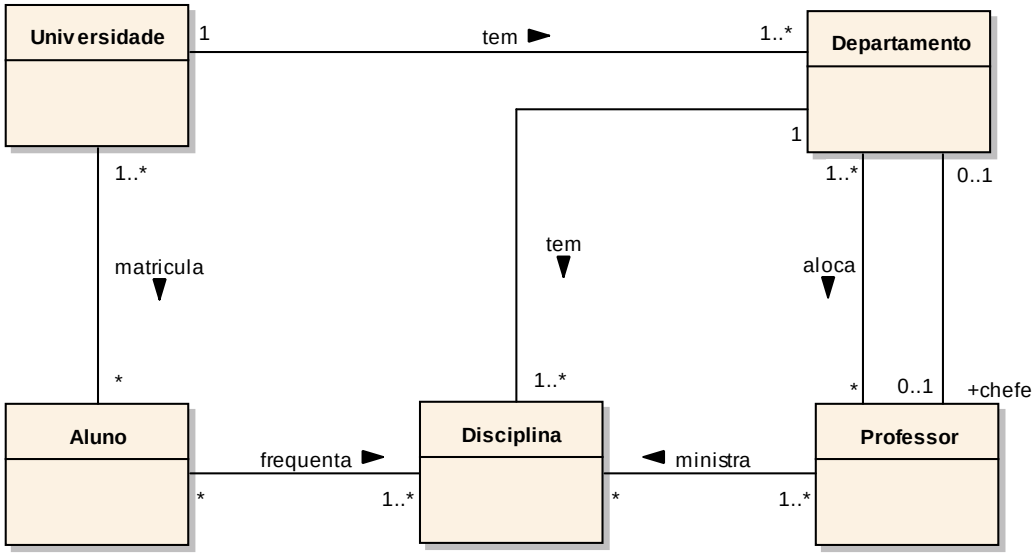


Figura A.10: Ambiente Acadêmico do Município de Sertãozinho Alegre.

- Um cliente PJ está associado a zero ou 1 representante de vendas e um representante de vendas está associado a qualquer número de clientes PJ.

Repare que não li *funcionário* no último item, mas sim o papel de representante de vendas que funcionário assume na associação com cliente PJ. No último item eu também poderia ler: cliente PJ *tem ou não* um representante de vendas, caracterizando a multiplicidade opcional `0..1`.

5. O modelo que desenvolvemos é o da Figura A.10.

   Um departamento encontra-se associado a somente uma universidade, porque o objeto (instância de `Departamento`) "Departamento de Física da PUC", por exemplo, está associado a somente um objeto da classe `Universidade`, no caso a PUC.

   Já discutimos a questão dos nomes das associações entre `Departamento` e `Professor`. As multiplicidades são opcionais em cada ponta da associação de chefia, porque um Departamento pode estar sem chefe e um professor pode não ser chefe mas, se for, ele é de no máximo um departamento (não há acúmulo de chefia, certo?).

   Já vimos que a expressão "qualquer número de" resulta em multiplicidade "zero ou mais", representada na UML por um $*$ ou `0..*`, indistintamente.

A questão de um aluno poder frequentar uma ou mais disciplinas muitas das vezes é interpretada como a não obrigatoriedade de ele estar frequentando pelo menos uma disciplina. Na realidade, os verbos poder e dever causam muita confusão em uma especificação. Embora os significados desses dois verbos estejam precisamente definidos na NM1 (Norma Mercosul 1), costumo aconselhar os meus alunos a não usá-los e, caso os encontrem em uma especificação em prosa, perguntem aos especialistas do negócio antes de elaborar os modelos UML. Imaginemos que eu tenta perguntado ao especialista do negócio "Ambiente Acadêmico" e ele me respondeu que um aluno precisa estar matriculado em pelo menos uma disciplina. Por isso usei a multiplicidade "1..*" na ponta direita da associação `frequenta`.

Algumas multiplicidades não foram especificadas no texto. Perguntei aos especialistas e usei o bom senso quando eles não souberam me responder.

Os atributos poderiam constar do modelo... eu só não me preocupei com eles neste momento.

A.5 Exercícios do Capítulo 6, página 98:

1. O modelo que desenvolvemos é o da Figura A.11.

   Tornamos as classes `Usuario` e `UsuarioComum` abstratas pelo fato de não existirem usuários desses tipos, ou seja, de só existirem usuários dos tipos `UsuarioComumAluno`, `UsuarioComumMembroComunidade` e `UsuarioFuncionario`. Além disso, poderíamos marcar no modelo essas especializações como coberturas completas.

   Criamos o atributo multivalorado `autor` na classe `Obra` para podermos armazenar qualquer número de autores, inclusive zero, para cada obra. A solução de termos nenhum autor para uma obra modela corretamente a possibilidade de cadastrarmos obras anônimas.

   Poderíamos fazer o mesmo com o atributo `assunto`, mas decidi criar uma classe `Assunto` associada à classe `Obra` como uma alternativa à multivaloração do atributo.

   Não associamos a classe `Exemplar` à classe `UsuarioComum` para fatorarmos as duas associações de empréstimo, porque elas têm multiplicidades diferentes nas pontas `UsuarioComumAluno` e `UsuarioComumMembroComunidade` (`0..1` e `0..2`, respectivamente).

   Repare que, na solução apresentada, não coloquei como atributo da classe `Exemplar` a marcação de que o exemplar se encontra emprestado. Marcações

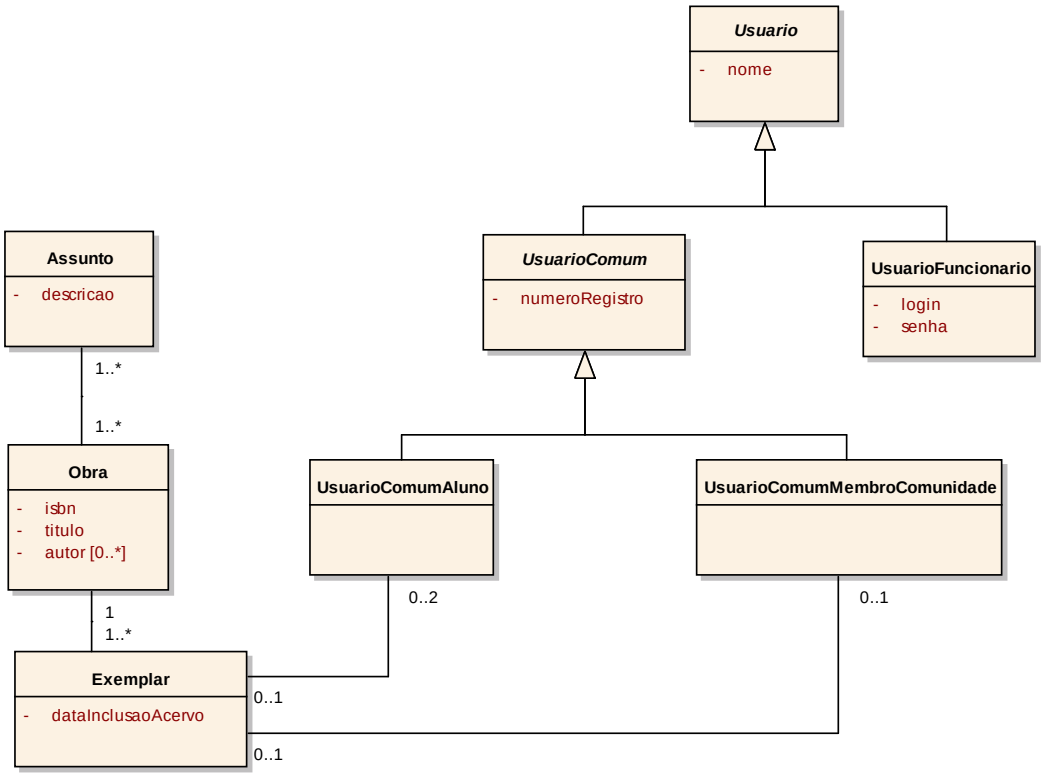


Figura A.11: Modelo da solução para o sistema de controle de uma biblioteca.

são as tais *flags* que o pessoal de implementação "adora" colocar como atributos de classes. Deixei de incluir essa *flag* porque ela não é, de fato, necessária. De maneira geral, *flags* não são necessárias porque são atributos derivados, ou seja, são obtidas por meio de algoritmos que executamos usando valores de outros atributos. Se uma informação é resultado de um algoritmo, essa informação não precisa ser armazenada, pois ela sempre pode ser calculada quando for necessária. Adicionar atributos derivados a classes pode, ainda, ser uma fonte de inconsistências. Atributos derivados são adicionados a classes apenas quando o algoritmo para obter seus valores é muito complexo ou demorado; essas considerações devem ser feitas somente em tempo de projeto. No nosso caso, a informação de que um exemplar não está disponível pode ser obtida por meio do exame da existência de instâncias das associações de empréstimo; estar disponível significa não estar emprestado a alguém, certo? Visto de outra

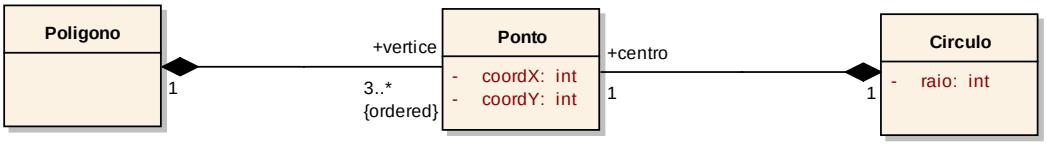


Figura A.12: Modelo da solução para o editor gráfico.

forma, um exemplar estar emprestado significa haver uma (e não mais do que uma) instância de associação do tal exemplar (que é uma instância da classe `Exemplar`) com alguma instância da classe `UsuarioComumAluno` ***ou*** (exclusive) `UsuarioComumMembroComunidade`.

2. A solução que demos é a da Figura A.12:

   Criamos uma classe `Ponto` para representar o par ordenado $P(x, y)$ do plano. As composições especificam que, se um ponto está associado a um polígono, representando o papel de vértice, ele não pode estar associado, ao mesmo tempo, a um círculo, representando o papel de centro. As multiplicidades 1 nas pontas das composições especificam que, se um polígono for removido, seus vértices serão removidos também, já que não podem estar associados a 0 (zero) polígonos. O mesmo raciocínio se aplica aos círculos e seus centros.

   Repare que colocamos junto à multiplicidade `3..*` o rótulo `{ordered}` para especificar que a coleção de vértices é ordenada, ou seja, segue uma sequência determinada, que não importa agora qual é. Expressões entre "" e "" colocadas no modelo são outras restrições que devem ser observadas durante a execução do sistema. Veremos mais sobre restrições adiante, no final deste capítulo.

3. A resposta é direta: basta aplicar a transformação da Figura 6.14; considerando que a classe `A` da Figura 6.14 corresponde à classe `Empresa` da Figura 6.13, a classe `B` corresponde à classe `Pessoa`, a classe `C` corresponde à classe `Emprego`, m corresponde à multiplicidade $*$ e n corresponde à multiplicidade `0..1`. O modelo fica, portanto, como na Figura A.13.

4. A solução é representarmos os dois pacotes com relacionamentos de dependência entre eles. Note que são duas setas, e não uma seta com duas pontas. A Figura A.14 ilustra.

   Ao detalharmos mais cada pacote, as classes de interface que proveriam esse isolamento estariam representadas dentro do pacote com o relacionamento de dependência chegando nelas.

Figura A.13: Modelo da solução para transformação de classe de associação para classe cheia.

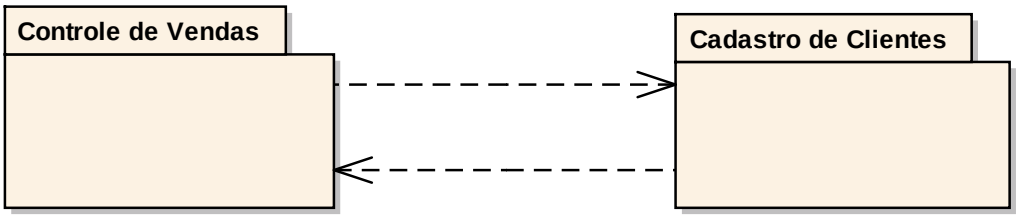


Figura A.14: Modelo da solução para dependência entre pacotes.

5. A restrição de que um exemplar não pode ser emprestado a dois alunos ou a dois membros da comunidade está garantida pelas multiplicidades `0..1` nas pontas das associações das classes `UsuarioComumAluno` e `UsuarioComumMembroComunidade` com a classe `Exemplar`, junto a esta. Entretanto, nada impede, segundo aquele modelo, que um exemplar seja emprestado a um aluno e a um membro da comunidade simultaneamente, o que não pode acontecer. A solução é tornarmos essas duas associações mutuamente excludentes, aplicando a elas a restrição `XOR`. A Figura A.15 ilustra.

A.6 Exercícios do Capítulo 7, página 119:

1. Eu não perderia meu tempo desenvolvendo um DME para os objetos da classe `Interruptor`, porque qualquer interruptor só possui dois estados: ligado e desligado, passando de um a outro unicamente pelos eventos de ligamento e desligamento.

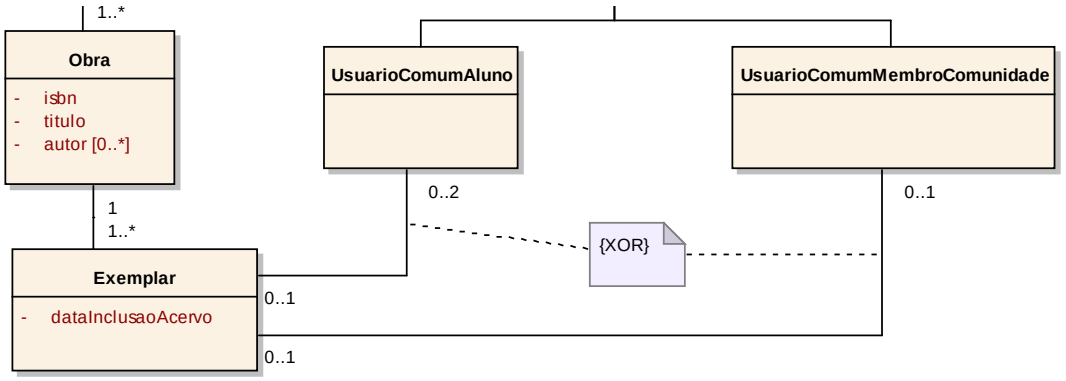


Figura A.15: Modelo da solução para tornarmos mutuamente excludentes instâncias de associações.

