

ANÁLISE ORIENTADA A OBJETOS

Maurício Acconcia Dias

[Ver anotações](#)

Deseja ouvir este material?

Áudio disponível no material digital.

CONHECENDO A DISCIPLINA

A disciplina Análise Orientada a Objetos é um dos principais pilares dos cursos de tecnologia da informação. Isto ocorre porque o desenvolvimento de softwares, atualmente, independentemente de seu objetivo, possui um alto nível de complexidade e utilizar ferramentas que auxiliam no processo de desenvolvimento de software é fundamental para o sucesso do projeto. A utilização dessas ferramentas na modelagem do sistema, ou seja, na fase anterior ao processo de desenvolvimento/codificação é fundamental, pois possibilita minimizar ou até evitar totalmente erros de implementação. Um exemplo de erro evitável por meio do uso de ferramentas de modelagem de sistema é o esquecimento ou implementação equivocada de algum requisito solicitado pelo cliente, por conta de falha no entendimento do projeto. A análise e a modelagem por meio dessas ferramentas podem evitar esses equívocos e minimizam retrabalho após a entrega do produto. Assim, é possível utilizar diversas ferramentas de análise (modelagem), projeto e desenvolvimento para auxiliar o processo de criação, implementação e teste de software. Tais ferramentas e as técnicas

auxiliam e viabilizam que o projeto esteja de acordo com a especificação e a documentação do projeto que permite que times grandes de desenvolvimento obtenham informação confiável da mesma fonte, evitando erros no momento do desenvolvimento. Isso é de extrema importância, pois esses erros podem gerar custos adicionais ao projeto e podem impactar no orçamento da empresa. Tais custos adicionais podem estar associados ao atraso da entrega do produto e a possíveis correções devido a erros no desenvolvimento, o que reflete no pagamento de horas a mais de desenvolvimento necessárias tanto antes da entrega quanto após a entrega do software ao cliente para correção.

Além disso, tais técnicas permitem antecipar os problemas enfrentados, auxiliam nas soluções e são de grande importância na área de desenvolvimento.

A UML, cuja sigla em inglês significa *Unified Modeling Language* ou Linguagem de Modelagem Unificada, é uma das principais ferramentas de modelagem utilizadas em empresas de desenvolvimento de software.

Sua utilização permite que a etapa de análise de requisitos seja feita de forma eficiente gerando a documentação necessária para que todo o processo de desenvolvimento ocorra dentro do planejamento.

Ao estudar os conteúdos apresentados neste material você será capaz de analisar um problema com foco na utilização da linguagem UML, o que é encarado atualmente como um conhecimento essencial para um desenvolvedor de software no mercado de trabalho.

E, considerando este cenário, este material irá abordar os principais aspectos relacionados a UML: inicialmente (unidade 1), o histórico e as principais definições são apresentados com objetivo de demonstrar a importância da linguagem e sua aplicabilidade, em seguida serão apresentadas as formas de modelagem dos diversos diagramas

utilizando a linguagem UML começando com os mais básicos (casos de uso, classes e atividades na unidade 2) e seguindo com os mais específicos (máquina de estados, sequência e demais diagramas de interação na unidade 3) e, por fim, na unidade 4, um caso de uso será apresentado com todas as etapas para que fique claro como todo o processo apresentado anteriormente é realizado.

Caro aluno, a disciplina Análise Orientada a Objetos é de extrema importância em sua formação para a análise e o desenvolvimento de sistemas. Após concluir-la, você estará preparado para realizar a modelagem de sistemas para, posteriormente, realizar a etapa de desenvolvimento/codificação. Além disso, poderá se aprofundar nos desafios que o mercado de trabalho atual apresenta, especialmente porque a UML é amplamente utilizada em diferentes empresas antes de iniciarem o desenvolvimento do software propriamente dito. Essa linguagem é fundamental para reduzir possível retrabalho no ciclo de desenvolvimento de software. Então, prepare-se para, a partir deste momento, iniciar seus estudos, conhecer essa importante ferramenta e se tornar um excelente profissional.

Bons estudos!

NÃO PODE FALTAR

FUNDAMENTOS DA UML

Maurício Accocia Dias

Ver anotações 0

A LINGUAGEM UML

A UML é uma linguagem de modelagem visual de soluções de problemas que torna a modelagem de software universal e de simples entendimento.



Fonte: Shutterstock.

Deseja ouvir este material?

Áudio disponível no material digital.

CONVITE AO ESTUDO

Diversas empresas atualmente, independentemente de seu tamanho, necessitam de serviços relacionados ao desenvolvimento de software. Como exemplo é possível citar os sistemas web de divulgação, cadastro de clientes, e-commerce, softwares de controle de estoque, gerenciamento de clientes e inteligência de negócios. Diversos tipos de clientes buscam empresas de desenvolvimento para solucionar seus problemas e nem sempre estes clientes têm o conhecimento necessário para saber o que desejam de verdade gerando um conjunto de requisitos de desenvolvimento complexo.

Ao fazer parte de uma equipe de desenvolvimento de software é necessário resolver diversos problemas para atender todos os requisitos solicitados. Antes de programar, normalmente, modelamos a solução de alguma forma como por meio de um pseudocódigo, seja um fluxograma ou, até mesmo, desenhos sem seguir um modelo padrão. Esta situação é aceitável quando estamos desenvolvendo um software em casa com objetivos acadêmicos ou de aprendizado, porém em uma empresa de desenvolvimento não será possível atender aos requisitos de forma satisfatória se não for utilizada alguma ferramenta de auxílio, já que os problemas são consideravelmente maiores e mais complexos. Além disso, em empresas, trabalhamos em meio a um time de desenvolvedores o que pode ser bem complexo, assim como você já deve ter percebido em seus trabalhos em grupo. Seria então interessante poder utilizar uma forma de modelar os problemas que todas as pessoas envolvidas em um projeto pudessem entender sem precisar de explicações.

Para entender bem como utilizar uma ferramenta é necessário um estudo de todas as suas principais características. O histórico da criação e desenvolvimento da ferramenta será capaz de mostrar o motivo de sua criação e evolução. Entender suas principais ferramentas permite que seja possível identificar quando e como utilizá-las da melhor forma possível.

Esta unidade irá apresentar a linguagem UML que é uma linguagem de modelagem visual de soluções de problemas que atinge o principal objetivo deste tipo de ferramenta: ela torna a modelagem de software uma linguagem universal e de simples entendimento.

A seção 1 apresenta o histórico da linguagem, quando foi criada, as motivações e justificativas. Mostra também como a linguagem evoluiu ao longo do tempo incorporando as necessidades dos desenvolvedores e melhorando seus pontos fracos. Em seguida, na seção 2, os diagramas da linguagem serão apresentados e divididos de acordo com sua aplicação, ou seja, diagramas estruturais (responsáveis por modelar a

estrutura do sistema como classes, atributos e operações), comportamentais (que detalham o funcionamento das partes do sistema) e de interação (que modelam as interações entre objetos de uma aplicação). Depois dessa etapa, vamos entender como é fundamentada a linguagem UML. Por fim, na seção 3, será abordada a linguagem em um processo de desenvolvimento e seus mecanismos gerais.

Ao final desta unidade, toda a base para o correto entendimento da linguagem estará formada e será possível, então, evoluir para o estudo em profundidade sobre UML.

PRATICAR PARA APRENDER

Para o desenvolvimento de software para a solução de problemas, se não há um entendimento real do problema em questão, a solução dificilmente é satisfatória. Por muitos anos, logo no início da programação de sistemas de informação para empresas, o desenvolvimento de software era feito sem seguir um padrão, sem a utilização de técnicas ou ferramentas que até então eram inexistentes. Ao longo dos anos, com uma maior exigência e necessidade de resolução de problemas mais complexos, surgiu maior demanda por sistemas mais complexos. Todavia, não existiam grandes preocupações com a manutenção e muitos códigos sequer tinham documentação porque eram produzidos informalmente. Em algumas ocasiões, a etapa de modelagem do sistema não existia, inviabilizando a manutenção nos códigos já desenvolvidos.

Neste momento, você já percebeu que em problemas complexos, sempre que utilizamos a ferramenta correta, é possível obter sucesso considerando um caminho menos árduo. Esta seção irá apresentar o histórico e as principais características de uma das ferramentas mais utilizadas para análise e modelagem de software atualmente: a linguagem UML.

Esta linguagem utiliza conceitos de modelagem visual, tendo como principal objetivo uniformizar a maneira como os sistemas, processos, projetos e negócios são modelados e descritos, solucionando os diversos problemas normalmente encontrados em empresas de desenvolvimento de software, especialmente quando há divergência entre o projeto e a entrega final e, ainda, o retrabalho que pode ocorrer por conta de implementações erradas de módulos do sistema (UML-DIAGRAMS, 2016).

Em seguida são apresentados os princípios, as características e a evolução da UML para que seja possível compreender o contexto relacionado à linguagem e as etapas de desenvolvimento para que ela atingisse a maturidade e aplicabilidade atual. Por fim, a maneira como a UML é utilizada para a modelagem de sistemas encerra esta primeira seção criando a base para o estudo aprofundado dos diagramas existentes na linguagem.

Este início é fundamental, pois irá demonstrar a importância da linguagem, seu impacto na área de desenvolvimento de software, os motivos que levaram a sua adoção em larga escala no mercado de trabalho e os fatores que resultaram em sua evolução.

EXEMPLIFICANDO

Um exemplo de impacto pode ser um software de uma loja de vendas onde o gerente deve ter acesso ao relatório de cada vendedor para tomar decisões sobre a equipe. Se este requisito não for devidamente documentado pode ser que o time de desenvolvimento, quando for programar a geração de relatórios, não implemente uma maneira de obter o relatório de vendas. Quando o software estiver terminado e for entregue ao cliente, logo na primeira reunião, o gerente irá procurar pela funcionalidade e não irá encontrá-la, causando um transtorno para o cliente. Além disso, o software terá de retornar a fase de desenvolvimento para adicionar esta funcionalidade, o que levaria tempo

dependendo de como o sistema foi construído. Todo o transtorno e o atraso neste caso poderiam ser evitados caso existisse uma documentação apropriada que descrevesse de forma clara as funcionalidades do sistema. A UML, por ser uma linguagem de modelagem visual, apresenta diagramas de fácil entendimento que de forma eficiente evitam este tipo de ocorrência.

o

Ver anotações

Você, recém-formado e ingressante no mercado de trabalho, foi contratado por uma empresa de desenvolvimento de software. A experiência de iniciar na equipe de desenvolvimento de uma empresa nova é um desafio normalmente, pois cada empresa possui seus métodos de desenvolvimento, seus padrões e todos os times de desenvolvimento estão adaptados a este universo fazendo com que a empresa consiga atingir suas metas e entregar os softwares no tempo determinado atendendo a todos os requisitos solicitados.

Apesar de todas as técnicas de desenvolvimento de software e modelos serem amplamente conhecidos, você percebeu que a empresa para a qual vai trabalhar agora ainda utiliza métodos antigos de desenvolvimento e, em alguns projetos, método algum. Agora que você iniciou seus estudos de UML, já sabe que é necessário desenvolver a modelagem dos sistemas utilizando métodos e linguagens específicas para esta finalidade. Sabe ainda que a linguagem mais utilizada é a UML por possuir todas as características principais desejáveis para o problema em questão.