Já os objetos da classe `Ar Condicionado` possuem mais estados: `desligado`, `ligado no ventilador`, `ligado na posição frio`, `ligado na posição quente` (isso para alguns deles), tudo isso usando exaustão ou recirculação e com vários níveis de frio ou calor. Dependendo dos tipos de controles do painel, os eventos que o operador precisa gerar para passar de um estado a outro podem não ser tão triviais quanto o *switch* de um interruptor.

2. A solução que demos está no DME da Figura A.16.

   Podemos assumir que, quando um apartamento é instanciado, ou seja, quando é cadastrado no sistema, ele assumirá automaticamente a condição `Livre`. Por isso colocamos o pseudoestado inicial apontando para esse estado.

   Estando `Livre`, um apartamento poderá transitar para `Ocupado`, pela ocorrência de um *check-in*, ou para `Reservado`, pela ocorrência de uma reserva feita por um cliente para o dia corrente.

   Estando `Ocupado`, um apartamento só poderá transitar para `Livre`, pela ocorrência de uma desocupação (*check-out*, no jargão da hotelaria). É nesse momento que a fatura deverá ser impressa, ou seja, imediatamente antes da saída do estado `Ocupado`. Por isso especificamos a impressão da fatura usando o rótulo `exit/imprimir fatura`. Note que poderíamos ter colocado a impressão da fatura como ação na transição de `Ocupado` para `Livre`, certo?

   Estando `Reservado`, um apartamento poderá transitar para `Livre`, caso ocorra o cancelamento da reserva, ou para `Ocupado`, caso o cliente que o tenha reservado efetue o *check-in*.

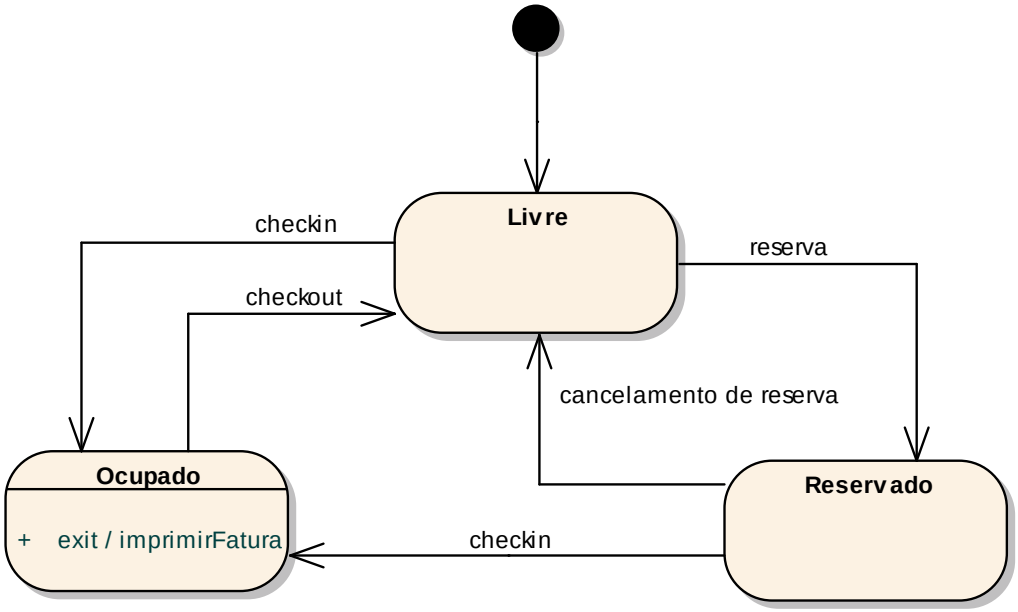


Figura A.16: Diagrama de máquina de estados para objetos da classe `Apartamento` do Hotel Cincoestrelas.

3. Existe algo em comum entre os estados `Reservado`, `Ocupado` e `Livre`, que se opõem ao estado `Interditado para Obras`. Esse algo em comum refere-se à situação, nos três primeiros casos, de um apartamento estar *operacional*. Interditado para obras e, portanto, não operacional significa que o apartamento não está livre e que não tem possibilidade de ser ocupado ou reservado. Os estados `Reservado`, `Ocupado` e `Livre` podem ser considerados, dessa forma, subestados independentes do estado `Operacional`. O diagrama da Figura A.17 ilustra esses conceitos usados na solução do exercício.

   De acordo com o diagrama da Figura A.17, o estado `Operacional` é aquele que um apartamento assumirá quando for instanciado; o subestado `Livre` é o estado em que o apartamento entrará quando entrar no estado `Operacional`, seja pela instanciação, seja pelo retorno do estado `Não Operacional`.

   Estando em qualquer subestado de `Operacional`, um apartamento poderá transitar para `Não Operacional` pela ocorrência de uma solicitação de reparo. Por essa razão, a atividade `imprimirFatura` na saída do estado `Ocupado`, na solução dada para o Exercício 2, foi transformada em uma ação da transição entre o estado `Ocupado` e o estado `Livre`, porque não teria sentido o cliente

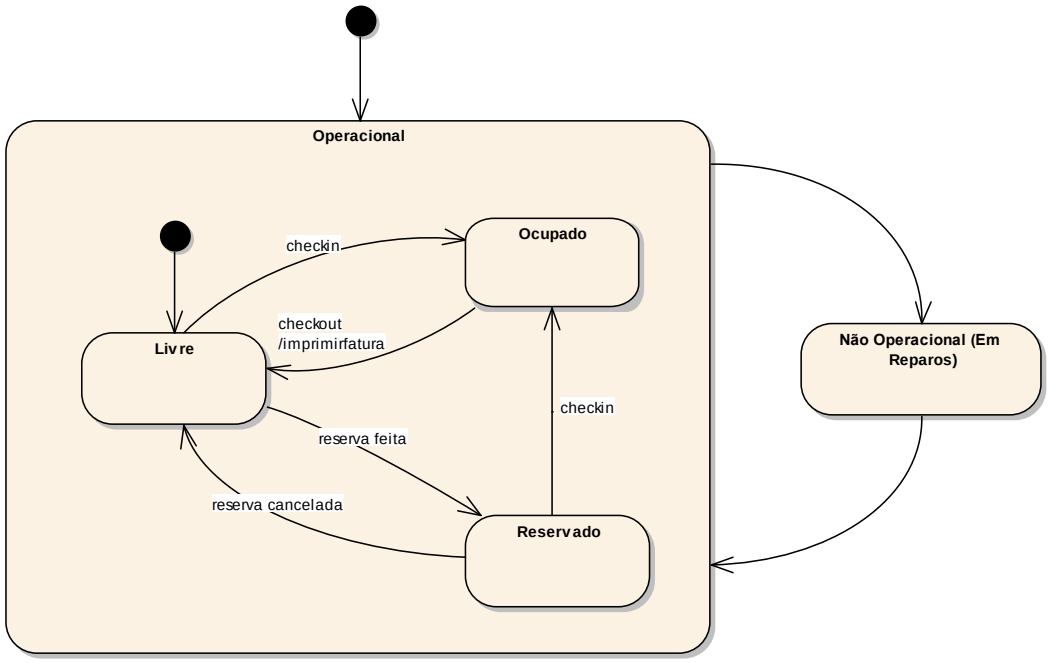


Figura A.17: Estados `Reservado`, `Ocupado` e `Livre` como subestados do estado `Operacional` na solução para o exercício do Hotel Cincoestrelas.

receber uma fatura tendo sido obrigado a sair do apartamento em consequência de sua interdição. Por isso, chamamos sua atenção para o cuidado que você precisa ter na escolha do lugar para colocação das ações e atividades ao usar estados compostos nos diagramas.

A.7 Exercícios do Capítulo 8, página 144:

1. A rotina matinal varia muito de pessoa para pessoa e, para a mesma pessoa, pode variar a cada dia. De maneira geral, uma rotina matinal pode ser modelada conforme o diagrama de atividade do diagrama da Figura A.18.

   Não vejo o que discutir no modelo além das ações que correspondem à preparação da refeição matinal. A preparação do sanduíche é uma ação que ocorre em paralelo com a preparação do café (que envolve ferver a água, derramá-la dobre o pó no coador e aguardar o café passar pelo coador, se

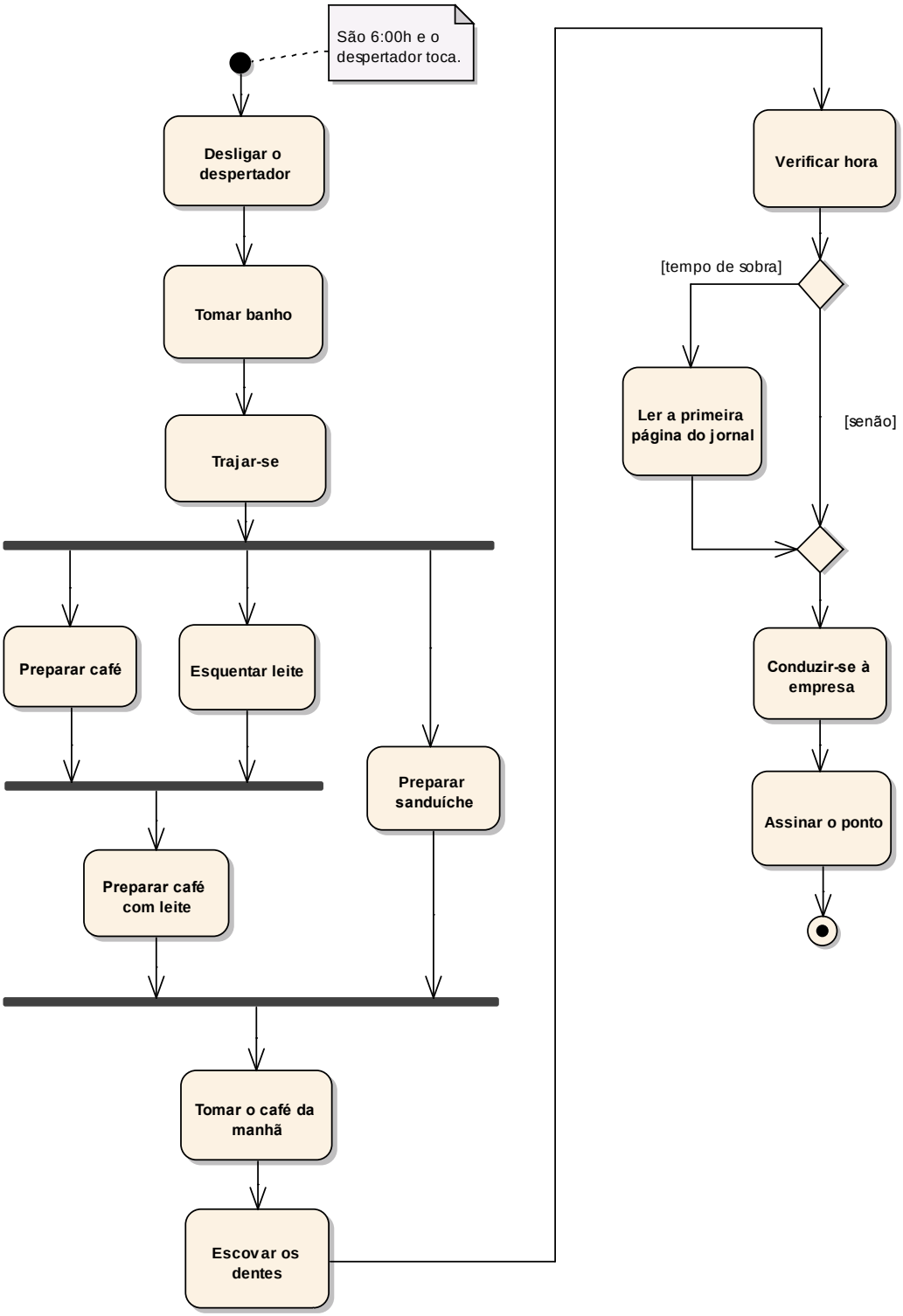


Figura A.18: Um exemplo de ações executadas em uma manhã típica.

você não usa uma máquina para isso). Para a preparação do café com leite, é necessário que o café esteja coado e que o leite esteja quente, por isso coloquei uma junção após essas duas ações para sincronizá-las antes da ação da preparação do café com leite.

2. O diagrama da Figura A.19 captura bem a iteração que é típica para quem usa a função "soneca" dos despertadores.

   Ao pressionar o botão "soneca", terminamos a atividade, preparando-nos (dormindo) para o próximo alarme. Da ação `Tomar banho` em diante, tudo se passa da mesma forma que na solução do Exercício 1.

3. O diagrama das Figuras A.20 e A.21 modela, apenas, o curso típico do caso de uso `Registrar Compra`.

   Pela complexidade, você pode perceber que a modelagem completa resultaria em um diagrama bem mais complexo, caso modelássemos completamente o caso de uso. Você pode elaborá-lo a título de exercício. Os nós de decisão em azul são pontos que originam cursos alternativos.

   As ações que foram apontadas como possíveis geradoras de cursos de exceção ou alternativos devem ser tratadas na solução final, sob pena de o sistema resultante não cobrir todos as possibilidades, como, por exemplo, o que deveria ser feito caso qualquer um dos sistemas (SCE, SF, SCR) estivesse fora do ar ou mesmo se a comunicação com a operadora do cartão não pudesse ser estabelecida.

A.8 Exercícios do Capítulo 9, página 163:

1. Acompanhando os desvios no diagrama, desde o início ao fim, é possível identificar os cenários assim:

   - *Cenário 1 (curso típico)*: Quantidade do único item do pedido está disponível em estoque na quantidade desejada;
   - *Cenário 2*: Quantidade do único item do pedido não está disponível em estoque na quantidade desejada;
   - *Cenário 3*: O pedido é composto por mais de um item e todos os itens estão disponíveis em estoque nas quantidades desejadas;
   - *Cenário 4*: O pedido é composto por mais de um item e nem todos os itens estão disponíveis em estoque nas quantidades desejadas.

   Repare que, com os quatro cenários identificados acima, todos os fluxos são percorridos, alguns deles mais de uma vez.

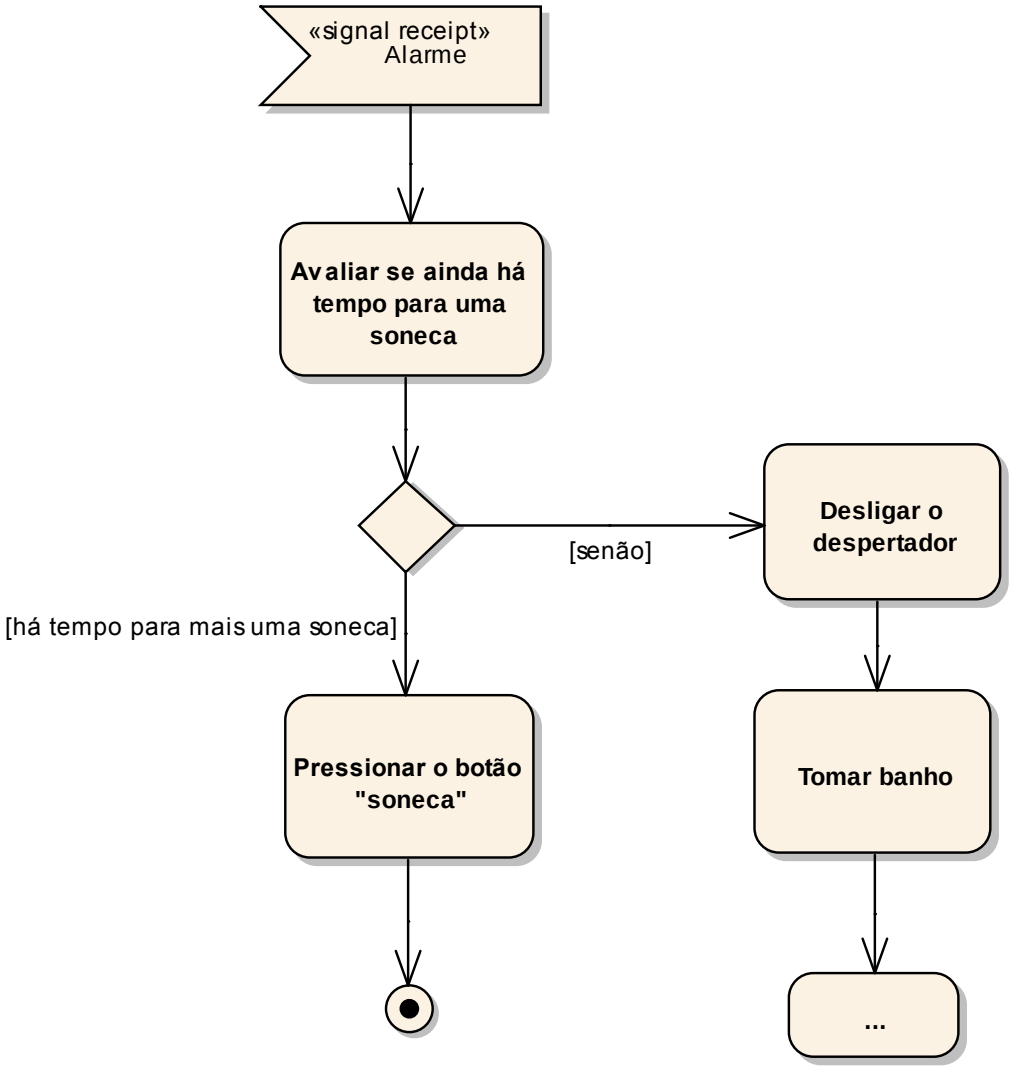


Figura A.19: Exemplo de ações para o tratamento de eventos temporais.

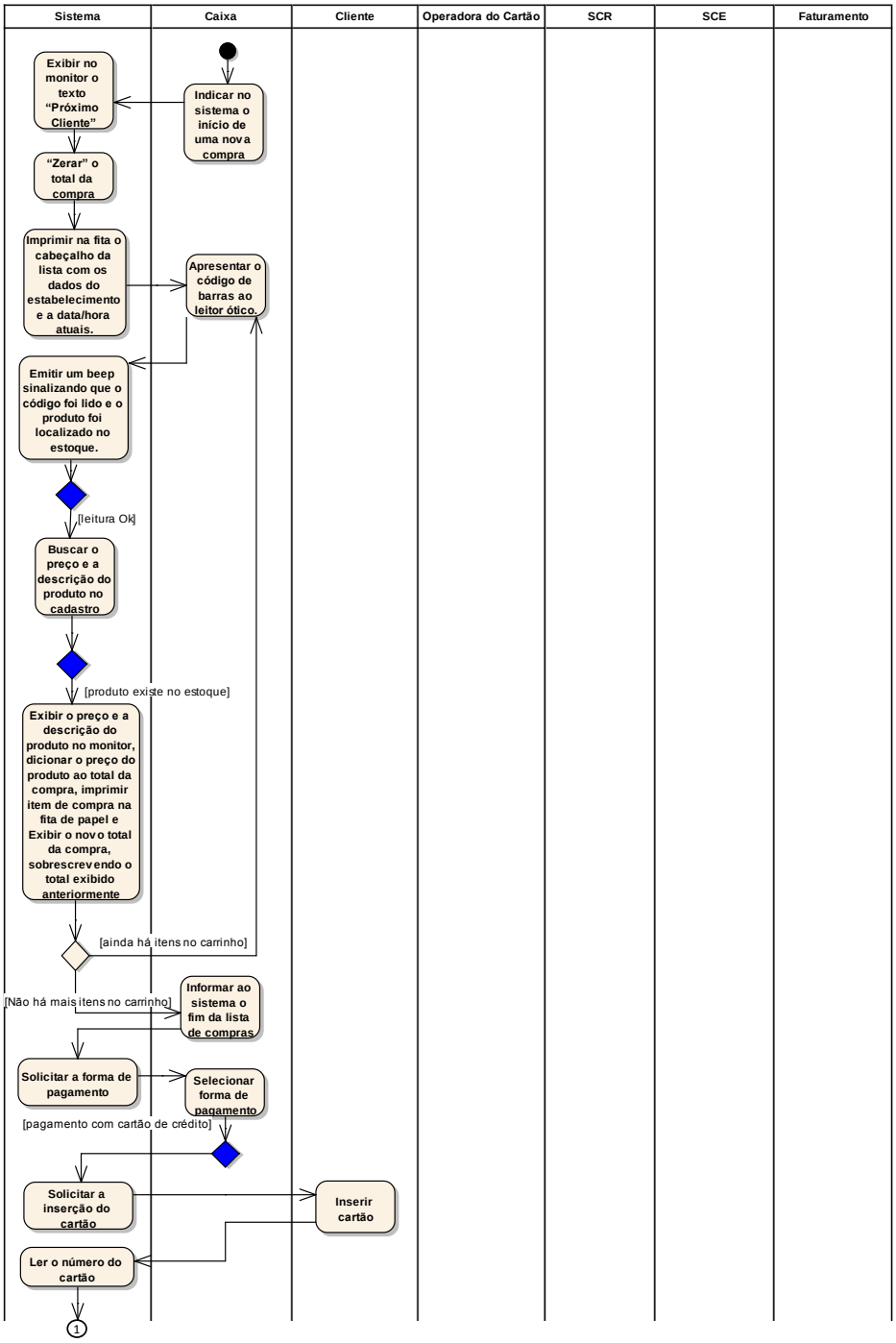


Figura A.20: Solução para a especificação na forma gráfica do caso de uso `Registrar Compra` – Parte I.

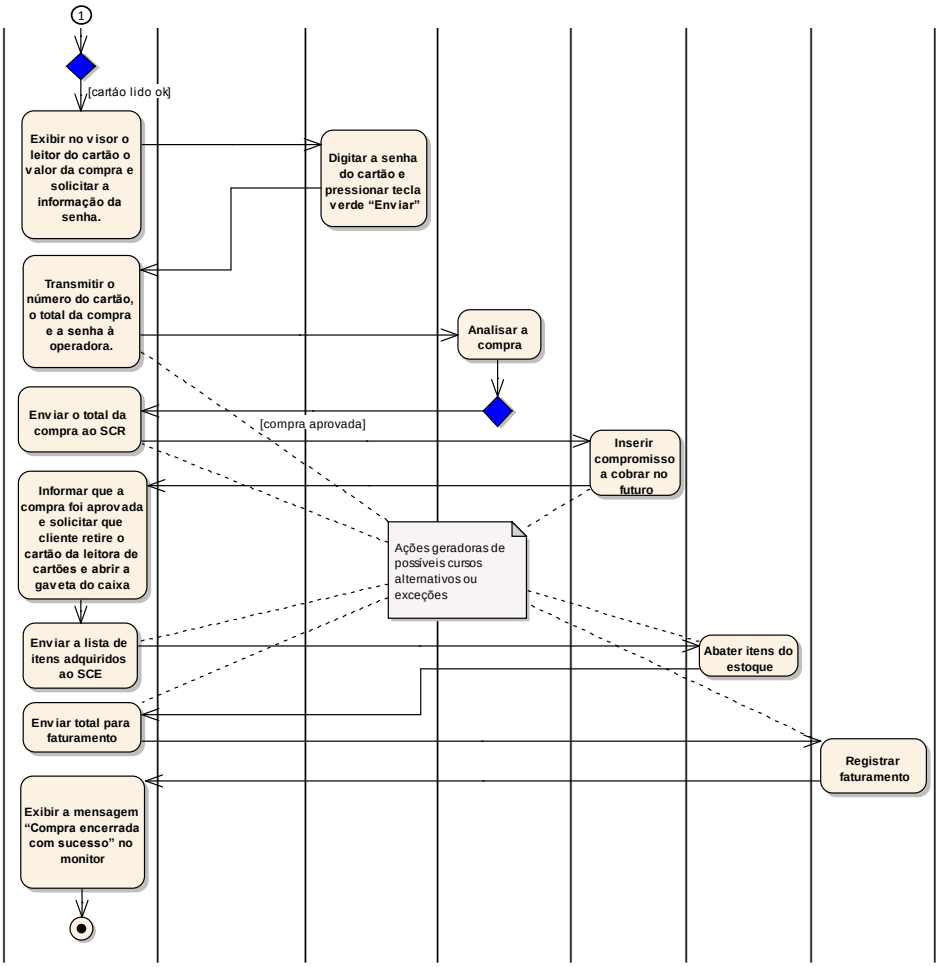


Figura A.21: Solução para a especificação na forma gráfica do caso de uso `Registrar Compra` – Parte II.

2. Os passos são os seguintes:

   a) Como temos o número do pedido, devemos pesquisar o conjunto de pedidos para localizar o pedido que corresponde ao número que temos. Há duas opções para isso:

      i. pesquisar os pedidos feitos pelos clientes até achar o que queremos. Para isso, iniciamos pela classe `ColecaoDeClientes`, obtendo os clientes da lista. Para cada cliente, obtemos os pedidos feitos por ele/ela até localizar o pedido. Nesse caso atribuiríamos mais essa responsabilidade à classe `ColecaoDeClientes`: localizar um pedido dado seu número em toda a coleção de clientes. Como consequência, a classe `ColecaoDeClientes` repassaria à classe `Cliente` as responsabilidades de indexar os pedidos de um cliente e de localizar um pedido na coleção de pedidos do cliente.

      ii. criar uma classe `ColecaoDePedidos` associada à classe `Pedido` (com multiplicidade 1 para muitos), que indexa os pedidos feitos à ZYX, e fazer uma busca nessa coleção, como fizemos com os clientes quando queríamos localizar um cliente pelo seu código. Nesse caso, a classe `ColecaoDePedidos` teria as responsabilidades de indexar pedidos e de prover mecanismos de pesquisa na coleção de pedidos.

   b) Com o pedido localizado, buscamos os dados do cliente que o colocou para compor o cabeçalho do pedido. A navegabilidade de `Pedido` para `Cliente` (seta na ponta da associação) indica essa responsabilidade da classe `Pedido`, que deve contar com operações e atributos para realizá-la. `Pedido` também precisa `imprimirCabecalhoPedido`, com os dados do cliente.

   c) Buscamos agora os dados dos itens que compõem o pedido. Para isso, a classe `Pedido` deve ter também a responsabilidade de indexar, localizar e obter os itens que compõem um pedido. O pedido localizado deve, para cada item de pedido, invocar a operação de obtenção dos dados do item, que vêm completos, incluindo a descrição e preço unitário – que não fazem parte dos atributos do item e sim do produto correspondente. Essa operação poderia ter a seguinte assinatura: `obterDadosItemPedido (ordem, quantidade, descricao, precoUnitario)`, sendo todos parâmetros de saída. Com esses dados de todos os itens, a classe `Pedido` precisa classificar os itens pela ordem, imprimi-los e calcular o total do pedido.

   d) Para executar a operação `obterDadosItemPedido`, o item de pedido precisa obter da classe `Produto` a descrição e o preço unitário do item. Solicitar esses dados é uma responsabilidade da classe `ItemDePedido`, que

Tabela A.4: Responsabilidades de cada classe na realização da colaboração do Exercício 2.