Sendo assim, como seus conhecimentos sobre o histórico de UML, suas características e seus benefícios estão sólidos, você acredita poder melhorar o desenvolvimento de software no novo trabalho. Porém, esta tarefa nem sempre é fácil devido à inércia apresentada pelos funcionários que já trabalham na companhia. Você terá que, aos

poucos, mostrar os conceitos, vantagens, desvantagens, motivações e justificativas para que a empresa adote a UML e se beneficie da ferramenta em seu processo de desenvolvimento.

Logo nos primeiros dias, você percebeu que existem problemas no processo de desenvolvimento de software e a maioria deles está associada ao fato de que os programadores nem sempre entendem realmente a análise e o projeto dos softwares, o que leva a uma implementação falha. Neste caso, você pediu um espaço para apresentar os conceitos de UML e desenvolvimento de software e conseguiu uma parte de uma reunião. Lembre-se de que a implantação de uma ferramenta nova em uma empresa, independente de qual seja, irá apresentar alguns problemas que ocorrem sempre e estes problemas também são importantes e devem ser considerados.

Apresente aqui como você iria abordar, nesta primeira reunião, o processo histórico de criação da UML, suas principais características e versões e como pode ser utilizada diretamente na modelagem dos sistemas. Descreva quais características você apresentaria a seus novos superiores para convencê-los de que ao fazer esta modificação no processo de desenvolvimento, os resultados obtidos serão melhores e o tempo gasto com seu desenvolvimento poderia ser inclusive diminuído. Faça isso em uma lista que contenha problemas de desenvolvimento, vantagens e desvantagens, se achar conveniente.

Atualmente no mercado de trabalho uma quantidade considerável de empresas utiliza UML para a modelagem de seus sistemas e você, como futuro analista de sistemas, precisa ter este conhecimento para obter êxito em sua jornada profissional. Preparado para aprender sobre UML? Bons estudos!

CONCEITO-CHAVE

O desenvolvimento de software apresenta diversos desafios e um dos principais é se certificar de que todos os requisitos solicitados pelo cliente serão atendidos e da forma correta. Uma das formas de se obter sucesso com relação a este problema é realizando uma boa análise dos requisitos e a partir da análise elaborar modelos que representem o sistema em formato mais facilmente entendível por todos os membros do time de desenvolvimento. Portanto, antes de desenvolver o sistema e codificar, é importante que os membros do time tenham uma documentação clara com todos os detalhes estruturais e como o sistema ou partes do sistema deverão se comportar dependendo da ação do usuário. Vale destacar que a documentação com esses detalhes é de extrema importância para que o resultado final, o produto de software, não gere insatisfação do cliente, por não desenvolver alguma funcionalidade requerida pelo cliente, por exemplo, ou até gerar retrabalho na programação do software, atrasos no cronograma e custos adicionais que impactam no orçamento da empresa. Para evitar esses transtornos, muitas empresas adotam ferramentas – como a linguagem UML – que irão auxiliar no processo de desenvolvimento.

LINGUAGEM UML

A Linguagem de Modelagem Unificada – UML (do inglês, *Unified Modeling Language*) é uma das principais ferramentas utilizadas para modelagem de negócios e processos similares, análise, design e implementação de sistemas baseados em software. A UML possibilita realizar especificações dos artefatos a serem desenvolvidos, bem como definir as tarefas do grupo e/ou de algum membro do grupo responsável pelo desenvolvimento do sistema. Importa destacar que a UML é uma linguagem de modelagem universal, não um processo de desenvolvimento de software. A análise orientada a objetos está diretamente ligada a UML, uma vez que, na elaboração de modelos, o sistema deve ser descrito com elementos (objetos) que sugerem o que

acontece mais concretamente no mundo real. Portanto, vamos entender melhor o surgimento desta relação para, em seguida, ver os conceitos iniciais sobre a linguagem.

■ PARADIGMA DE PROGRAMAÇÃO ORIENTADA A OBJETOS

O paradigma de orientação a objetos (OO) é um paradigma de programação importante que teve seu início oficial com uma linguagem chamada SIMULA (HOLMEVICK, 1994). Esta linguagem apresentou os conceitos que em seguida viriam a ser englobados por linguagens como JAVA, C++, C#, PHP. Estes conceitos são as classes, os objetos e as relações entre eles como herança, polimorfismo, agregação e composição. Em oposição ao paradigma de programação estruturada composto por variáveis, funções e tipos abstratos de dados, a orientação a objetos propôs um modelo novo de programação que permite a modelagem dos problemas a serem solucionados pelos programadores pensando em objetos do mundo real. Este fato impacta diretamente um dos passos importantes do desenvolvimento de software: a análise.

A fase de análise de requisitos é importante para que seja possível modelar e projetar o software corretamente de acordo com o que foi especificado pelo cliente. Este processo (análise) em modelos atuais de desenvolvimento de software é classificado como modelagem e sua definição é: o processo de desenvolvimento de modelos abstratos do sistema, sendo que cada um destes modelos apresenta uma perspectiva diferente do mesmo sistema (SOMMERVILLE, 2011). Considerando esta definição, que especifica a necessidade de diversos modelos abstratos do sistema, pode-se entender a importância de uma ferramenta de modelagem para auxiliar no processo. Esta ferramenta é a UML.

Seria interessante se desde o início do desenvolvimento utilizando OO os desenvolvedores percebessem a necessidade de se modelar o sistema detalhadamente com diferentes perspectivas do mesmo

problema a ser resolvido, porém não foi assim que aconteceu o processo de desenvolvimento da linguagem UML.

LINGUAGEM ADA

Dentre as linguagens inicialmente desenvolvidas com o objetivo de permitir a programação de softwares orientados a objetos é possível destacar o caso da linguagem ADA. A linguagem ADA foi desenvolvida pelo Departamento de Defesa dos Estados Unidos e, desde o início, possuía compiladores extremamente rápidos e eficientes para as aplicações-alvo que eram militares em sua maioria (ICHBIAH *et al.*, 1996). O alto desempenho da linguagem ADA fez com que fosse amplamente adotada nos anos 1980 e um pesquisador chamado Grady Booch apresentou um dos primeiros trabalhos relacionados à análise orientada a objetos utilizando a linguagem ADA (BOOCH, 1986).

A partir destes trabalhos iniciais com foco na linguagem ADA, outros projetos também foram apresentados com métodos de análise e modelagem para linguagens orientadas a objetos. Estes métodos tinham duas características principais que os tornavam não tão atraentes de uma forma geral: ou eram orientados a determinadas linguagens, ou eram orientados a modelos de dados e estruturas de dados específicas. Um exemplo neste sentido é o método OMT criado por James Rumbaugh *et al.* (1991) que apresentava uma solução de descrição de um modelo específico de banco de dados, o modelo entidade-relacionamento.

REFLITA

Neste momento é interessante pensar um pouco sobre a situação descrita em relação às soluções apresentadas para uma linguagem ou aplicação específica. Quando um problema é comum a diversas áreas de pesquisa e desenvolvimento é sempre interessante pensar em uma solução genérica que resolva o problema do maior número possível de pessoas. Considere o grande número de linguagens de programação existentes, cada uma com o seu

escopo de aplicação específico, ou seja, umas melhores para desempenho como C++, outras melhores para interfaces com o usuário e portabilidade como JAVA, outras ainda específicas para aplicações WEB como o PHP. Todas as linguagens listadas são linguagens orientadas a objetos utilizados para diferentes aplicações e a questão é: por melhor que seja uma solução de modelagem aplicada a um tipo específico de linguagem ou problema, o quanto importante esta solução é para a área de pesquisa? Não seria melhor desenvolver uma solução genérica? Reflita, então, neste momento, sobre os benefícios de se ter uma solução de desenvolvimento aplicável a qualquer linguagem orientada a objetos e tente listar estes benefícios.

| PADRONIZAÇÃO

Um número considerável de linguagens e métodos voltados para orientação a objetos com os problemas já listados foi desenvolvido até que, em meados dos anos 1990, o grupo de padronização chamado OMG (do inglês, *Object Management Group*) (OMG, 2020) percebeu o problema e resolveu solucioná-lo. Em 1996 foi aberta uma chamada para um padrão unificado de modelagem pela OMG e isto casou com o que estava sendo feito por três pesquisadores da área, conhecidos como “*three amigos*”, Grady Booch, Ivar Jacobson e James Rumbaugh. Eles já haviam apresentado, um ano antes, ideias em relação à padronização da modelagem na conferência chamada OOPSLA (do inglês, *Object Oriented Systems Languages and Applications*). Neste momento, o objetivo deles era (BÉZIVIN, MULLER, 1999):

- Utilizar conceitos de OO para representar sistemas complexos.
- Estabelecer uma ligação entre a modelagem e a implementação.
- Considerar no modelo fatores de escala que são inerentes a sistemas complexos e críticos.

- Criar uma linguagem que poderia ser processada tanto pelos computadores como pelos programadores.

O resultado deste esforço foi a primeira versão da linguagem UML enviada como resposta à chamada do OMG em 1997. Após esta primeira versão, pessoas envolvidas em versões paralelas também enviadas ao OMG foram convidadas para participar do projeto da versão 1.1. Neste segundo momento, o principal objetivo foi a definição de um metamodelo, ou seja, a definição de quais elementos seriam disponibilizados pela linguagem UML e como utilizá-los. O padrão UML 2.0 lançado em 2005 foi desenvolvido a partir dos anos 2000 pela OMG em um processo de modernização. Apesar de todas as mudanças propostas, até hoje a linguagem possui as características iniciais de sua criação.

| CARACTERÍSTICAS DA UML

A linguagem UML apresenta algumas características que a tornam uma linguagem que cumpre de maneira satisfatória o que é esperado para modelagem do sistema de software. Sendo assim, a seguir serão apresentadas algumas importantes características da linguagem em relação a seu processo de unificação (RUBAUGH, JACOBSON, BOOCH, 2004):

- UML combina os conceitos comuns de linguagens orientadas a objetos apresentando uma definição clara para cada um, bem como notação e terminologia. Isto faz com que seja possível representar a maioria dos modelos existentes utilizando UML.
- UML é compatível com o desenvolvimento de software desde os requisitos até as etapas finais do desenvolvimento. Os mesmos conceitos e notações podem ser utilizados em diferentes estágios sem necessidade de tradução dos modelos.
- UML é compatível com diversos escopos, ou seja, é capaz de modelar diferentes linguagens, bancos de dados, documentação organizacional, trabalha com diversos frameworks, engloba inclusive o desenvolvimento de software de controle de hardware (*firmwares*).
- Um dos resultados mais importantes do desenvolvimento da UML, que não era um de seus objetivos iniciais, foi um esforço para modelar os relacionamentos entre os conceitos das linguagens orientadas a objetos tornando os diagramas aplicáveis a várias situações que eram conhecidas para os desenvolvedores e outras que ainda não eram.

| OBJETIVOS DA UML

Agora que entendemos os principais conceitos relacionados à linguagem UML é possível apresentar seus objetivos como uma ferramenta de modelagem que já foram atingidos em seu desenvolvimento.

- O primeiro, e mais importante, é ser geral no sentido de modelar diferentes linguagens e situações. A questão de ser uma ferramenta não proprietária, resultado de um acordo realizado com grande parte da comunidade de desenvolvedores, permite que seja utilizada por todos que desejarem modelar seus softwares.