Classe	**Responsabilidade**
`ColecaoDeClientes`	Localizar um pedido por seu número
`Cliente`	Indexar os pedidos
	Localizar um pedido por seu número
	Prover os dados do cliente
`Pedido`	Obter os dados do cliente
	Imprimir o cabeçalho do pedido
	Obter os dados dos itens de pedido
	Imprimir os dados dos itens de pedido e totalizá-los
`ItemDePedido`	Prover os dados (completos) do item
	Solicitar descrição e preço unitário a `Produto`
`Produto`	Prover descrição e preço unitário do produto

a delega à classe `Produto`. Informar esses dados é uma responsabilidade da classe `Produto`.

Resumimos, na Tabela A.4, as responsabilidades de cada classe na realização dessa colaboração. Consideramos a alternativa *a* do passo 1.

Cabem, ainda, alguns comentários importantes.

A Tabela A.4 não faz parte de qualquer metodologia para elaboração de DSs. Ela serve apenas para consolidar o que discutimos neste exercício.

A resposta que demos acima é uma das possíveis respostas. Raciocinar e especificar correta e completamente uma colaboração na forma textual não é uma tarefa simples nem desejada num projeto. Felizmente, os diagramas de sequência ajudam na tarefa de atribuir responsabilidades e de descobrir operações, o que você verá na Seção 9.5. Deixaremos a elaboração de um diagrama tão abrangente para um exercício do próximo capítulo, quando tivermos apresentado mais elementos da notação gráfica da UML.

É importante observar que as responsabilidades que relacionamos são apenas parte da solução completa. Só teremos a solução completa quando considerarmos todos os cenários de todos os casos de uso do sistema.

3. Como solução do exercício elaboramos o diagrama da Figura A.22.

   Conforme vimos, a ordem dos objetos na dimensão horizontal não é relevante, ou seja, posso colocar Maria à esquerda de Paulo, conforme o diagrama da Figura A.23 sem alteração da solução. Repare que mantivemos a ordem e as

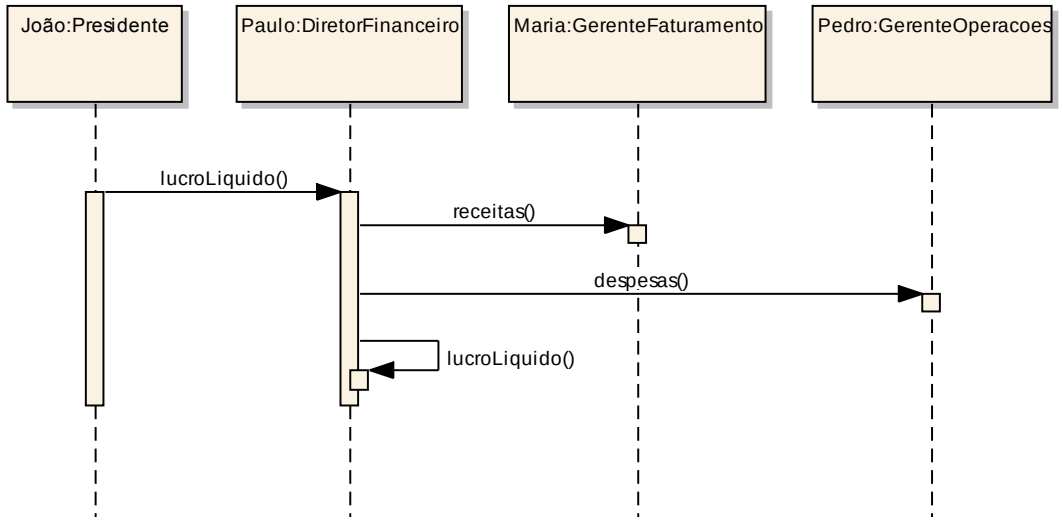


Figura A.22: Uma solução para o cálculo do Lucro Líquido da ZYX.

origens e destinos das chamadas na dimensão vertical, porque a colaboração foi especificada no enunciado dessa forma.

A.9 Exercícios do Capítulo 10, página 181:

1. A solução dada consta da Figura A.24. Supor que não existe quantidade disponível em estoque significa que o prazo de fornecimento não é imediato, ou seja, é necessária a obtenção do valor do atributo `prazoFornecimento` da classe `Fornecimento`.

   Iniciando, portanto, no item de pedido para o qual se deseja o prazo de fornecimento, solicita-se o prazo de fornecimento ao objeto `produto` associado (só existe um produto associado a um item de pedido).

   Como o produto "não sabe" qual é esse prazo (isso é atributo da classe `Fornecimento`), o objeto `produto` repassa a consulta ao primeiro fornecedor da lista de fornecedores, que são em qualquer número.

   O fornecedor, então, repassa a consulta ao objeto da classe `Fornecimento` associado. Nessa situação, no entanto, como fornecedores fornecem um número possivelmente grande de produtos, uma pesquisa por código de produto precisa ser feita para localização do produto para o qual precisamos do prazo de

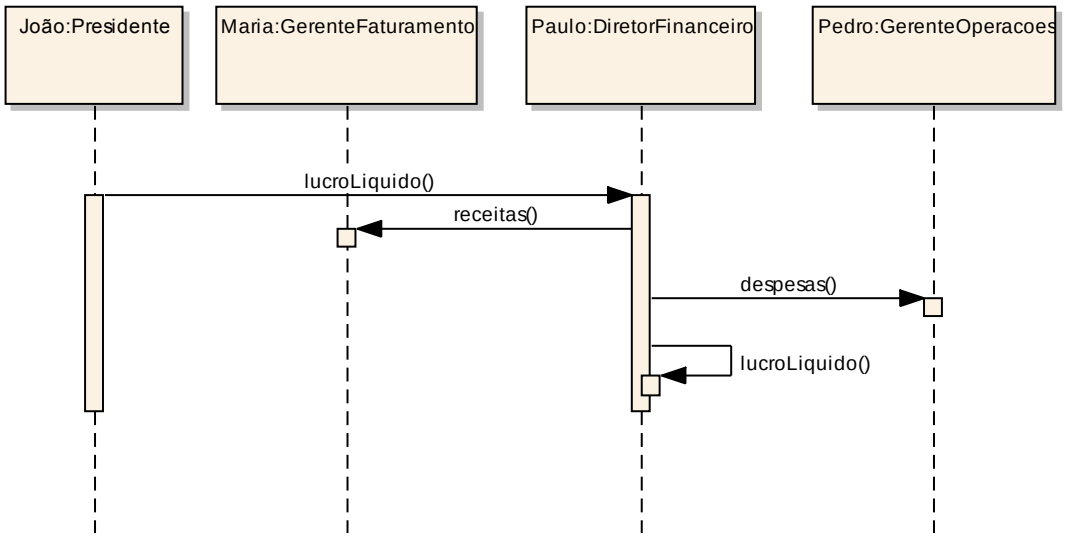


Figura A.23: A mesma solução que a da Figura A.22 para o cálculo do Lucro Líquido da ZYX, porém com outra ordem de apresentação dos objetos na dimensão horizontal.

entrega. Para isso, foi feita a autodelegação da pesquisa do produto no objeto `umFornecedor`, que retorna uma referência ao produto localizado (objeto `oFornecimento`).

Por meio dessa referência, obtemos o prazo de entrega, invocando a operação `getPrazoFornecimento` desse objeto. O retorno da informação é feito nos sentidos opostos às chamadas.

As operações descobertas e seus parâmetros constam do DS. Precisaríamos, agora, atualizar o diagrama de classes, colocando essa informação no terceiro compartimento das classes que instanciaram os objetos usados no diagrama.

Quanto às visibilidades, todas as chamadas são públicas, com exceção da autodelegação `localizarProduto`, que é privada, porque não precisa ser pública.

2. O DS de nossa resposta para o Exercício 1 evolui para o diagrama da Figura A.25. Os segmentos do quadro `alt` especificam as situações alternativas de haver e de não haver a quantidade necessária do item em estoque. O quadro `loop` verifica o preço mínimo, considerando todos os fornecedores do mesmo produto.

3. Elaboramos o diagrama da Figura A.26 como solução do exercício.

   No diagrama de sequência da Figura A.26 destacamos os seguintes aspectos:

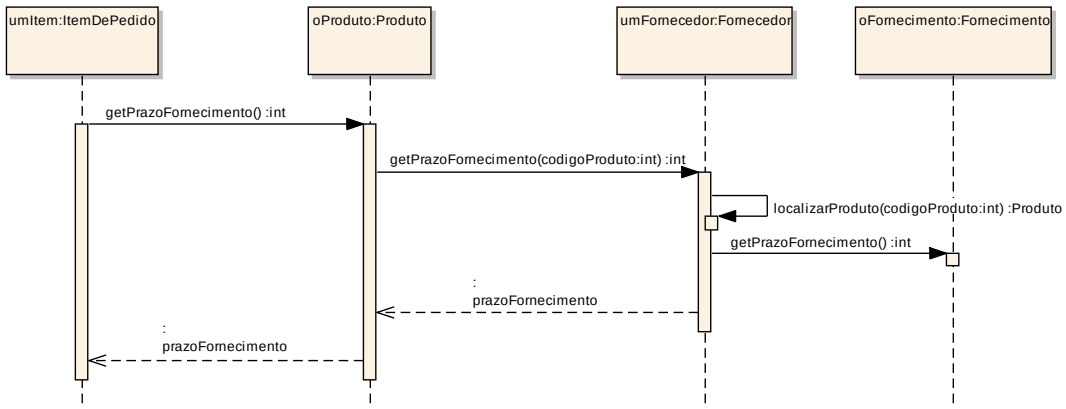


Figura A.24: Uma solução de colaboração para obtenção do prazo de entrega de um pedido feito à ZYX.

- todo o diagrama foi colocado em um quadro com rótulo `sd`, o que permite que o diagrama seja referenciado (chamado) em outro diagrama qualquer por meio de um operador `ref`;
- os atores são também aqui representados por "bonequinhos";
- criamos uma classe de interface através da qual o ator interage com o sistema;
- o objeto de interface delega quase toda a responsabilidade pela lógica do sistema ao objeto controlador;
- as operações `exibirTelaAutenticacao` e `exibirCamposEntradaItensPedido` são, na realidade, sequências de operações de exibição de campos no formulário, correspondendo à invocação de construtores de objetos dos tipos rótulo, caixa de texto e botão com seus correspondentes conteúdos;
- optamos por solicitar ao recém-criado objeto `oNovoPedido`, por meio do objeto `controlador`, a criação do item do pedido. Este, por sua vez, ficou incumbido de obter os dados do produto. Isso facilita a definição da associação entre o pedido e seus itens e os produtos aos quais se referem;
- adicionamos ao diagrama de classes uma classe `CatalogoDeProdutos` associada à classe `Produto` (com multiplicidades 1 para muitos) para indexá-los, facilitando a busca pelo código e a obtenção dos seus atributos.

Como você sabe, é preciso retornar ao diagrama de classes para adicionar as classes que criamos, as operações que descobrimos e suas respectivas

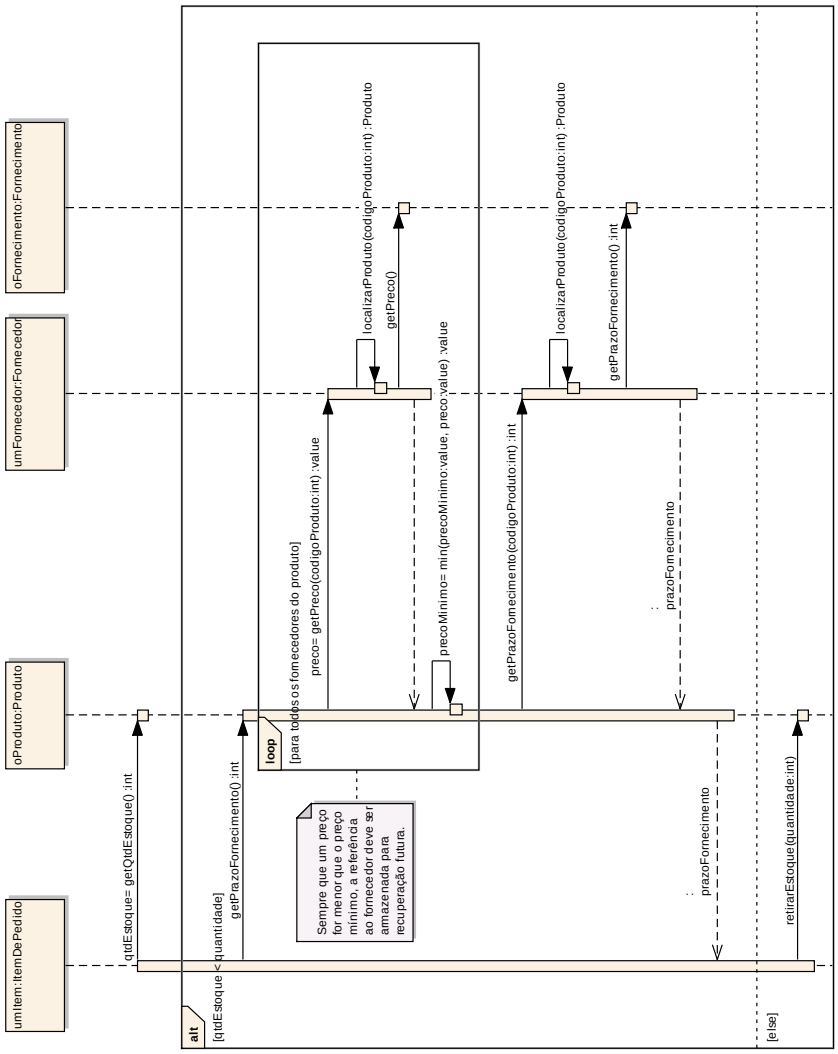


Figura A.25: Uma solução mais completa de colaboração para obtenção do prazo de entrega de um pedido feito à ZYX.

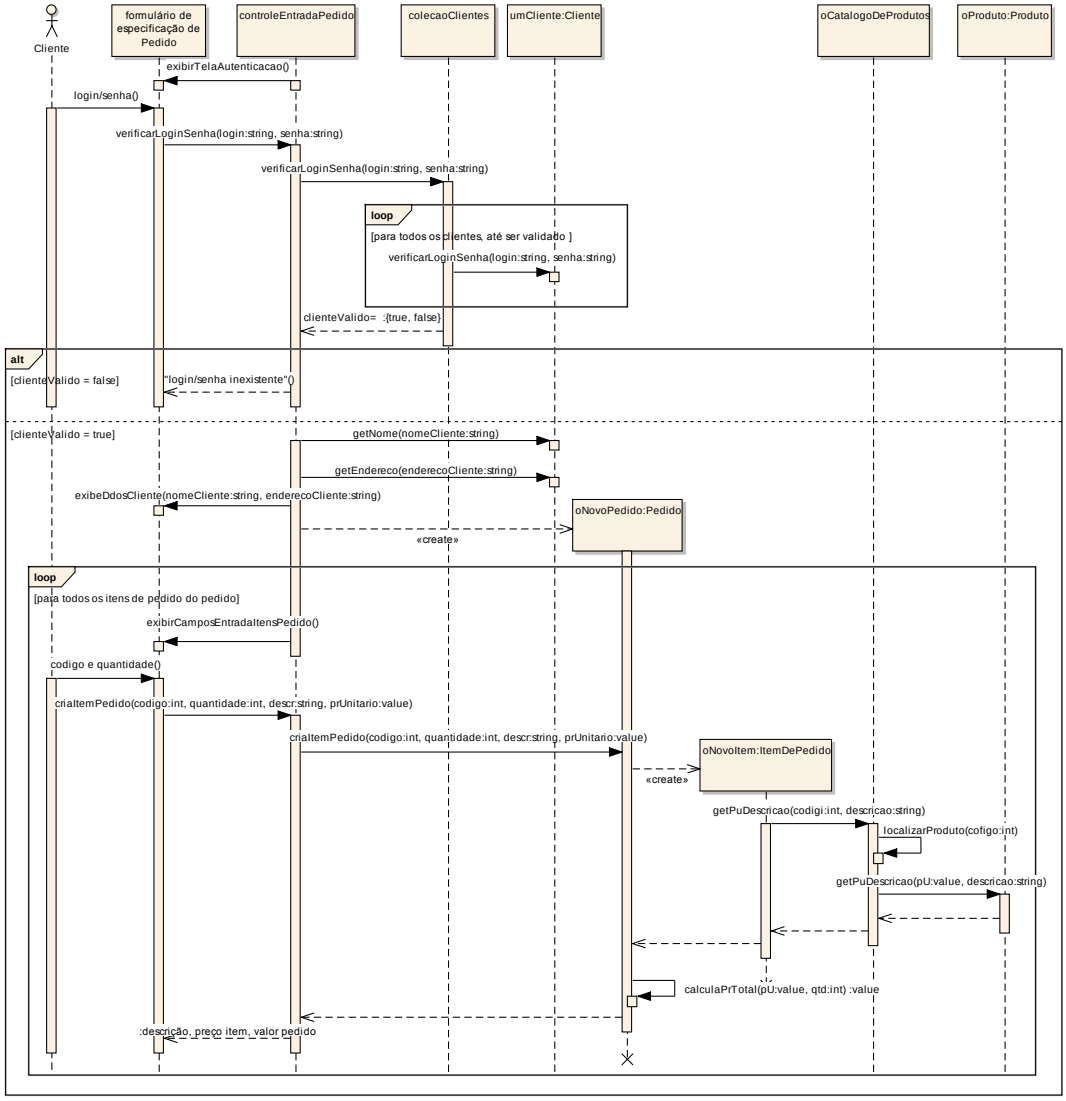


Figura A.26: Uma solução para a colaboração do caso de uso `Efetuar Pedido`.

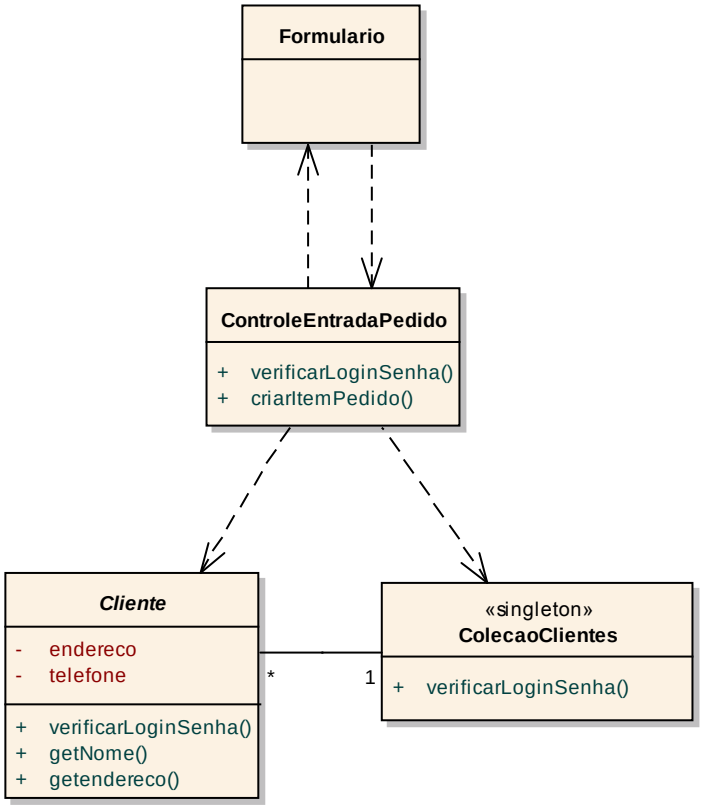


Figura A.27: Relacionamento de dependência entre as classes de interface (formulário), controladora e conceitual resultantes da realização do caso de uso `Efetuar Pedido`.

visibilidades. As classes de interface e controladoras aparecem, no diagrama de classes, relacionadas entre si e às demais classes de conceito e de projeto por meio de relacionamentos de dependência; o formulário depende do (ou usa o) controle que depende da coleção de clientes e dos clientes. Sendo assim, as classes e seu relacionamentos a serem adicionados ao diagrama de classes já existente é o da Figura A.27:

A descrição do caso de uso que elaboramos não está completa nem tivemos a intenção de deixá-la da melhor forma possível. As validações dos dados de entrada, por exemplo, não foram feitas por questões de simplificação do diagrama. Nosso objetivo foi elaborar uma solução que permitisse exercitar os conceitos apresentados em um diagrama com complexidade visual dentro dos limites.

A.10 Exercícios do Capítulo 11, página 198:

1. O diagrama de atividade que especifica graficamente o caso de uso encontra-se na Figura A.28. No DA estão especificadas apenas as ações do sistema que estão no trecho descrito. Cabe agora especificar, como quadros de interação, cada uma das ações deste DA. O diagrama de visão geral da interação fica, portanto, como nas Figuras A.29 e A.30.

   Note que, conforme proposto no enunciado do exercício, usamos os trechos de colaboração do DS da solução que apresentamos para o exercício três da Seção 10.11 (Figura A.26).

   Alguns aspectos importantes do diagrama de nossa solução merecem ser destacados:

   - a primeira mensagem em um quadro partiu do objeto que recebeu o controle no quadro anterior no fluxo;
   - um diagrama de visão geral da interação fica mais "espalhado" no papel que seu equivalente DS, mas seu entendimento é mais fácil porque a estrutura de controle de mais alto nível fica fora dos quadros de interação;
   - quadros de interação mais simples produzem diagramas mais fáceis de ser interpretados. No entanto, decidimos elaborar um único quadro contendo as ações de fornecimento dos itens do pedido para aproveitar o quadro `loop`. Isso só foi possível porque a interação dentro do quadro `loop` não é tão complexa, inclusive porque não há condições alternativas nesse trecho de colaboração.

2. A Figura A.31 mostra o diagrama de pacotes que elaboramos.

   Poderíamos, claro, ter colocado um pacote por diagrama (um pacote em cada folha de papel). Só não fizemos isso para mostrar que/como podemos fazer e porque, mesmo com todos os pacotes no mesmo diagrama e as classes representadas dentro deles, ainda conseguimos ler os elementos textuais. Isso, no entanto, não é possível na maioria dos casos, porque os diagramas são usualmente bem complexos (aliás, uma das razões para usar pacotes é para resolver o problema da complexidade visual dos diagramas).

   Separando os pacotes em diagramas diferentes, é bom que tenhamos um diagrama, que tipicamente colocamos no início, que mostra uma visão de mais alto nível. A Figura A.32 ilustra o que falamos, completando nossa atividade.

3. O diagrama de distribuição está representado na Figura A.33.

   Não há muito o que discutir a respeito dessa solução que demos, exceto o fato de que usamos os mesmos relacionamentos de dependência que apresentamos

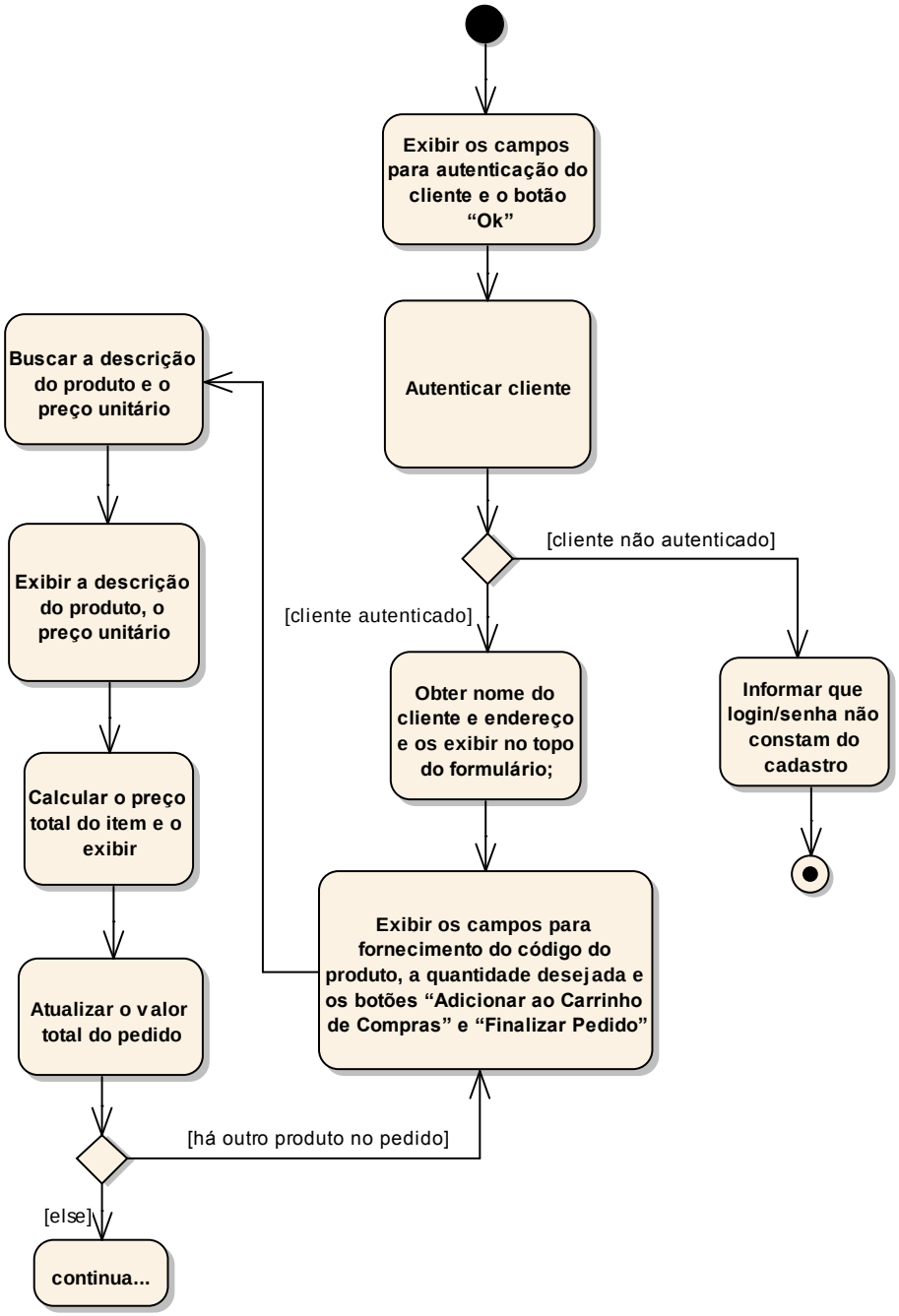


Figura A.28: Diagrama de atividade parcial especificando os passos do caso de uso `Efetuar Pedido`.