- Outro importante objetivo foi a superação de outros modelos já existentes na época de seu lançamento para modelagem, permitindo que se tornasse realmente um padrão para o desenvolvimento de software.
- O último, porém igualmente importante objetivo, foi ser tão simples quanto possível sem perder a capacidade de modelagem de sistemas complexos.

| VERSÕES DA UML

Em relação às versões desenvolvidas da linguagem UML é interessante analisar sua evolução para entender como a linguagem foi desenvolvida (BOOCH, RUBAUGH, JACOBSON, 2005).

- A primeira versão, que foi enviada ao OMG em janeiro de 1997, não estava totalmente completa e precisou de reformulações. A primeira versão realmente lançada como funcional ao mercado foi a 1.1 em novembro de 1997.
- Após o retorno das informações sobre os principais problemas da linguagem, pequenas modificações foram feitas com relação a semântica, notações e metamodelos gerando a versão 1.3.
- Um contínuo processo de desenvolvimento que acrescentou aos diagramas opções de visibilidade, artefatos e estereótipos resultou na versão 1.4, que foi seguida pela versão 1.5 com a adição de procedimentos e mecanismos de *data flow*. Em janeiro de 2003, entre os lançamentos das versões 1.4 e 1.5, a UML foi aceita pela ISO como um padrão.
- Em julho de 2005 foi lançada a UML 2.0 com a inclusão de interações nos diagramas como *object*, *package* e *timing* (AMBLER, 2005). Os diagramas de colaboração foram renomeados para diagramas de comunicação, dentre outras modificações significativas em todos os outros diagramas existentes na linguagem.
- Entre as versões 2.1 a 2.3 apenas correções de erros e pequenas modificações foram realizadas.
- A versão 2.4.1 de julho de 2011 incorporou mudanças em classes, pacotes e estereótipos.
- A versão da linguagem – UML 2.5 – é uma revisão da versão anterior com melhorias na questão da simplicidade da linguagem, com a geração mais rápida de modelos mais eficientes e descartando características de versões anteriores não mais utilizadas.
- Em sua última versão, 2.5.1 de 2017, houve a correção de erros reportados pelos usuários em relação aos diagramas e sua utilização, porém, segundo o controle apresentado no site da OMG, alguns problemas ainda não foram resolvidos

A linguagem UML é toda baseada em modelos, portanto é necessário saber o que é um modelo, seus propósitos, o que compõe um modelo e, principalmente, seu significado. Um modelo pode ser definido, de forma simples e direta, como uma representação de algo de alguma natureza (software, problema, sistema matemático) utilizando algo da mesma ou de outra natureza.

Vamos explicar com mais detalhes o conceito do que é um modelo e sua natureza. Um sistema mecânico, como um motor de carro, pode ser representado por um modelo matemático de natureza diferente. Outro exemplo possível é a simulação de um sistema elétrico em um software de simulação e, neste caso, ao criar um modelo computacional de um sistema elétrico real, estamos trabalhando com objetos de natureza diferente. Um software, programa de computador, pode ser representado por um diagrama UML, que é uma ferramenta visual. Os dois objetos em questão, o software e o modelo UML do software, possuem a mesma natureza computacional, ou seja, tanto o software quanto o modelo são criados e definidos em um computador.

Um modelo captura aspectos importantes e de alguma forma modifica ou omite o restante das informações. A forma como o modelo é apresentado e desenvolvido deve ser escolhida para facilitar tanto sua construção quanto sua interpretação e utilização. O modelo de softwares e sistemas computacionais é normalmente feito em uma linguagem de modelagem e, atualmente em sua maioria, utilizando UML (GUEDES, 2018).

Os propósitos em um desenvolvimento de modelos são vários, porém é possível destacar alguns deles:

- Capturar e definir com precisão os requisitos do software para realmente atender às necessidades de quem contratou o desenvolvimento do software.
- Auxiliar o início do projeto do sistema no sentido de apresentar melhor as características que estariam condensadas em um texto.
- Apresentar uma solução que contenha as decisões de projeto em uma forma que não depende diretamente dos requisitos.

- Melhorar a exploração de diferentes soluções com uma apresentação simples de ser feita e de fácil entendimento.
 - E, principalmente, permitir que um sistema complexo seja descrito de forma que possa ser entendido em sua totalidade, possibilitando seu desenvolvimento de forma eficiente.
-

Modelos podem ser apresentados de várias formas diferentes com diversos objetivos e níveis de abstração. A quantidade de detalhes em um modelo deve ser considerada de acordo com seu objetivo de construção.

O Quadro 1.1 sintetiza os níveis de abstração e a função do diagrama para cada um desses níveis de abstração.

Quadro 1.1 | Resumo dos tipos de abstração e seus diagramas correspondentes

Nível de Abstração	Objetivo do Diagrama
ALTO	Ser claro e simples, apresentar os conceitos ao cliente para tomada de decisão
MÉDIO	Guiar o desenvolvimento apresentado, sem detalhar demais, as classes, os objetos e as interações
BAIXO	Demonstrar como deve ser desenvolvido o sistema propriamente dito. Necessita de diagramas e modelos com a especificação completa de cada módulo, interação e outras informações que possam ser necessárias

Fonte: elaborado pelo autor.

Em um nível alto de abstração, a ideia é apresentar diagramas que são construídos no início do desenvolvimento para mostrar aos clientes as possibilidades e auxiliar na tomada de decisão em pontos que precisam de aprovação. Após este nível é necessário detalhar um pouco mais com a abstração da estrutura essencial do sistema, apresentando sem muitos detalhes como serão desenvolvidos os objetos, as classes e como irão interagir. Em seguida, o nível de abstração “desce” ainda mais com o desenvolvimento de diagramas e modelos com a especificação completa de cada módulo, interação e tudo mais que seja necessário para que realmente seja possível construir o sistema propriamente dito. A única maneira de melhorar ainda mais este cenário seria apresentando exemplos possíveis do sistema com certa funcionalidade (protótipos) para que a funcionalidade seja demonstrada de forma completa (GUEDES, 2018).

Neste momento é importante entender como a utilização da linguagem UML faz diferença no processo de desenvolvimento de software. A modelagem tem um impacto muito grande no resultado final do software desenvolvido e, portanto, é necessário que seja feita de forma que todos os requisitos sejam atendidos e que todo o time de desenvolvimento do software seja capaz de visualizar os detalhes necessários para o desenvolvimento. Ao utilizar a linguagem UML e seus diagramas, o resultado da modelagem é visual e pode ser entendido sem problemas por todos os envolvidos em seu desenvolvimento. Esta característica irá resultar em um software com menor número de erros no desenvolvimento e na versão final do software, diminuição no tempo gasto com testes (justamente pelo número de erros ser menor) e com menor probabilidade de atrasos em seu tempo de desenvolvimento.

FLUXO DE DESENVOLVIMENTO

Existem diversos fluxos de desenvolvimento de software, tanto clássicos (cascata, espiral, prototipação) como os atuais (chamados de metodologias ágeis de desenvolvimento) que possuem uma etapa de modelagem prevista. A linguagem UML pode ser utilizada em qualquer um deles por ser **uma linguagem de modelagem independente do fluxo de desenvolvimento de software escolhido** (UML-DIAGRAMS, 2016).

Alguns componentes principais devem estar presentes no modelo e dentre eles é possível destacar:

- A **apresentação visual da semântica do sistema**, ou seja, uma forma de visualizar tudo que o sistema precisa ter e as relações entre as partes do sistema de forma gráfica e simples.
- E o **contexto**, ou seja, o padrão de apresentação para o público no qual o modelo foi desenvolvido.

Informações importantes para o sistema, porém, que estão fora do padrão adotado pela organização em questão, para o público-alvo esperado, devem ser armazenadas em outros locais e apresentadas de outras formas. Um exemplo de quebra de contexto é um diagrama de alto nível de abstração que contém vários trechos de código em uma linguagem específica e é apresentado aos executivos que contrataram o desenvolvimento. Se a empresa desenvolve softwares para diferentes segmentos de mercado, os clientes não irão tirar proveito algum desta informação na maioria dos casos.

Por fim é importante lembrar do real significado de um modelo, que é ser um gerador de potenciais configurações para o sistema em questão em seus diferentes níveis. O fato de se ter diversos diagramas possíveis em UML é para agrupar características diferentes nos modelos certos e só apresentar o que interessa para quem interessa. Isso inclui a escolha de modelos que apresentam o que o sistema faz em detrimento a modelos que apresentam como o sistema realiza determinadas ações,

sendo que cada um destes diagramas possui públicos específicos diferentes entre si. Por isso é importante conhecer bem a ferramenta de modelagem e utilizar todo seu potencial para obter o melhor resultado possível no momento do desenvolvimento.

Apesar de todas as facilidades da linguagem UML e do esforço de seus desenvolvedores em torná-la simples, algumas questões são utilizadas como justificativa para a não utilização da ferramenta, como a falsa impressão de que a linguagem é complexa. Isso pode se dar por, basicamente, dois motivos: pouco conhecimento sobre sua utilização e o costume de utilização de um método específico pelo desenvolvedor, que acredita ser esse método mais eficiente que a UML.

Em relação a esses problemas é importante considerar que a UML é uma ferramenta completa, amplamente utilizada e que foi desenvolvida por um grupo de pessoas que são especialistas na área de desenvolvimento para ser simples e eficiente.

Sendo assim é necessário que se aprenda a utilizar a UML e que seu uso se dê em diversos projetos antes de se decidir por outra ferramenta ou nenhuma. Na maioria dos casos, estes “problemas” acabam se mostrando como apenas uma resistência sem fundamento por parte dos desenvolvedores.

Neste ponto você já foi apresentado ao contexto da linguagem UML, sua evolução e algumas de suas principais características. Está familiarizado com diversos conceitos relacionados a modelos e abstração de dados e está preparado para iniciar seu estudo específico dos modelos presentes na linguagem UML. Bons estudos!

REFERÊNCIAS

AMBLER, S. W. **The Elements of UML 2.0 Style**. Cambridge University Press. 2005.

BÉZIVIN, J.; MULLER, P. A. UML: The Birth and Rise of a Standard Modeling Notation. The Unified Modeling Language. **UML'98: Beyond the Notation**, v.1618 of Lecture Notes in Computer Science, pp. 1-8. Springer, 1999.

BOOCH, G. Object-Oriented Development. **IEEE Transactions on Software Engineering**, n. 12, v. 2, pp. 211-221, 1986.

BOOCH, G.; RUBAUGH J.; JACOBSON, I. **The Unified Modeling Language User Guide**. 2. ed. (Addison-Wesley Object Technology Series). Addison-Wesley Professional. 2005.

GUEDES, G. T. A. **Uml 2 – Uma Abordagem Prática**. 3. ed. São Paulo: Novatec, 2018.

HOLMEVICK, J.R. Compiling SIMULA: A Historical Study of Technological Genesis. **IEEE Annals of the History of Computing**, n. 16, v. 4. pp. 25-37, 1994.

ICHBIAH, J.D. *et al.* **Rationale for the Design of the Ada Programming Language**. Cambridge: Cambridge University Press, 1986.

OMG. **OMG® Unified Modeling Language® (OMG UML®) version 2.5.1**. 2017. Disponível em: <https://bit.ly/30WhGSJ>. Acesso em: 19 maio 2020.

OMG. **Object management group page**. 2020. Disponível em: <https://bit.ly/3360Aof>. Acesso em: 19 maio 2020.