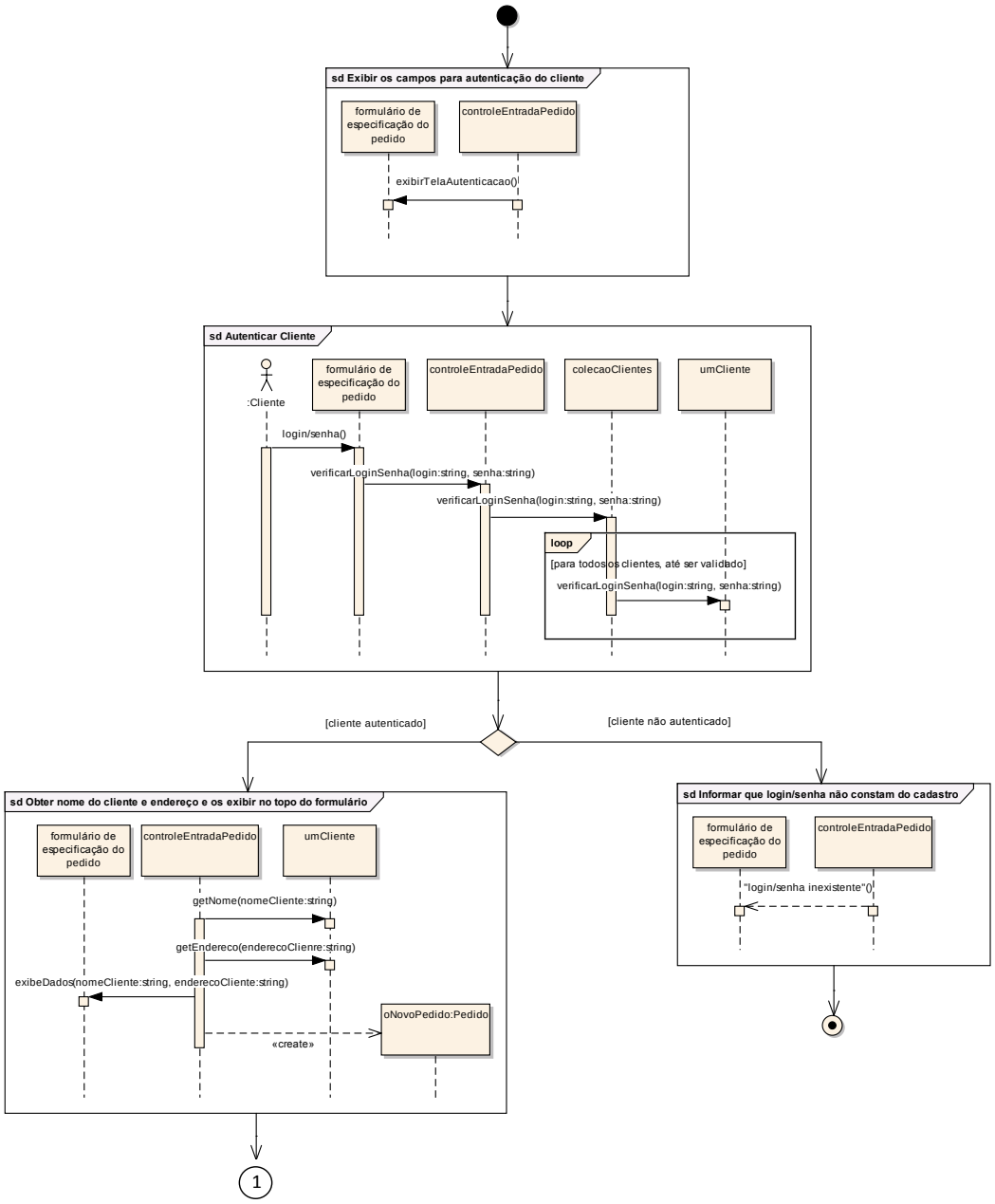


Figura A.29: Primeira parte do diagrama de visão geral da interação especificando as ações do DA da Figura A.28.

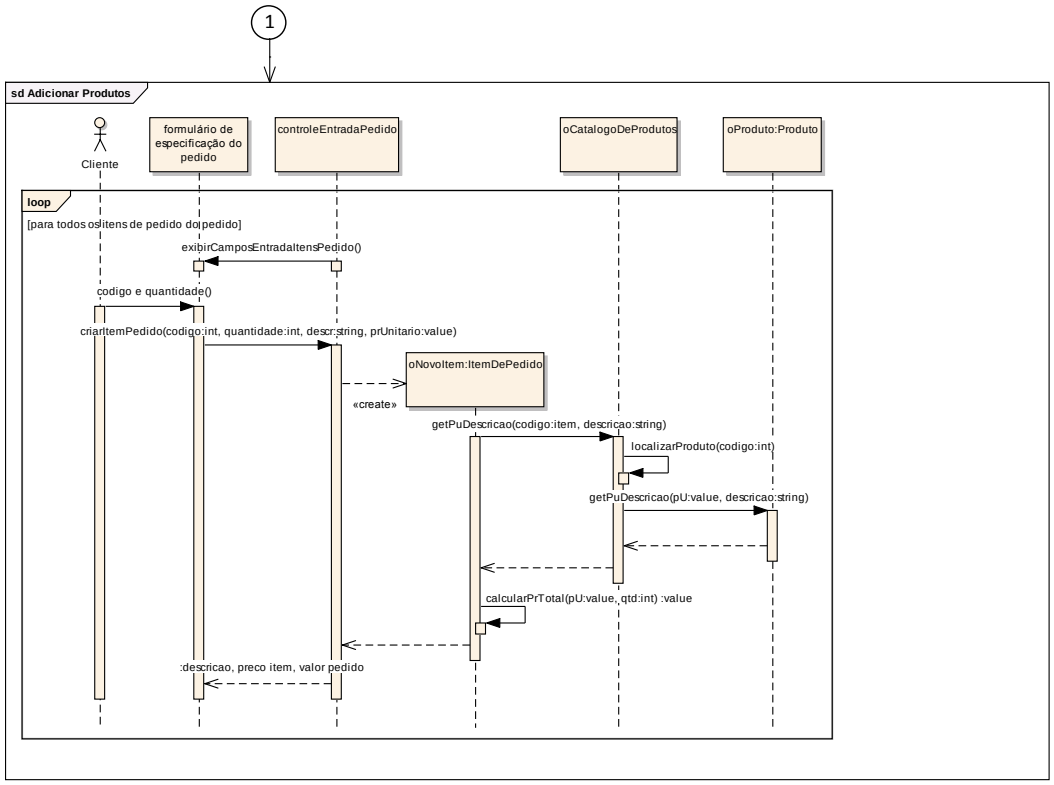


Figura A.30: Segunda parte do diagrama de visão geral da interação especificando as ações do DA da Figura A.28.

na solução do exercício 2. Como componentes apresentam e usam interfaces para comunicação entre eles, o relacionamento de dependência poderia ser substituído pela notação da interface fornecida e exigida da Figura 11.6, considerando que a dependência corresponde à interface exigida e vice-versa, conforme a Figura 11.7.

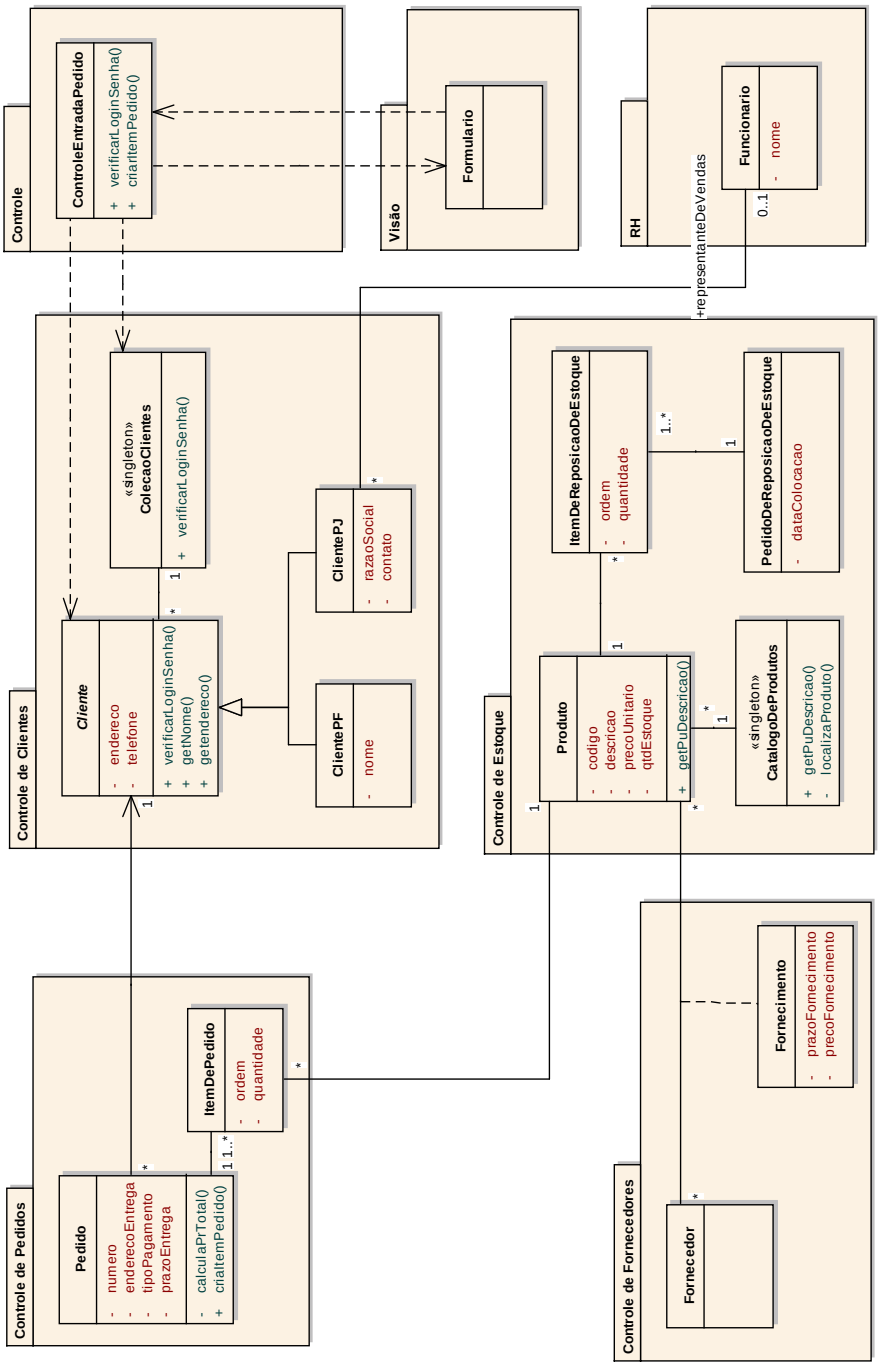


Figura A.31: Classes da ZYX agrupadas em pacotes.

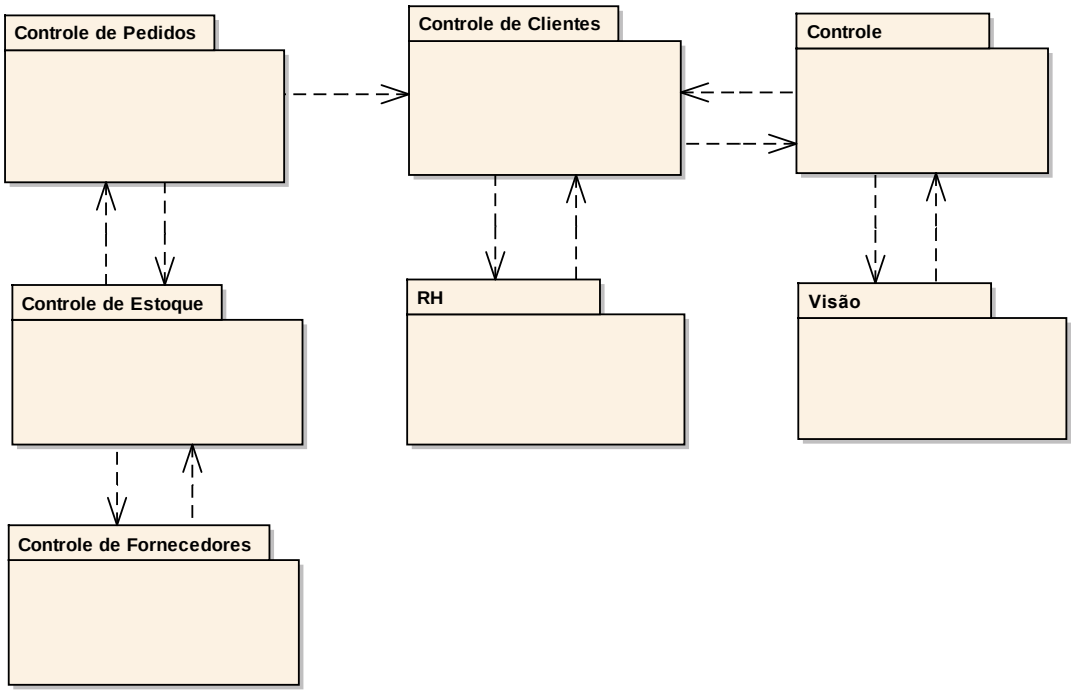


Figura A.32: Diagrama de pacotes de classes da ZYX.

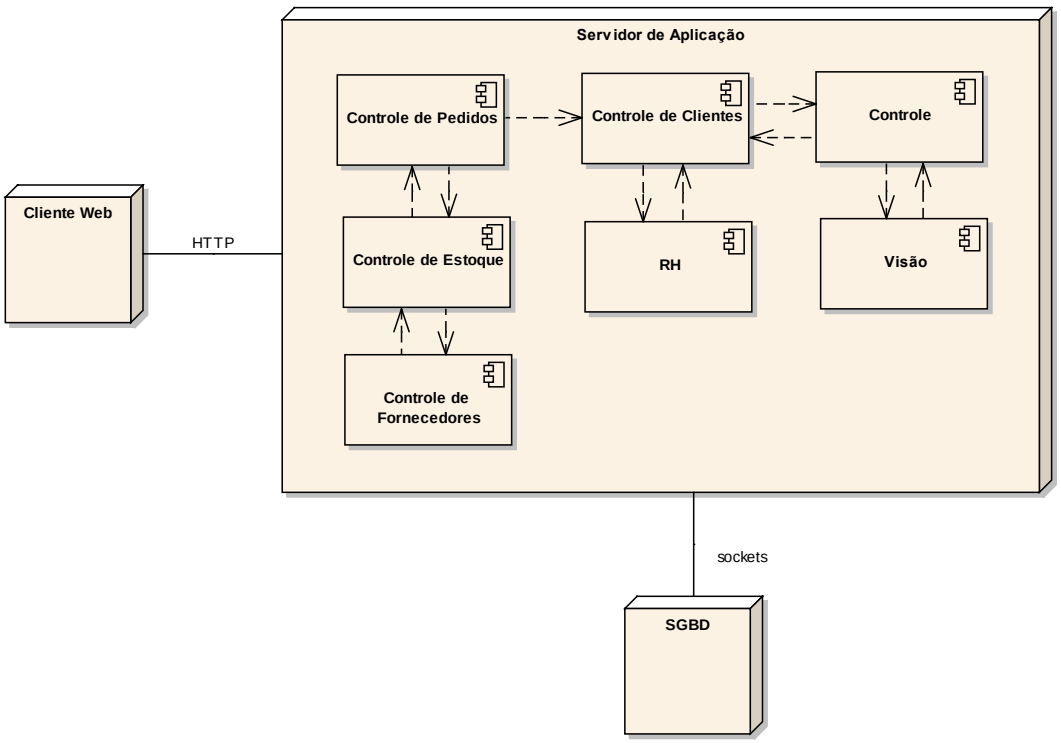


Figura A.33: Diagrama de distribuição dos sistemas da ZYX.

APÊNDICE B

MINIMUNDOS COMPLETOS

B.1 Peixaria Q-Sereia

A peixaria Q-Sereia teve suas vendas aumentadas substancialmente quando passou a fornecer pescado limpo para grandes restaurantes de frutos do mar no Rio de Janeiro. Há algum tempo resolveu entrar no ramo da pesca, adquirindo algumas traineiras e, eventualmente, comprando a produção de embarcações de terceiros. O Sr. Manoel, proprietário da peixaria, procurou nossa empresa, a Cooperativa de Competentes Engenheiros de Software – CCE-S –, com a finalidade de contratar o desenvolvimento de um sistema computadorizado para controle de suas operações. Inicialmente é necessário desenvolvermos o Sistema de Controle da Produção e Vendas – SCPV –, conforme descrevemos a seguir.

É necessário que o SCPV armazene a matrícula na Capitania dos Portos e a capacidade em toneladas de qualquer embarcação. Para as embarcações próprias, é necessário que o sistema armazene, também, a capacidade em litros do tanque de diesel, além do status da mesma (situações possíveis são descritas ao final). Para as embarcações de 3os., apenas uma referência à empresa proprietária é necessária (existe um cadastro de empresas terceiras mantido pelo Sistema de Contas a Pagar que é consultado no cadastramento de uma nova embarcação de terceiros).

Embarcações (quaisquer) saem em missões de pesca para as quais são necessários os registros da data de saída, data prevista de chegada, data da efetiva chegada de volta ao porto. Os registros de suas produções são feitos no sistema

pelo Auxiliar Administrativo quando as embarcações voltam de suas missões, sendo compostos de itens de produção (zero ou mais), dos quais constam a espécie de pescado e o peso bruto. As espécies são cadastradas no SCPV através de seus nomes populares. Do cadastro de espécies constam, também, o percentual estimado de perda, os preços por quilo bruto e limpo de cada espécie.

O Capitão possui um número de registro de habilitação na Capitania. Também como parte do cadastramento de uma nova missão, todos os tripulantes (inclusive o Capitão) que a realizarão são relacionados. As missões são definidas no sistema na seguinte seqüência: o Auxiliar Administrativo cadastra a data de saída, data prevista de chegada. Em seguida informa a matrícula da embarcação na Capitania dos Portos. Associa os tripulantes (pré-cadastrados, quando funcionários da Q-Sereia) à nova missão. Se a embarcação ainda não se encontra cadastrada, esse é o momento de fazê-lo. Para embarcações de terceiros é necessário o cadastramento de todos os tripulantes, fornecendo-se seus nomes e telefones em terra para eventual necessidade de contato.

A chegada de uma missão ao porto é informada pelo Capitão, via rádio, ao Auxiliar Administrativo para que este informe a data/hora do fim da missão ao SCPV. A embarcação passa a ser, então, descarregada, quando o Capitão coordena a separação e peso por espécie do produto da pesca, preenchendo um formulário em duas vias com esses dados (além da data/hora da chegada ao porto). Os dados da produção são conferidos pelo Encarregado do Estoque que rubrica o formulário após a conferência. O Capitão dirige-se, então, ao escritório da peixaria no porto e entrega o diário de bordo (no caso de embarcações próprias da peixaria), juntamente com as duas vias do formulário. O Auxiliar Administrativo, após verificar o preenchimento correto do formulário, carimba "Recebido" na segunda via, entregando-a ao Capitão para seu controle. A primeira via é usada pelo Auxiliar para informar os dados da produção ao SCPV e para posterior arquivamento.

No início do cadastramento da produção o sistema deverá solicitar a licença da embarcação, que já identificará se a embarcação é própria (o que acontece na maioria das vezes) ou de terceiros. O SCPV passa a solicitar os dados da produção. Ao final do cadastramento da produção o sistema deverá "passar", automaticamente, os dados da mesma aos Sistemas de Controle de Estoque (SCE), ao de Folha de Pagamento (SFP), se a embarcação é própria (parte do pagamento da tripulação dos barcos próprios da Peixaria é função da produção), ou de Contas a Pagar (SCP), se embarcação é de terceiros, pois parte do pagamento pelo aluguel do barco/tripulação às terceiras é função da produção.

É também tarefa do Auxiliar o cadastramento no SCPV dos dados das espécies de pescado, usados também pelo sistema de contas a pagar para pagamento dos terceiros. A informação da perda média pode ser alterada pelo Sistema de Controle de Estoque

(SCE), baseado nas quantidades de cada espécie que entram e nas quantidades que são efetivamente estocadas após a limpeza.

Os Vendedores devem dispor de uma rotina de cadastramento de vendas, que verifica as quantidades disponíveis em estoque, verifica se os restaurantes compradores estão na "lista negra" e que processa a venda, solicitando ao SCE a baixa no estoque e criando novo compromisso no Contas a Receber.

A lista negra de restaurantes inadimplentes é mantida pelo próprio Sr. Manoel, como outra funcionalidade do sistema SCPV. Um aviso é enviado ao Contas a Receber quando um restaurante é colocado/retirado na/da lista negra.

Qualquer embarcação própria da Q-Sereia pode estar em uma das seguintes situações:

1. disponível (quando está pronta, aguardando nova missão),
2. em missão,
3. descarregando (após chegada ao porto),
4. em avaliação (após ser descarregada, quando é feita uma limpeza e avaliação para verificação da necessidade de reparos antes de uma nova missão), e
5. em reparos (quando, na avaliação, é constatada a necessidade de reparo(s)).

OBS: Todos os sistemas da Q-Sereia se comunicam/comunicarão diretamente através da rede interna e as mensagens são trocadas usando-se um protocolo que garante e informa os seus recebimentos corretos, ou seja, todos os sistemas estão/estarão online.

B.2 Empresa 5-E

Fomos contratados para o desenvolvimento dos sistemas necessários à informatização das atividades administrativas da Empresa de Engenharia Eletroeletrônica Excelência (5E) Ltda. Para tal, será preciso que desenvolvamos os sistemas de Gestão de Pessoal (SRH), de Controle de Fornecedores e Estoque (SCFE), de Controle Financeiro (SCF) – que englobará contabilidade, aplicações financeiras e os controles do "contas a pagar" e do "contas a receber" – além do Sistema de Gestão de Contratos – SGC –, que é o mais urgente e deverá ser o primeiro a ser desenvolvido por completo. Todos esses sistemas operarão de forma integrada, usarão a tecnologia web (servidores e Clientes web) e se comunicarão, exclusivamente, através de trocas automáticas em tempo real de

mensagens eletrônicas em XML (todos os sistemas estarão perfeitamente integrados). O SGC é descrito a seguir. Os itens a seguir descrevem as funcionalidades do sistema e os demais itens descrevem os principais aspectos estruturais e dinâmicos do mesmo.

1. Os Auxiliares Administrativos (Auxiliares) da 5E poderão cadastrar contratos e, durante o cadastramento dos mesmos, poderão incluir os dados de novos Clientes. Caso os dados de um Cliente já estejam disponíveis no sistema, os Auxiliares simplesmente farão a associação do novo contrato ao Cliente já existente. Como uma funcionalidade isolada do sistema, deverá ser possível o cadastramento de Clientes (Clientes em potencial, sem que eles venham a ser, de imediato, associados a qualquer contrato). No final do cadastramento de um contrato será escolhido pelo sistema um Gerente de Contrato (Gerente) num esquema de escalonamento round robin. O Gerente escolhido é avisado pelo sistema (deverá ser feita a tentativa de exibir-se uma janela popup na estação de trabalho do Gerente, se ele estiver "logado" no SGC, o que demandará a confirmação pelo mesmo) para que se encarregue do "fechamento" (formalização /celebração) do contrato.

2. O processo de cadastramento de um contrato ocorre da seguinte forma: o sistema solicita ao Auxiliar o CPF ou CNPJ do Cliente. O sistema busca e exibe, quando já existentes no cadastro, os dados do Cliente para confirmação pelo Auxiliar (o que pode ou não acontecer). Caso os dados não estejam cadastrados, o que é mais comum, essa é a hora de informá-los. O sistema, então, solicita o escopo principal do contrato (se de eletricidade ou de eletrônica, se de projeto ou de manutenção) e os demais dados (vide item IX). O Auxiliar passa a informar a duração do contrato, o valor e, caso o contrato seja de manutenção (caso mais comum), o local de prestação dos serviços. Antes ainda do final do cadastramento de um contrato, o mesmo será associado a um único Gerente de Contrato (funcionário da 5E), conforme já mencionamos anteriormente. Como última etapa do cadastramento de um contrato, o sistema exibirá o identificador único do contrato (Número do Contrato) obtido automaticamente para referências futuras ao mesmo.

3. Os contratos cadastrados e ainda não fechados serão entendidos como "minutas de contratos" e, nesse estado, poderão ser alterados a qualquer momento pelos Auxiliares ou seus Gerentes.

4. Deverá existir uma outra funcionalidade do SGC para informar o fechamento de um contrato já cadastrado, quando é fornecida a data de início (a duração já está no corpo do contrato), são impressas duas cópias do mesmo e é emitido um carnê de pagamento, consistindo de um ou mais boletos de pagamento bancário. Contratos fechados não poderão ser alterados. O fechamento de

um contrato é feito pelo Gerente de Contrato a ele associado, que poderá, a qualquer momento, consultar em tela e imprimir cópias dos contratos pelos quais é responsável.

5. Após a associação de contrato a um Gerente, este poderá adicionar ao contrato notas (comentários/observações) em qualquer número. Um comentário poderá estar associado a uma data/hora para que o Gerente o receba, como um lembrete pelo sistema, na data e hora especificados.

6. Os diretores da 5E poderão executar no sistema as mesmas funcionalidades que os Gerentes de Contrato. Adicionalmente poderão informar o cancelamento de um contrato (atividade essa que será exclusiva dos diretores) juntamente como o motivo do cancelamento.

7. Às 8:00h de todo início de semana, o sistema deverá produzir automaticamente uma relação impressa de todos os contratos ainda em aberto (que ainda não foram fechados).

8. É importante que, quando um contrato se tornar vencido, uma comunicação do fato seja feita ao respectivo Gerente.

9. Com relação aos dados dos contratos, estes poderão ser de projeto ou de manutenção. Qualquer contrato terá sua data de início e duração em meses, além do escopo, que pode ser "ELE" (eletricidade) ou "ELO" (eletrônica). Contratos de projeto possuirão uma descrição e contratos de manutenção possuirão a relação dos equipamentos cobertos. Equipamentos cobertos serão especificados através da marca, modelo e número de série. Um contrato estará associado a um único Cliente. Clientes possuirão qualquer número de contratos no SGC. Um contrato será associado a um único Gerente. Gerentes poderão gerir qualquer número de contratos.

10. Clientes são pessoas físicas ou jurídicas. No primeiro caso, será necessário o armazenamento do nome, endereço, telefone e o CPF; no segundo caso será necessário o armazenamento do endereço, um nome de contato, o telefone, a razão social e o CNPJ.

11. Para o cancelamento de um contrato, será necessário que a data/hora e os motivos do cancelamento fiquem registrados no sistema, além de uma referência ao Diretor que cancelou o contrato.

12. Um contrato poderá estar "Aberto", situação em que se encontrará logo após o cadastramento. Passará a "Fechado" quando o Cliente e a 5E concordarem com os termos e celebrarem o acordo. Se tornará "Vencido", quando findo o prazo de duração, e "Cancelado", quando o cancelamento for informado ao SGC por

um Diretor. O cancelamento só poderá ocorrer enquanto o contrato vigorar. É possível que, através de um processo de renovação, um contrato vencido possa voltar a vigorar.

B.3 Sistema de Controle de Ordens de Serviço – Refrigeração ManutAir

A empresa ManutAir Ltda. tem como atividade a prestação de serviços de manutenção preventiva e corretiva de equipamentos de condicionamento de ar. O quadro de funcionários da empresa é composto de técnicos em ar condicionado e pessoal administrativo.

A empresa possui contratos com diversos Clientes, contratos esses que podem ser de dois tipos: contratos com cobertura total (peças e mão-de-obra) ou contratos com cobertura parcial (mão-de-obra somente). Além disso, a empresa efetua manutenção corretiva em equipamentos não cobertos por nenhum contrato pré-existente. Nesse caso é criado um contrato, chamado avulso, de cobertura total (não alterável – um para cada equipamento a ser reparado), de duração de 90 dias para cobertura das peças trocadas e mão-de-obra envolvidos no reparo, conforme determina o PROCON.

Quando um Cliente fecha um novo contrato, ele deve informar ao Atendente a razão social, endereço, CNPJ, nome e telefone do responsável, para o caso de Cliente PJ, ou nome, endereço, telefone e CPF, para o caso de Cliente PF. O sistema busca e exibe os dados do Cliente (tipicamente os dados já se encontram cadastrados no sistema). Caso os dados do Cliente ainda não se encontrem cadastrados, esse é o momento de fazê-lo. Em ambos os casos o Cliente informa, também, a lista dos equipamentos cobertos pelo contrato (através de suas marca+modelo+número de série), a data de início da vigência e o prazo de duração em meses.