O QUE é o software livre? Free Software Foundation. [S.l. s.n.], 3 jul. 2019. 03/07/2019. Disponível em: <https://bit.ly/3k1e62u>. Acesso em: 11 maio 2020.

RUBAUGH J.; JACOBSON, I.; BOOCH, G. **The Unified Modeling Language Reference Manual**. 2 ed. Pearson Higher Education. 2004.

RUMBAUGH J. *et al.* **Object-Oriented Modeling and Design**. New Jersey: Prentice Hall, 1991.

SOMMERVILLE, I. **Engenharia de software**. 9. ed. São Paulo: Pearson Universidades, 2011.

UML-DIAGRAMS. **The Unified Modeling Language**. 2016. Disponível em: <https://bit.ly/3f7qM41>. Acesso em: 19 maio 2020.

29/04/2019. Disponível em: <https://bit.ly/309nGbt>. Acesso em: 1 jul.
2020.

FOCO NO MERCADO DE TRABALHO

FUNDAMENTOS DA UML

Maurício Accocia Dias

Ver anotações

UTILIZAÇÃO DA LINGUAGEM UML

A UML é uma das principais ferramentas de modelagem utilizadas em empresas de desenvolvimento de software e seu conhecimento é essencial para um desenvolvedor no mercado de trabalho.



Fonte: Shutterstock.

Deseja ouvir este material?

Áudio disponível no material digital.

SEM MEDO DE ERRAR

A solução da situação-problema está pautada na apresentação das principais vantagens e desvantagens de se utilizar a UML na empresa. Uma maneira de atingir seu objetivo neste caso é não apenas apontar as soluções para os problemas, mas indicar os problemas que ocorrem no processo de desenvolvimento de software de uma empresa em decorrência da não utilização de uma correta ferramenta de modelagem.

• Ver anotações

VANTAGENS

RESOLUÇÃO DE PROBLEMAS

Dentre os problemas apontados no desenvolvimento de software que podem ser resolvidos com a UML é possível salientar:

- Ao iniciar o desenvolvimento, muitas empresas, principalmente as startups e empresas menores, ignoram os passos para um bom processo de desenvolvimento por acreditar que o tempo gasto na utilização de métodos e técnicas poderia ser empregado no próprio desenvolvimento do software.

- Outra questão importante neste caso é que quanto maior o número de erros no desenvolvimento e nas funcionalidades implementadas de forma errada, maior o retrabalho nas fases de teste e validação do sistema.
- Uma empresa de desenvolvimento possui um ou mais grupos de desenvolvedores. Ao se modelar o sistema de forma precária, sem um bom método definido, é possível que diversos problemas causem mais danos que deveriam no desenvolvimento. Por exemplo, caso um funcionário da empresa saia ou seja necessário realocar uma parte do time de desenvolvimento para outro projeto, os novos integrantes da equipe podem demorar muito tempo até se adaptar ao desenvolvimento do software e entender como devem proceder.

Estes problemas são comuns a empresas de desenvolvimento de software, mas podem ser resolvidos adotando-se práticas e ferramentas no processo.

I BENEFÍCIOS

A UML é uma ferramenta que quando utilizada apresenta muitos benefícios, destacando-se:

- Para a empresa é importante dizer que a utilização da UML não envolve custos já que o padrão é aberto. Além disso, por ser amplamente utilizada existem ferramentas de código aberto disponíveis para auxiliar na criação dos diagramas.
- O tempo de teste e validação do software será reduzido já que o número de erros tende a diminuir. Esta diminuição ocorre pois os desenvolvedores envolvidos no projeto irão entender melhor o software em questão e a relação entre os módulos do sistema com os diagramas disponíveis.
- A questão sobre entender melhor o software que está sendo desenvolvido também é importante para a satisfação do cliente já

que o produto final tende a atender melhor as especificações que ficam mais claras quando modeladas em diagramas.

- O fato de ser uma ferramenta aberta e muito utilizada também apresenta a vantagem de existir muito material disponível e aberto para estudo sobre o tema.

| DESVANTAGENS

Ao apresentar uma nova ferramenta que se pretende adotar em uma empresa é necessário discutir também as desvantagens. A adoção da UML traz benefícios a médio e longo prazo, porém em curto prazo apresenta alguns possíveis problemas:

- Apesar de ser uma linguagem simples, a criação de diagramas melhores e mais detalhados aparece com a experiência na utilização da linguagem, portanto a melhoria total da utilização da UML vai demorar um tempo para aparecer.

- Será necessário promover um treinamento da equipe de desenvolvimento para o uso correto da linguagem e dos softwares escolhidos para a sua implantação.
- Também é possível que não se tenha a aderência desejada por parte da equipe como um todo e seja necessário trocar membros dos grupos de desenvolvimento para que o processo de implantação como um todo não seja prejudicado.

Com essas informações será possível analisar o caso e fazer uma decisão consciente sobre sua adoção.

NÃO PODE FALTAR

TÉCNICAS DE MODELAGEM DA UML

Maurício Accocia Dias

Ver anotações

DIAGRAMAS UML

A UML é uma linguagem de modelagem visual que apresenta diferentes tipos de diagramas para diferentes tipos de situações e etapas do desenvolvimento.



Fonte: Shutterstock.

Deseja ouvir este material?

Áudio disponível no material digital.

Olá, aluno! Bem-vindo ao início do estudo sobre os diagramas UML. A modelagem é uma fase importante do desenvolvimento de software e deve ser realizada da maneira mais clara, simples e funcional possível. A UML proporciona esses resultados por ser uma linguagem de modelagem visual que apresenta diferentes tipos de diagramas para diferentes tipos de situações e etapas do desenvolvimento. Como o mercado de trabalho solicitará de você, futuro desenvolvedor, um conhecimento sólido sobre esses conceitos, é importante criar uma boa base de conhecimento, para que você seja capaz de resolver, da melhor maneira possível, os problemas que serão enfrentados na profissão.

Recém-formado e ingressante no mercado de trabalho, você foi contratado por uma empresa de desenvolvimento de software. A experiência de iniciar na equipe de desenvolvimento de uma empresa nova já é um desafio, pois cada empresa tem seus métodos de desenvolvimento e seus padrões, e todos os times de desenvolvimento estão adaptados a esse universo, fazendo com que a empresa consiga atingir suas metas e entregar os softwares no tempo determinado, atendendo a todos os requisitos solicitados.

Apesar de todas as técnicas de desenvolvimento de software e modelos serem amplamente conhecidos, você percebeu que sua nova empresa ainda utiliza métodos antigos de desenvolvimento e, em alguns projetos, método nenhum, ou seja, os softwares não apresentam documentação, sendo difícil para um membro novo na equipe trabalhar no código. Você, um profissional recém-contratado, percebe a dificuldade para o entendimento do software desenvolvido e entende ser necessário realizar a modelagem do sistema. Além disso, conhece a importância da documentação, utilizando métodos e linguagens específicas para essa finalidade. Sabe, ainda, que a linguagem mais utilizada é a UML, pois apresenta todas as principais características desejáveis para o problema em questão.

Sendo assim, como seus conhecimentos sobre o histórico de UML, características e benefícios da linguagem estão sólidos, você acredita poder melhorar o processo de desenvolvimento e manutenção de software em seu novo emprego. Todavia, você terá que mostrar a importância da documentação e modelagem de sistemas para si e demais membros do time de desenvolvimento, mencionando a importância considerando o reuso. Você terá que, aos poucos, mostrar os conceitos, vantagens, desvantagens, motivações e justificativas para que sua empresa adote o UML e se beneficie da ferramenta em seu processo de desenvolvimento.

Você já apresentou a linguagem UML para os diretores da sua empresa, que ficaram interessados em conhecer mais acerca do tema. Então, continuando com seu objetivo de melhorar o processo de desenvolvimento da empresa de software que o contratou, você ficou entusiasmado após saber que sua primeira reunião surtiu efeito, e os funcionários querem aprender mais a respeito da linguagem UML. Agora você deve apresentar um pouco mais da teoria da linguagem UML para seus colegas e para os diretores da empresa.

Faça, portanto, um roteiro de apresentação dos diagramas UML contendo, em resumo, as informações sobre a classificação dos diagramas e a definição básica de cada um deles. Lembre-se de que é interessante relacionar exemplos de situações reais a cada um dos diagramas sempre que possível.

Vamos, então, começar a construir uma base sólida de conhecimentos sobre UML e seus diagramas? Bons estudos!

CONCEITO-CHAVE

A linguagem UML é uma importante ferramenta para a modelagem de sistemas, a qual possibilita elaborar modelos abstratos, tendo um visual do sistema e de como os objetos se comunicam, tudo mostrado na forma de diagramas. Cada modelo elaborado representa um aspecto do sistema, ou seja, suas diferentes perspectivas. Esse tipo de ferramenta,

quando utilizada durante as etapas de desenvolvimento do software, proporciona uma série de benefícios e melhorias a esse processo, mesmo sendo uma linguagem independente de processos e não uma metodologia de desenvolvimento.

o

Ver anotações

O fato de a linguagem ser tão bem-estruturada e não apresentar custos de utilização faz com que ela seja amplamente utilizada no mercado de trabalho e torna seu conhecimento indispensável para a análise e modelagem de sistemas.

A ferramenta dispõe de uma ampla lista de funcionalidades. No caso da UML, essas funcionalidades são os diagramas, em 14 tipos diferentes. Seria possível começar a abordá-los um a um sem qualquer tipo de diferenciação básica entre eles, porém é importante que se entenda o objetivo principal de sua utilização. Para que torne mais simples encontrar o diagrama necessário para um determinado problema, a linguagem apresenta uma classificação de tipos de diagramas.

| DIAGRAMAS

Durante a criação de diagramas de modelagem constatou-se que duas questões eram principais em seu desenvolvimento: a estrutura e o comportamento do que se deseja modelar. Então os diagramas UML foram divididos em dois grandes grupos: **os diagramas UML estruturais e os diagramas UML comportamentais**. Há, ainda, os diagramas de integração, que basicamente fazem parte do grupo de diagramas comportamentais. A partir desses diferentes grupos de diagramas podemos ter a visão do sistema em diferentes perspectivas. Essas divisões ficarão mais claras ao longo do estudo individual dos diagramas; por ora é importante saber como podem ser classificados.

A Figura 1.1 apresenta os 14 diferentes diagramas UML como proposto em UML- Diagrams (2016).

Figura 1.1 | Classificação de diagramas UML com relação a sua natureza

Para visualizar o vídeo, acesse seu material digital.

Fonte: elaborada pelo autor.

Embora não seja indicada pelos manuais da linguagem UML a ordem em que devem ser criados e utilizados os diagramas em um determinado fluxo de desenvolvimento de software, normalmente se aborda os diagramas em uma ordem mais didática quando são explicados. Essa ordem é baseada na complexidade dos diagramas, e usualmente são apresentados os diagramas mais simples primeiro para que o processo de criação seja assimilado de forma mais eficaz.

Seguindo essa ideia, começaremos de forma simplificada com os diagramas estruturais, em seguida iremos para os diagramas comportamentais, que têm uma subdivisão para os diagramas de interação, a qual será também abordada. A abordagem dos diagramas neste momento será superficial, para proporcionar um entendimento da estrutura de forma geral. Ao longo do estudo da linguagem UML cada diagrama será detalhado, de forma que sua estrutura e utilização ficarão mais claras.

■ DIAGRAMAS ESTRUTURAIS

Os diagramas estruturais apresentam como um determinado sistema é organizado em partes (suas estruturas), seus componentes e os relacionamentos entre esses componentes. Os diagramas estruturais muitas vezes estão associados à modelagem estática, pois mostram a estrutura do sistema. Em geral, os diagramas estruturais são elaborados no momento do projeto da arquitetura do sistema. Eles representam os conceitos significativos do sistema como abstrações, questões de implementação e do mundo real.

Os diagramas estruturais, como apresentado na Figura 1.1 são sete, descritos a seguir, iniciando com o diagrama que provavelmente é um dos mais utilizados até por desenvolvedores que não estão totalmente familiarizados com os conceitos da UML: o **diagrama de classes**.

■ DIAGRAMA DE CLASSES

Como um dos fundamentos da criação da linguagem UML é o paradigma de programação de Orientação a Objetos, o diagrama de classes remete às classes criadas em um software desenvolvido em uma linguagem orientada a objetos.

O objetivo do diagrama de classes é representar as classes, suas definições e as relações entre elas.

Segundo a definição do paradigma de orientação a objetos, uma **classe** é uma abstração que descreve entidades do mundo real e quando instanciadas dão origem a objetos com características similares. A **abstração classe** é composta por atributos e métodos.

- Os **atributos** são como as variáveis ligadas ao conceito apresentado, por exemplo: considerando uma classe pessoa, tudo que uma pessoa “possui” (nome, endereço, profissão, sexo, naturalidade) são seus atributos.
- Os **métodos** são como as funções, por exemplo: tudo que uma pessoa “pode fazer” (andar, comer, falar, trabalhar).

Além de suas características, as classes também se relacionam de formas variadas. Algumas dessas relações são a herança e o polimorfismo conforme exemplos a seguir:

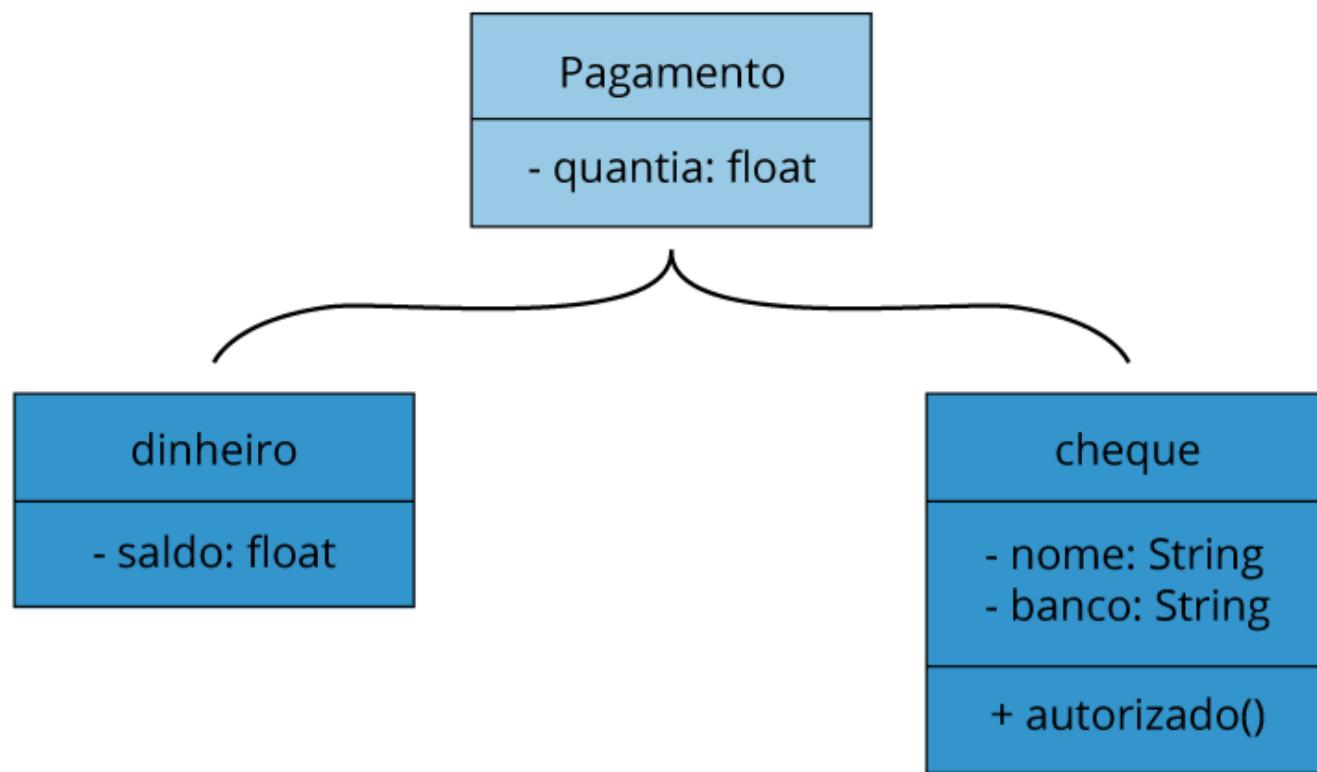
- **Herança:** quando vamos modelar veículos como classes, algumas características são comuns e outras são específicas. Um carro de passeio não tem uma caçamba como um veículo utilitário. Sendo assim, as características comuns são colocadas em uma “superclasse”, e as específicas são colocadas em uma “subclasse” que herda as características da superclasse. Então, nesse caso, pode haver uma superclasse veículo que apresenta rodas, portas e motor como atributos, e acelerar, frear e virar como métodos. Também poderia haver a subclasse utilitário, que herda todas essas características da superclasse veículo, incluindo a caçamba nos atributos e o método carregar, por exemplo.
- **Polimorfismo:** imagine que há uma classe que modela um grupo de animais. Todos os animais se locomovem, porém uns andam, outros rastejam e outros ainda voam. O polimorfismo trata esses casos, por exemplo, com a implementação do método andar na superclasse animal e a redefinição desse método para cada subclasse, de acordo com a característica específica do animal a ser modelado.

Os atributos e métodos que definem as classes são também representados no diagrama de classes da UML. Relações representadas no diagrama de classes apresentam como as classes se comunicam e se relacionam.

EXEMPLIFICANDO

Vejamos um exemplo simples do diagrama de classes:

Figura 1.2 | Exemplo de diagrama de classes



Fonte: elaborada pelo autor.

Nesse exemplo, é possível visualizar a classe **Pagamento** definida como superclasse das classes **dinheiro** e **cheque**. A seta saindo das classes **dinheiro** e **cheque** e chegando na classe **Pagamento** indica herança. No caso da classe **dinheiro** há apenas um atributo **saldo**; já na classe **cheque** temos o atributo **nome**, outro atributo **banco** e o método **autorizado**. Esse é um exemplo simples que ilustra o diagrama de classes e facilita a familiarização com o conceito.

■ DIAGRAMA DE PACOTES

O diagrama de pacotes representará os subsistemas contidos no software desenvolvido e as grandes áreas de cada um desses sistemas. Nesse diagrama é possível separar e indicar interfaces de usuário, bancos de dados, módulos de segurança do sistema e pacotes administrativos.

■ DIAGRAMA DE COMPONENTES

O diagrama de componentes, como diz o nome, modela estruturalmente a relação dos componentes utilizados no software.

Um componente é uma parte do seu software que não foi desenvolvida por você, que já vem pronta para utilização e realiza uma função específica.

Quando vamos utilizar uma função matemática em linguagem C e incluímos a biblioteca *math.h*, por exemplo, ela pode ser classificada como um componente. O diagrama de componentes é importante, pois demonstra em quais partes da arquitetura do sistema foram utilizados componentes, e isso torna mais claro o motivo das decisões (RUBAUGH; JACOBSON; BOOCHE, 2004).

■ DIAGRAMA DE PERFIL

A UML define o modelo de perfis como “leve e de extensão”. Um **perfil** é uma redefinição de uma classe para um determinado domínio ou plataforma. Por exemplo: um perfil determinado de usuário pode ser diferente no caso de um servidor ou de um dispositivo móvel. O diagrama de perfis é utilizado para apresentar os diferentes perfis necessários no desenvolvimento de um software. Antes de sua inclusão na UML, os perfis eram descritos utilizando outros diagramas.

■ DIAGRAMA DE INSTALAÇÃO

O diagrama de instalação descreve a estrutura de hardware e software necessária para a correta execução do software em desenvolvimento. No diagrama é possível relacionar quais componentes do software

desenvolvido serão executados em cada um dos nós de hardware descritos.

■ DIAGRAMA DE OBJETOS

O diagrama de objetos demonstra os objetos e os seus relacionamentos em tempo de execução. Imagine um software com um menu de seis opções, das quais você sempre utiliza somente cinco. Na prática, a relação expressa pela sexta opção não impacta o sistema, pois você não a usa. Esse diagrama tem a função de expressar essas relações justamente porque se existe algo modelado que nunca é usado, esse algo precisaria estar na modelagem?

A ideia é que o diagrama de objetos auxilie na análise de multiplicidades de objetos e relações na prática.

Repare que ele é classificado como estático, mas pode necessitar de atualizações, pois à medida que a implementação evolui os objetos e suas relações mudam.

| DIAGRAMA DE ESTRUTURA COMPOSTA

A mesma análise pode ser feita com relação aos componentes utilizando o diagrama de estrutura composta, que apresenta relações entre os objetos e componentes do software desenvolvido em tempo de execução. A diferença entre esses diagramas é que o de objetos apresenta apenas os objetos, e o de estrutura composta apresenta os componentes.

Agora que já conhecemos os diagramas estruturais, podemos avançar para os diagramas comportamentais. Esses diagramas são diferentes dos que foram apresentados até este momento, por focarem o comportamento entre os componentes do sistema e não mais em como são organizados.

| DIAGRAMAS COMPORTAMENTAIS

Os diagramas comportamentais têm como objetivo mostrar o fluxo de informações e os eventos do sistema longo do tempo. Em outras palavras: apresenta a resposta do sistema a algum evento do seu ambiente, mostrando, assim, o comportamento dinâmico dos objetos em um sistema e como o sistema reage a determinadas ações/eventos.