Os contratos, após cadastrados, recebem do sistema um número. No fechamento de um contrato não avulso também é emitido um carnê de pagamento para o Cliente, correspondente às parcelas mensais a serem pagas durante a vigência do contrato. Também no caso de contrato não avulso, a lista de equipamentos cobertos pode ser alterada com a inclusão ou exclusão de novos equipamentos. É feito, se cabível, o correspondente reajuste do valor das cotas mensais. Os equipamentos incluídos estão sujeitos à carência, a critério do Supervisor. No caso de contratos avulsos, o pagamento é feito através de um boleto bancário, que é emitido após a conclusão do serviço. Em ambos os casos, o banco envia à ManutAir a relação impressa dos pagamentos recebidos a cada dia para que seja feita a conciliação bancária pelo Atendente.

Os Clientes, quando necessitam de algum atendimento, ligam para o número telefônico de solicitação de serviço. As chamadas são recebidas pelos Atendentes que fazem a abertura das Ordens de Serviço (OS), deixando-as com status "Aberta". Para tal, os Atendentes solicitam ao Cliente o número do contrato (que tipicamente já está cadastrado no sistema), o equipamento que necessita reparo (marca+modelo+número de série), o endereço onde este se encontra e uma breve descrição do problema.

Caso o equipamento não esteja coberto por nenhum contrato, é preparada uma OS especial (correspondente a um contrato avulso) para que se faça um atendimento corretivo.

O Supervisor Técnico disporá de uma funcionalidade para consulta e alocação de novas OS abertas aos técnicos de campo, o que é feito num esquema de rodízio. Ao fazer a alocação de uma OS a um técnico o Supervisor anota o dia, a hora marcada para a visita do Técnico, deixando a OS com status "Em Andamento". Nessa oportunidade é emitida uma cópia impressa da OS que é colocada na caixa de entrada do técnico. Em casos de urgência o Supervisor contata os técnicos via Nextel.

Os Técnicos realizam as visitas aos Clientes, onde prestam o atendimento solicitado que, normalmente, é encerrado na visita inicial. Após um atendimento, o técnico entra em contato com o Supervisor e informa, via Nextel, a situação da OS. Caso seja necessária a troca de peças, o Supervisor solicita as peças ao estoque através de uma funcionalidade do sistema. A OS assume, então, o status "Aguardando Peça". Quando as peças ficam disponíveis para a OS, o Estoquista informa o fato no sistema, levando a OS ao estado de "Material Disponível". Através de uma funcionalidade de consulta a OSs pendentes, o Supervisor é informado da disponibilidade de peça para que possa marcar uma nova visita ao Cliente. Quando a nova visita é marcada, a OS é reativada (retornando ao status "Em Andamento") e uma nova comunicação impressa ao Técnico é gerada.

Após a conclusão do serviço, o Técnico informa (via Nextel) o fechamento da OS ao Supervisor, informando o total de horas gastas no reparo, bem como o material utilizado. A OS assume, então, o status "Concluída" quando, se necessário, é iniciada sua cobrança. Após o recebimento do pagamento, a OS passa para "Encerrada". No caso de OS com cobertura total pelo contrato, ao ser concluído o serviço, a OS é automaticamente encerrada.

A empresa recebe de cinquenta a setenta chamados por dia e trabalha com um Supervisor, dois Atendentes, quinze Técnicos de campo e um Estoquista.

O dono da empresa deseja dispor de um sistema informatizado que permita a gestão dos dados dos contratos e que controle todas as etapas de uma chamada, desde o momento do registro até a finalização do serviço.

B.4 Sistema de Acompanhamento de Entregas da Rapidão Espacial

A empresa de vendas pelas Internet *LatinoamericanasPontoCom* (LAPC) possui um sistema de vendas com rastreamento detalhado dos passos dos pedidos feitos por seus clientes enquanto os pedidos se encontram em suas dependências. Entretanto, como efetua parcerias com empresas de entrega dos pedidos nas diversas praças do País, observa uma grande insatisfação de seus clientes quanto à falta de informações precisas das entregas após os pedidos serem deixados nas transportadoras. A insatisfação consubstancia-as na grande quantidade de reclamações e até mesmo no retorno das mercadorias à LAPC. Em consequência, a LAPC passou a se interessar por parcerias com empresas de entregas que disponham de sistemas informatizados que possam ser integrados eletronicamente ao sistema de vendas da LAPC, de forma a mantê-la minuciosamente informada, para que ela possa repassar as informações a respeito das entregas a seus clientes.

Nossa cliente, a Rapidão Espacial Ltda (RE), com filiais nas principais cidades do Brasil e com rede de entregas que cobre quase a totalidade de municípios brasileiros é uma das principais transportadoras da LAPC. A RE, interessada nessa "fatia" de mercado – pois observa que essa é também uma necessidade de várias outras empresas de vendas pela Internet –, contratou nossos serviços para o desenvolvimento de um sistema que permita o acompanhamento detalhado de cada etapa de uma entrega, tanto por parte dos clientes compradores da LAPC quanto por parte da própria LAPC. Em nossos entendimentos iniciais definiu-se uma relação de requisitos para o novo sistema que apresentamos ao longo do texto a seguir.

Quanto à comunicação LAPC/RE, toda e qualquer comunicação ligada ao negócio entre a LAPC e a RE ocorrerá por via eletrônica (link de dados dedicado, ainda a ser instalado), utilizando um protocolo já definido pela LAPC. As mensagens trocadas entre a RE e LAPC (nos dois sentidos) serão usadas para atualização em tempo real dos sites tanto da RE quanto da LAPC, permitindo que o cliente comprador acompanhe a entrega acessando, via Internet, qualquer um dos sites. As mensagens geradas durante o processo de entrega de um pacote são centralizadas no sistema da RE para posterior re-envio para a LAPC, caso necessário.

Quanto à entrada de informações no novo "Sistema RE de Acompanhamento de Entregas" – SREAE –, identificamos que as informações poderão chegar ao SREAE de três formas:

1. provir diretamente da LAPC, através do mecanismo de mensagens já definido e apresentado acima;

2. provir dos computadores portáteis dos funcionários de campo da RE, que utilizam moderníssima tecnologia de computação móvel, com comunicação via rádio com os escritórios da RE e com capacidade de leitura de códigos de barras, ou

3. da forma convencional, através de entrada manual de dados, via teclado.

As características e necessidades do SREAE são descritas a seguir.

Quanto ao despacho inicial das mercadorias, diariamente, pela manhã, a LAPC embala as compras em pacotes individuais, por destinatário. Cada pacote recebe um identificador da forma `LAPCPCTXXXX`, onde `XXXX` é um número de sequência de pacote gerado pelo sistema já existente da LAPC.

Os pacotes recebem uma etiqueta contendo as informações relevantes para o despacho: o identificador (também em código de barras), o nome, endereço e CEP do destinatário, o número do lote (ver adiante), o número da nota fiscal e o peso e dimensões do pacote. Também é afixado ao pacote um envelope plástico contendo cópia da nota fiscal de venda dos produtos que vão no pacote.

Um conjunto de pacotes (o *lote*, formado a critério da LAPC) recebe um identificador de lote do tipo `LAPCLOTXXXX`, onde `XXXX` é o número de sequência de lote, gerado pelo sistema da LAPC. Cada lote é deixado no balcão da RE que fica junto ao setor de despacho de cargas da LAPC.

Nesse momento a LAPC manda uma mensagem ao SREAE informando que existe um lote de mercadorias no depósito da RE a ser despachado, passando, também, os dados de cada pacote (as mesmas info. contidas em cada etiqueta). A RE confere a relação e, caso "OK", envia uma mensagem à LAPC informando da aceitação do lote. Nesse momento também manda mensagens (uma para cada pacote) à LAPC, informando que a entrega foi iniciada e que o pacote está em *triagem* no depósito de origem. Em função do destino final de cada pacote, o SREAE deverá informar todos os despachos intermediários que serão necessários (roteamento). Essa função não é 100% automática, ou seja, é considerada como uma função de auxílio ao Auxiliar Administrativo da RE na definição das rotas e despachos intermediários.

Em seguida inicia-se, efetivamente, a triagem inicial, quando os pacotes são separados manualmente por destinos imediatos (aeroporto tal, rodoviária tal, etc). Os funcionários da RE responsáveis pela entrega de cada novo lote nos respectivos destinos imediatos, *scaneiam* os pacotes e, ao final, transferem essas informações ao SREAE. Passam, então, ao embarque dos pacotes nas viaturas da RE para serem entregues em cada destino imediato. Ao fazerem a entrega de um lote, disparam uma mensagem de entrega de lote para a RE que, ao mesmo tempo que atualiza seu site, trata de mandar mensagens à LAPC com o novo estado de cada pacote. O sistema

envia, também, mensagens às localidades de destino (intermediárias e/ou finais) para que agendem a busca e, possivelmente, redespacho dos pacotes.

Quanto aos despachos intermediários das mercadorias, um pacote pode ser roteado muitas vezes (recebido de um transportador intermediário e enviado a outro) até que chegue ao município destino, configurando-se, nesse caso, uma *entrega local*. Sempre que um pacote é deixado em um destino (final ou intermediário) seu identificador é passado à RE para que produza uma atualização no *site* e para que seja gerada uma mensagem para a LAPC.

Ao sair para a coleta dos pacotes que estão chagando, o funcionário local da RE já dispõe da relação desses pacotes carregada em seus computadores portáteis. Dessa relação consta, para cada pacote, seu destino imediato (redespacho ou entrega local). Se há pacote esperado que, porventura, não tenha sido recebido, é necessário que o SREAE receba uma mensagem de que o pacote extraviou, para que seja iniciada sua busca.

Quanto às entregas locais, ao receber uma mercadoria para entrega no mesmo município ou logradouro próximo, o SREAE informa à LAPC que uma entrega local foi iniciada. A(s) mercadoria(s) será(ão) armazenada(s) temporariamente para entrega final no dia seguinte. É iniciada a triagem final, onde o roteiro de entrega deverá ser definido com a ajuda de uma função do sistema.

O SREAE produzirá uma relação dos bairros ou distritos das entregas, colocados segundo a sequência definida com ideal, com as respectivas horas de entrega aproximadas. Essa informação também é enviada à LAPC. Entregas nas grandes cidades onde, tipicamente, a quantidade de entregas é grande e onde o trânsito é complicado, o que pode comprometer o planejamento inicial, as viaturas da RE são ligados via rádio às filiais e transmitem, com a periodicidade programada, as suas coordenadas obtidas de um equipamento GPS (*Global Positioning System* – Sistema de Posicionamento Global). Essa informação é usada pelo sistema da RE para recalcular as horas previstas de entrega nos diversos bairros.

Outras necessidades do SREAE, além das descritas anteriormente, são:

Função para alteração da rota de um pacote: executada em qualquer ponto intermediário da rota de um pacote, pelo funcionário de campo da RE. Essa função deverá manter a consistência com todos os demais passos para a entrega de um pacote.

Função para alteração em campo das rotas urbanas: executada em qualquer ponto de um trajeto urbano, basicamente em função de imprevistos no trânsito (acidentes, obras urbanas, etc.) que possam alterar significativamente o trajeto.

Função de manutenção dos dados da rede de cobertura RE: para manter a relação

de municípios cobertos pela RE, da rotas pré-estabelecidas para eles e os respectivos custos de envio

Funções de consulta aos municípios cobertos pela RE: para permitir a consulta, via Internet (para o público em geral) e via mensagem (para a LAPC), dos municípios cobertos e dos custos de envio.

Função de faturamento pelos serviços de transporte prestados à LAPC: mensalmente a RE emite fatura discriminada sos serviços prestados à LAPC.

Função de consulta ao histórico e situação de um pacote: para a obtenção, via Internet, do histórico e da situação atual de um pacote. Isso pode ser feito a partir do número de identificação do pacote ou a partir do `Nome` e `CPF` do cliente comprador.

Função de suspensão da entrega de um pacote: a LAPC pode solicitar a suspensão de uma entrega a qualquer momento, o que implica que o pacote deve ser roteado de volta à LAPC. A cobrança pelos serviços da RE é recalculada e o novo valor é informado à LAPC.

Função de alteração do endereço de entrega: a LAPC pode solicitar a alteração do endereço de uma entrega a qualquer momento, o que implica que o pacote deve ser re-roteado para o novo destino. A cobrança pelos serviços da RE é recalculada e o novo valor é informado à LAPC.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] Grady Booch, James Rumbaugh, and Ivar Jacobson. *UML: Guia do Usuário*. Editora Campus Ltda, 5a. tiragem edition, 2000.

[2] Alistair Cockburn. *Writing Effective Use Cases*. Addison-Wesley, 2001.

[3] Sthephen M. McMenamim e/and John F. Palmer. *Análise Essencial de Sistemas*. Makron Books Editora Ltda., 1st edition, 1991.

[4] Chris Gane e/and Trish Sarson. *Análise Estruturada de Sistemas*. Livros Técnicos e Científicos, 1983.

[5] Martin Fowler and Cris Kobryn. *UML Essencial*. Bookman, third edition, 2004.

[6] Martin Fowler and Kendall Scott. *UML Essencial*. Bookman, second edition, 2000.

[7] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Professional, 1995.

[8] Craig Larman. *Utilizando UML e Padrões – Uma Introdição à Análise e ao Projeto Orientados a Objetos e ao Processo Unificado*. Bookman, 3rd edition, 2008.

[9] Suzan Lilly. How to avoid use-case pitfalls. `http://www.ddj.com/architect/184414560`, acesso em maio de 2009.

[10] OMG. *Unified Modeling Language Infrastructure (OMG UML)*, 2.2 edition, Fevereiro 2009.

[11] OMG. *Unified Modeling Language Superstructure (OMG UML)*, 2.2 edition, Fevereiro 2009.

[12] Roger S. Pressman. *Engenharia de Software*. McGraw-Hill, 5 edition, 2002.

ÍNDICE REMISSIVO

ISBN 978-85-911695-0-4