Os diagramas de interação fazem parte do grupo de diagramas comportamentais, e a partir dele pode-se modelar as interações do sistema ou a interação entre os componentes de um sistema.

| DIAGRAMA DE CASOS DE USO

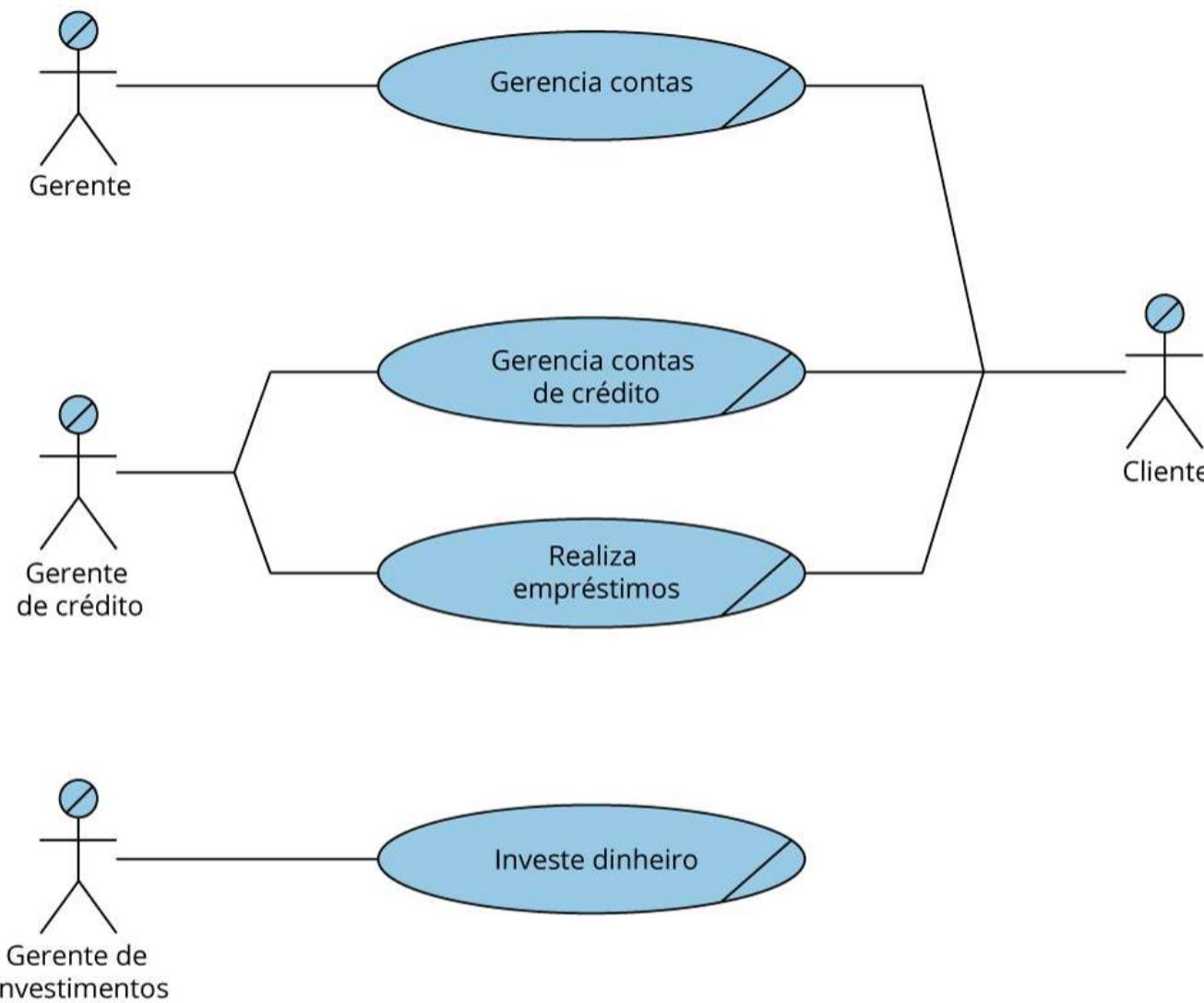
O primeiro diagrama a ser tratado será o diagrama de casos de uso. Ele é um dos primeiros a ser desenvolvido na modelagem de um sistema, por apresentar uma visão geral das funcionalidades do sistema da perspectiva dos usuários. Seria, em outras palavras, a apresentação da forma como o usuário pode usar o sistema que será desenvolvido.

Esse diagrama é muito simples de interpretar e tem grande importância nas fases iniciais do desenvolvimento do software, já que proporciona ao cliente uma visão completa do que será

Nele, caso alguma funcionalidade estiver faltando ou tenha sido descrita de forma errada, a falha será facilmente identificada e corrigida. A correção na fase inicial do processo de desenvolvimento será de grande auxílio para a diminuição de erros.

Vejamos um exemplo simples do diagrama de casos de uso:

Figura 1.3 | Exemplo de diagrama de casos de uso



Fonte: elaborada pelo autor.

Nesse diagrama de casos de uso temos as relações que acontecem em um banco com três tipos de gerente e um cliente. Os “bonequinhos” são o que chamamos de atores (por realizarem as ações), os balões são os casos de uso e as setas ou traços representam as comunicações presentes na relação. Portanto, o cliente pode gerenciar as contas, as contas de crédito e realizar empréstimos. O gerente gerencia contas, o gerente de crédito gerencia contas de crédito e realiza empréstimos. Somente o gerente de investimentos pode investir o dinheiro.

| DIAGRAMA DE ATIVIDADES

O diagrama de atividades é importante por complementar o diagrama de casos de uso, apresentando os fluxos que ocorrem no sistema como um todo. Para cada possibilidade criada no diagrama de casos de uso (usuário pode abrir um arquivo novo e iniciar sua edição, por exemplo), o fluxo da interação será descrito no diagrama de atividades. O

interessante é que não só os fluxos normais são apresentados, mas também os alternativos e as exceções, o que torna o entendimento do sistema como um todo mais completo.

■ DIAGRAMA DE VISÃO GERAL DE INTERAÇÃO

O diagrama de visão geral de interação apresenta todas as relações que ocorrerão no sistema em alto nível. A importância desse diagrama é que ele mostra como os diferentes diagramas UML se relacionarão e as dependências entre eles. Ele é o último tipo de diagrama comportamental estático.

■ DIAGRAMA DE TRANSIÇÃO DE ESTADOS

O diagrama de transição de estados demonstrará, essencialmente, o ciclo de vida de determinado objeto em tempo de execução. Esse ciclo de vida mostra todos os estados possíveis para o objeto quando o programa está em execução, e as condições para a mudança entre esses estados. Por exemplo, após fazer o *login* em um sistema, o usuário normalmente tem um conjunto de opções e, para cada escolha, uma sequência de ações pode ser realizada. O mapeamento de todas as opções para todas as escolhas vai gerar o diagrama de transição de estados.

■ DIAGRAMAS DE SEQUÊNCIA E DE COLABORAÇÃO

A modelagem das interações entre objetos ao longo do tempo é apresentada pelo **diagrama de sequência**. Nesse caso, são representadas as possíveis sequências de trocas de mensagens e informações realizadas entre os objetos ao longo da execução do software. Os **diagramas de colaboração (ou comunicação)** têm o mesmo objetivo do diagrama de sequência, porém a disposição da informação e a maneira como a informação é apresentada são diferentes. O foco do diagrama de sequência é a relação temporal, ou seja, a ordem em que as mensagens são trocadas, enquanto o foco do diagrama de colaboração está na estrutura da relação dos objetos, sem relação com informações temporais.

■ DIAGRAMA DE TEMPO

Encerrando a análise inicial dos diagramas UML falaremos do diagrama de tempo. Esse diagrama tem por objetivo apresentar a execução do sistema como um todo em períodos de tempo. A importância dessa representação é a possibilidade de ver todos os objetos ativos em um determinado instante de tempo e as relações que podem estar ocorrendo entre eles. Para encontrar a solução de problemas de implementação, esse diagrama é de grande importância.

No início da apresentação dos diagramas foram citadas duas categorias para os diagramas: estrutural e comportamental. É possível, após a análise dos sete diagramas estruturais, entender que os diagramas que apresentam alguma estrutura do software são classificados como estruturais. Os outros oito diagramas são focados no comportamento das estruturas do sistema e não em como são organizados.

Também fica mais simples compreender que um diagrama será estático se não apresentar informações que sofram mudanças ao longo do tempo de desenvolvimento do software, como o comportamento em tempo de execução. Já os diagramas dinâmicos apresentam informações sobre o comportamento em tempo de execução e, portanto, mudam à medida que o software evolui e sofre modificações.

REFLITA

Após uma visão geral dos diagramas UML é possível analisar uma questão que fica evidente em sua utilização. Seria realmente necessário construir todos os diagramas para todos os sistemas? A resposta para essa pergunta é complexa e está relacionada com a complexidade do sistema a ser desenvolvido e com o número de pessoas envolvidas no desenvolvimento. Tente refletir sobre essa questão, considerando sistemas de tamanhos diferentes sendo desenvolvidos em empresas de tamanhos diversos.

Conhecer a linguagem UML como um todo é interessante, pois nos permite tentar encontrar situações em que cada um dos diagramas é aplicável, e quais problemas podem solucionar. Portanto, continue seus estudos e descubra de que maneira construir cada um dos diagramas e como utilizá-los no desenvolvimento de software para obter o melhor resultado possível. Bom trabalho!

REFERÊNCIAS

COSTA, A. N.; WERNECK, V. M. B.; CAMPOS, M. F. Avaliação de Ferramentas para Desenvolvimento Orientado a Objetos com UML.

Cadernos do IME. Série Informática. V. 25. 2008. Disponível em: <https://bit.ly/313DUIs>. Acesso em: 1 jul. 2020.

OMG – OBJECT MANAGEMENT GROUP. Página inicial. 2020. Disponível em: <https://bit.ly/3360Aof>. Acesso em: 19 maio 2020.

RUBAUGH, J.; JACOBSON, I.; BOOCHE, G. **The Unified Modeling Language Reference Manual.** 2. ed. Pearson Higher Education, 2004.

UML – DIAGRAMS. **The Unified Modeling Language.** 2016. Disponível em: <https://bit.ly/3f7qM41>. Acesso em: 19 maio 2020.

FOCO NO MERCADO DE TRABALHO

TÉCNICAS DE MODELAGEM DA UML

Maurício Acconcia Dias

Ver anotações 0

CLASSIFICAÇÃO DOS DIAGRAMAS

Os diagramas mostram o sistema de software em diferentes perspectivas, apresentando aspectos estruturais e aspectos comportamentais.



Fonte: Shutterstock.

Deseja ouvir este material?

Áudio disponível no material digital.

Para a sua apresentação é possível construir uma tabela que resume o conteúdo trabalhado na seção. É importante mencionar a importância da UML para que novos membros do time possam compreender o software na totalidade.

A utilização da ferramenta possibilita o entendimento de partes e do todo sistema de software. Lembre-se de que a UML tem diversos diagramas que mostram o sistema de software em diferentes perspectivas, apresentando aspectos estruturais e aspectos comportamentais.

Além disso, posteriormente, caso o seu cliente solicite a adição de alguma funcionalidade ao sistema de software, os diagramas da UML podem auxiliar no entendimento de partes e do sistema como um todo. Ao apresentar o conteúdo, siga o que está descrito no quadro, falando a respeito de cada um dos tópicos.

Quadro 1.2 | Classificação e definição básica dos tópicos de diagrama UML

Diagrama UML	Classificação		Definição Básica
Classe	Estrutural	Estático	Apresenta as classes, suas definições e relações (interações, colaborações).
Pacote	Estrutural	Estático	Apresenta o sistema em pacotes de acordo com sua funcionalidade. Divide rotinas de banco de dados, segurança, interfaces.
Componente	Estrutural	Estático	Apresenta todos os componentes que serão utilizados no sistema, como bibliotecas, softwares de terceiros e web <i>services</i> , se for o caso.
Perfil	Estrutural	Estático	Apresenta os possíveis perfis de implementação para cada objeto em diferentes situações

Diagrama UML	Classificação		Definição Básica
Instalação	Estrutural	Estático	<p>Apresenta o hardware e os softwares necessários para implantação do sistema.</p>
Objetos	Estrutural	Estático	<p>Apresenta os objetos e suas relações em tempo de execução.</p>
Estrutura composta	Estrutural	Estático	<p>Apresenta os componentes e suas relações com objetos em tempo de execução.</p>
Casos de uso	Comportamental	Dinâmico	<p>Visão geral das funcionalidades do sistema. Como o usuário "usa" o sistema.</p>
Atividade	Comportamental	Dinâmico	<p>Apresenta todos os fluxos que existem no sistema. Completa os casos de uso.</p>

Diagrama UML	Classificação		Definição Básica
Visão geral de interação	Comportamental	Dinâmico	<p>Apresenta uma visão geral, em alto nível, das interações do sistema, permitindo a visualização das relações entre os diagramas UML.</p>
Transição de estados	Comportamental	Dinâmico	<p>Apresenta o ciclo de vida dos objetos, ou seja, todos os possíveis estados de um objeto em tempo de execução.</p>
Sequência	Comportamental	Dinâmico	<p>Modela como as relações entre objetos ocorrem ao longo do tempo.</p>
Colaboração	Comportamental	Dinâmico	<p>Demonstra como os objetos interagem em um determinado momento em tempo de execução.</p>

Diagrama UML	Classificação		Definição Básica
Tempo	Comportamental	Dinâmico	<p>Mostra o estado geral do sistema em um determinado instante, permitindo visualizar todos os objetos ativos e suas relações.</p>

o

Ver anotações

Fonte: elaborado pelo autor.

NÃO PODE FALTAR

O PROCESSO DE DESENVOLVIMENTO DE SOFTWARE COM UML

Maurício Acconcia Dias

PROCESSO UNIFICADO NO DESENVOLVIMENTO DE SOFTWARE

É preciso utilizar a UML em conjunto com um processo de desenvolvimento de software, como o UP, um método interativo e incremental que apresenta quatro fases: concepção, elaboração, construção e transição, divididas em fluxos de trabalhos.



Fonte: Shutterstock.

[Deseja ouvir este material?](#)

PRATICAR PARA APRENDER

Agora que você já tem os conhecimentos básicos da linguagem UML, é necessário entender a relação da linguagem com os processos de desenvolvimento. A linguagem auxilia no processo de desenvolvimento de software, por isso é utilizada nesse processo. Nesta seção, você vai aprender a respeito do processo de desenvolvimento de software chamado UP (do inglês *Unified Process*). Esse processo de desenvolvimento, desenvolvido por Ivar Jacobson, um dos criadores da UML, foi escolhido por apresentar características que facilitam a utilização da UML como linguagem de modelagem.

Outro ponto importante que abordaremos nesta seção é a relação de consistência entre os diagramas UML gerados para o desenvolvimento de software. Você deve se lembrar de que esses diagramas proporcionam diferentes visões do mesmo software, porém, essas visões devem ser consistentes, ou seja, devem apresentar uma representação única de onde e como se deseja chegar no processo de elaboração de um sistema de informação. Assim, imagine que haja dois diagramas UML distintos e com informações/requisitos diferentes, sendo utilizados por diferentes membros de um time de desenvolvimento do produto de software. Possivelmente, o produto final apresentará erros ou a comunicação entre módulos do software pode ser impactada, devido à inconsistência de informações dos diagramas.

Por fim, serão abordados os mecanismos gerais da notação UML. Cada diagrama tem suas particularidades, porém existem algumas ferramentas que são aplicáveis a todos eles e que podem melhorar sua legibilidade e enriquecer a informação apresentada. Dessa forma, conhecer essas ferramentas é importante para um programador que pretende utilizar a UML em seu dia a dia profissional.

Analizando o que foi apresentado é possível perceber que esta seção situará você, caro aluno, em relação à utilização da ferramenta que está aprendendo: a linguagem UML. Feito isso, será ainda mais fácil entender o motivo da linguagem ser tão importante e como utilizá-la de forma a obter o melhor resultado de sua aplicação. Cada uma das fases do UP será relacionada a um diagrama, tornando o entendimento ainda mais completo.

Com o objetivo de colocar os conhecimentos a serem aprendidos em prática, vamos analisar a seguinte situação-problema: você foi contratado por uma startup de desenvolvimento de sistemas, e agora fará parte de um time de desenvolvimento. É importante saber que cada empresa tem seus métodos de desenvolvimento e seus padrões e que todos os times de desenvolvimento estão adaptados a esse universo, para atingir suas metas e entregar os softwares no tempo determinado, atendendo a todos os requisitos solicitados pelos clientes.

Apesar de todas as técnicas de desenvolvimento de software e modelos serem amplamente conhecidos, você percebeu que a startup ainda utiliza métodos clássicos de desenvolvimento e, em alguns projetos, ainda não existe um processo de desenvolvimento com uma documentação clara do que se deseja nos produtos de software. Agora que você iniciou seus estudos de UML, sabe que é necessário desenvolver a modelagem dos sistemas utilizando métodos e linguagens específicas para essa finalidade. Sabe, ainda, que a linguagem mais utilizada é a UML, por apresentar todas as características principais desejáveis para o problema em questão.

Sendo assim, como seus conhecimentos acerca do histórico de UML, suas características e seus benefícios estão sólidos, você acredita poder melhorar o desenvolvimento de software na empresa em que trabalha. Assim, você terá o desafio de mostrar os conceitos, vantagens, desvantagens, motivações e justificativas para que sua empresa adote o UML e se beneficie da ferramenta em seu processo de desenvolvimento.

Ao iniciar seus dias de trabalho nessa empresa você percebeu que existem problemas no processo de desenvolvimento de software, e a maioria deles estão associados ao fato de que os programadores nem sempre entendem realmente a importância e a análise e modelagem de um sistema antes do desenvolvimento, muitas vezes gerando falhas na implementação.

Após algumas apresentações feitas, você já ganhou destaque entre os diretores, que agora gostariam de saber como implantar a UML no processo de desenvolvimento. Você foi informado que deverá seguir o processo unificado dividido em quatro etapas. O líder do projeto solicitou um relatório descrevendo a melhor maneira de incorporar a utilização da linguagem UML no processo de desenvolvimento. Seu trabalho será fazer um relatório de no máximo duas páginas relacionando o UP à linguagem UML. Lembre-se de que eles já conhecem o UP, mas não identificaram os diagramas que podem ser utilizados no processo unificado. Você acha que seria interessante utilizar recursos gráficos para a apresentação dos conceitos? Bom trabalho!

Preparado para conhecer melhor como aplicar a UML em um processo de desenvolvimento de software e utilizá-lo futuramente no desenvolvimento de um software? Então, bons estudos!

CONCEITO-CHAVE

Você agora já está entendendo melhor o poder da linguagem UML e como ela pode ser útil para o desenvolvimento de um sistema. Além disso, você já compreendeu os tipos de diagramas e o objetivo de cada um deles. Apesar de todas as vantagens que conhecemos da linguagem UML, é necessário compreender como utilizar essa linguagem de modo a obter os resultados esperados. Vamos considerar três pontos e, a partir deles, desenvolver uma análise a respeito da utilização da linguagem. Os três pontos são:

1. A linguagem UML é uma ferramenta que pode ser aplicada para a modelagem do sistema em processos de desenvolvimento de software.

2. A aplicação da linguagem de forma incorreta pode prejudicar o desenvolvimento de software, sendo necessário saber como fazê-lo.

3. A existência de mecanismos comuns a todos os diagramas da linguagem, que devem ser utilizados para que esses diagramas fiquem mais completos e explicativos.

Para entender como a linguagem UML pode ser aplicada ao UP, é necessário entender seu funcionamento. Definimos inicialmente a linguagem UML como uma linguagem visual para criação de modelos.

Um modelo de um software consiste em uma descrição completa desse software a partir de uma determinada perspectiva.

Embora a UML tenha 14 diferentes perspectivas para o software, a UML não define um processo de software padrão, logo, tais modelos em si não são suficientes para suprir as necessidades do desenvolvimento como um todo, e é preciso utilizar a UML em conjunto com um processo de desenvolvimento de software.

PROCESSO UNIFICADO

Basicamente, processos descrevem **quem** é o responsável por fazer um determinado artefato (**o que**), **como** serão executadas as tarefas e **quando**. Uma forma de escrever tais processos é por meio do chamado UP (do inglês *Unified Process*) ou Processo Unificado (PU), criado pelos fundadores da UML (JACOBSON, I., BOOCH, G., RUMBAUGH, 1999). Uma evolução do UP é o RUP (do inglês *Rational Unified Process*). O RUP é um refinamento do PU desenvolvido pela *Rational Corporation* (daí a origem do nome), posteriormente adquirida pela empresa IBM. O processo foi melhorado e foram criadas ferramentas para auxiliá-lo.

O UP consiste em um processo de desenvolvimento de software interativo e incremental em que, a partir de um conjunto de atividades bem-definidas, os requisitos de clientes (usuários) são convertidos em um sistema de software. Dessa forma, o desenvolvimento do software no UP foi definido com base nas características a seguir (WAZLAWICK, 2014):

- O UP deve ser interativo e incremental, ou seja, a cada nova iteração são introduzidos incrementos de novas características à arquitetura do sistema. Pode-se também corrigir e refinar partes do projeto.
- O UP é dirigido por uma lista de casos de uso.
- O UP deve ser focado na arquitetura do sistema, a qual possibilita implementar os requisitos.
- O UP é orientado a riscos, e as partes do projeto com maior risco são priorizadas e tratadas primeiramente. Como exemplo, podemos

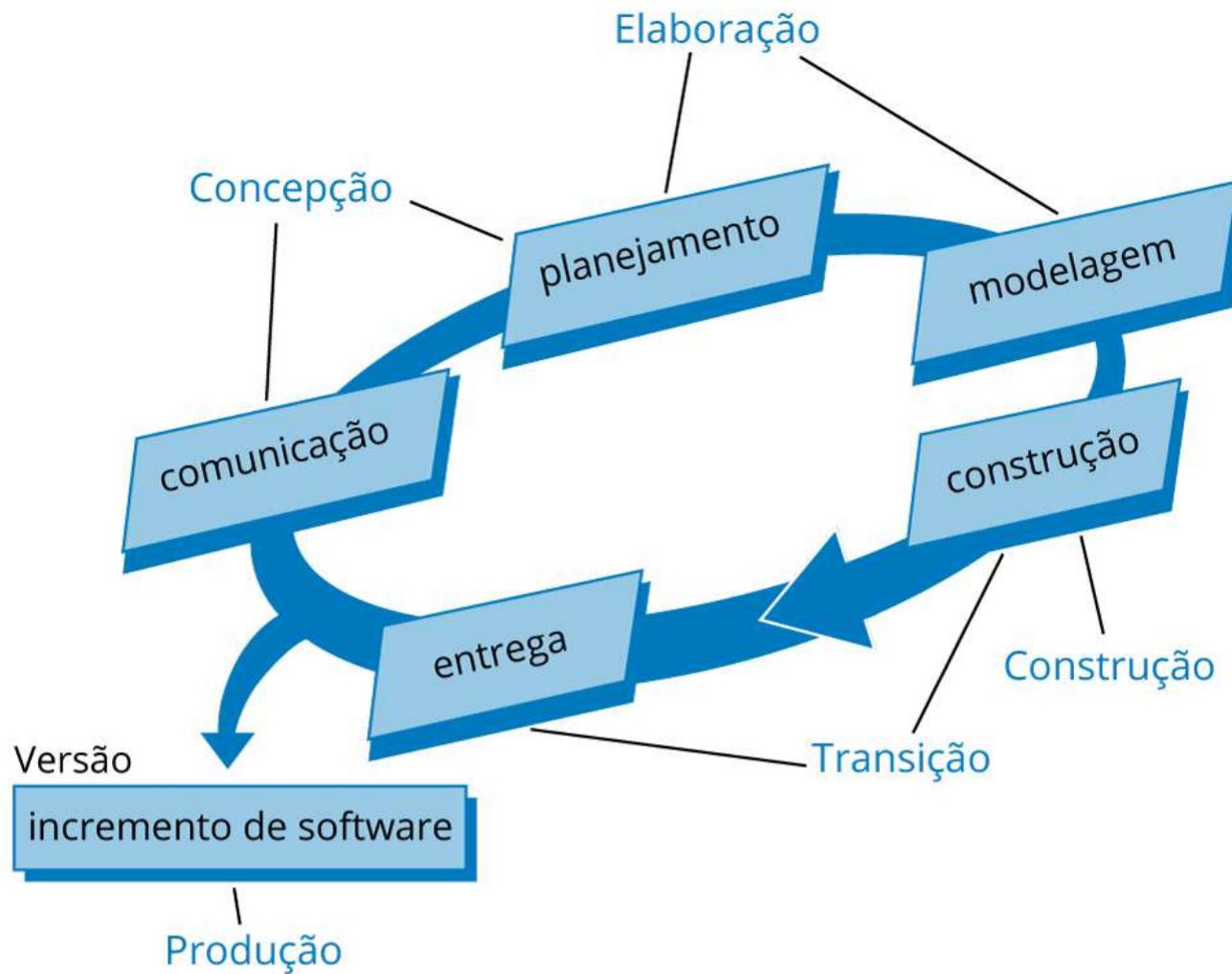
citar os casos de uso que, identificados como mais críticos, são implementados e priorizados no momento do desenvolvimento.

Sendo assim, foi definido o **ciclo de vida do UP**. É importante destacar o que seria o ciclo de vida de um método de desenvolvimento, que nada mais é do que o conjunto de etapas que esse processo desempenhará e como essas etapas são divididas ao longo do tempo de desenvolvimento.

| FASES DO UP

No caso do UP, o ciclo de vida apresenta quatro fases. A Figura 1.4 ilustra o processo unificado e como são organizadas as fases, segundo Pressman e Maxim (2016, p. 57).

Figura 1.4 | Organização do Processo unificado



Fonte: Pressman e Maxim (2016, p. 57).

As quatro fases do processo unificado são descritas a seguir:

- **Concepção:** é definido o escopo do projeto e são elaborados os casos de uso e de negócio. As definições ocorrem após uma conversa com o cliente, momento em que são definidos os casos de uso do sistema – e esse é um dos artefatos dessa parte do fluxo. Elabora-se uma proposta de arquitetura rudimentar que apresenta um estado inicial provisório do sistema e seus subsistemas, que seria outro artefato. Além disso, nessa fase identificam-se os possíveis riscos que podem surgir no decorrer do projeto e propõe-se o cronograma de desenvolvimento do sistema, estimando os custos do projeto considerando as restrições econômicas.
- **Elaboração:** nessa fase do projeto do software as características principais são especificadas com o detalhamento da arquitetura inicial realizada na fase concepção. Assim, na fase elaboração são definidos os requisitos funcionais e refinada a base da arquitetura do software. Nesse momento, os casos de uso desenvolvidos anteriormente são expandidos. A elaboração inclui cinco visões do software, também chamadas de fluxos de trabalho ou disciplinas, e

os artefatos desenvolvidos nessa fase são: casos de uso, análise, projeto, implementação e disponibilização. Nesse momento também ocorre uma revisão detalhada de riscos, escopo do projeto e datas de entrega.

- **Construção:** quando o desenvolvimento do software é realizado, tendo como entrada a arquitetura do software. São construídos ou adquiridos os componentes de software necessários para tornar o sistema funcional para o usuário final. Há uma remodelagem dos modelos de requisito e projeto, para considerar o incremento de software, e são esses os artefatos gerados.
- **Transição:** fase importante do desenvolvimento, quando o produto é entregue para os usuários. Na transição testes são feitos e os incrementos do sistema podem ser implantados. O software em fase beta é testado pelos usuários finais, que relatam os erros e mudanças necessárias. Esse também é o momento no qual a equipe de software elabora o material de apoio (como manuais e tutoriais).

Além dessas, após a transição, de acordo com Pressman e Maxim (2016, p. 57), há a **fase de produção**, em que há um contínuo monitoramento da utilização do software e é realizado o suporte para o ambiente operacional (infraestrutura). Relatórios com problemas e mudanças a serem realizadas são elaborados.

A questão de ser interativo e incremental está associada ao fato de que as fases do ciclo de vida ocorrem simultaneamente e de forma escalonada e não sequencial, como o clássico modelo cascata.

Como é possível perceber nas definições, os artefatos gerados em cada uma das fases do fluxo são as informações internas ou externas, que desempenham papéis no desenvolvimento do sistema.

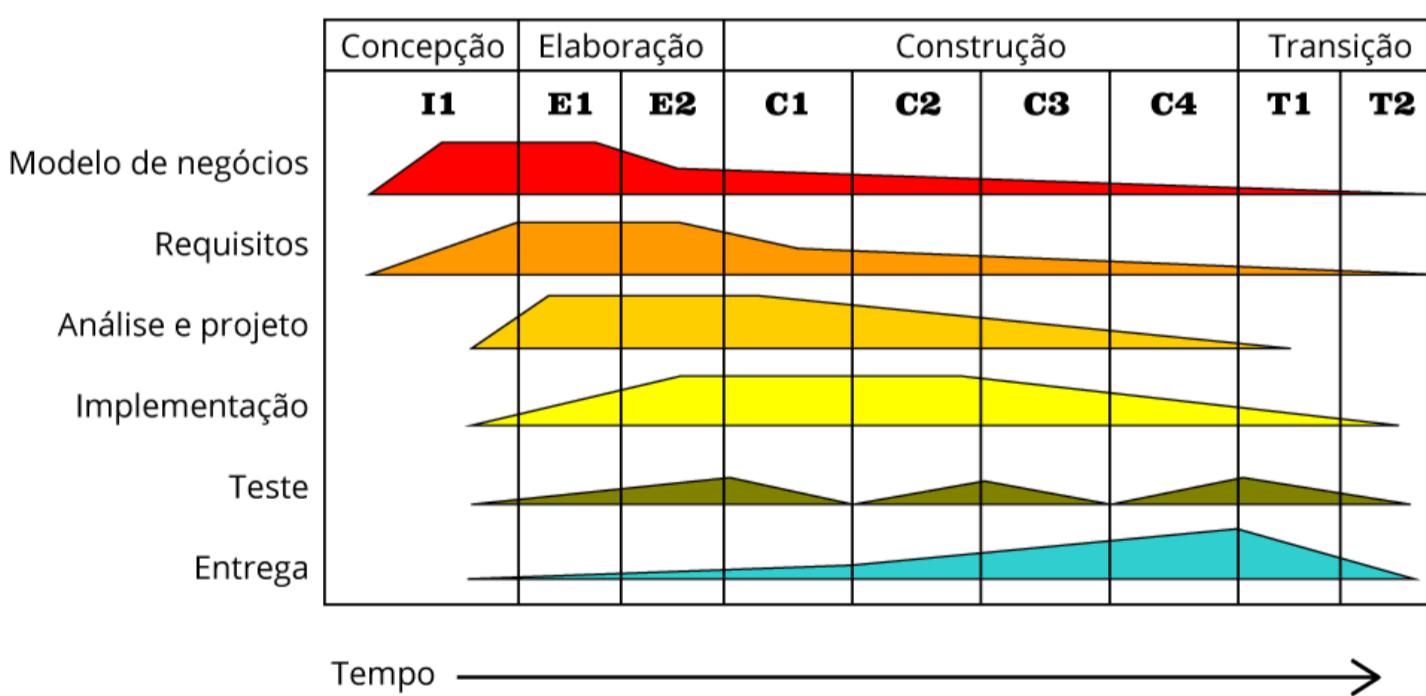
FLUXOS DE TRABALHO (DISCIPLINAS)

O desenvolvimento incremental é proposto da seguinte forma no UP: são feitos vários miniprojetos em tempos limitados, chamados de iterações, e o resultado de cada uma das iterações é um sistema testado, integrado e executável. Assim, as quatro fases do UP são divididas em fluxos de trabalho, também chamadas de disciplinas.

Os fluxos de trabalho correspondem ao conjunto de atividades (disciplinas) e os responsáveis por executar essas atividades. No processo unificado, cada fase pode ser dividida em disciplinas, as quais abrangem todas as fases do UP. Alguns exemplos de disciplinas são o modelo de negócios, requisitos, análise e projeto, implementação, teste e entrega. Para cada disciplina pode-se produzir um ou mais artefatos, que são documentos, manuais, contratos, diagramas, código-fonte e plano de testes, entre outros.

Importa destacar que as fases do UP englobam todo o processo do desenvolvimento de software para cada uma das iterações. Esse comportamento é mostrado na Figura 1.5, uma imagem clássica sobre o UP que apresenta a concorrência entre as quatro fases iniciais.

Figura 1.5 | Desenvolvimento iterativo do UP



Fonte: adaptada de Wikimedia Commons.

Cada um dos produtos das iterações é indicado por uma das colunas da Figura 1.5, porém é possível notar que, dependendo da fase da UP, algumas das disciplinas podem aparecer mais concentradas em determinadas fases (apresentadas pelos gráficos coloridos da figura).

Isso ocorre devido à evolução no desenvolvimento do software, por exemplo: a disciplina Modelo negócios (gráfico vermelho), que corresponde a I1 da Figura 1.5, requer uma maior atenção na fase Concepção, comparando à fase Transição T1 e T2. Nessa fase, o modelo de negócios já deve estar claro e consolidado. Outro exemplo é a disciplina requisitos, que tem uma carga mais concentrada nas fases de concepção e elaboração, quando comparada às fases Construção e Transição (WAZLAWICK, 2014).

o

Ver anotações

REFLITA

Os métodos clássicos de desenvolvimento de software, como o modelo Cascata (PRESSMAN; MAXIM, 2011), apresentavam diversos problemas, principalmente relacionados à necessidade de finalizações de tarefas para iniciar outras – e tais situações podem ser contornadas com a aplicação de métodos iterativos como o UP. Considere o seguinte fato: quando se desenvolve um software em que só se pode executar a versão final, existem vários riscos e problemas, como alto número de erros e a possibilidade de o cliente não ficar satisfeito com o produto final – mudanças nos requisitos causam um grande retrabalho. No caso do UP não acontece isso justamente pelo fato de o desenvolvimento ser iterativo. Você consegue descrever por que a iteratividade no processo – ou seja, a criação de pequenos softwares seguidos pela integração – melhora os requisitos que apontamos como riscos e problemas?

■ RELAÇÃO ENTRE OS DIAGRAMAS E AS FASES

Agora que você conhece melhor o UP, é possível discutir a relação entre os diagramas UML e as fases apresentadas.

Iniciando pela parte da análise de requisitos e modelo de negócios na **fase de concepção**, é possível relacionar diretamente o diagrama de casos de uso que apresenta as principais relações de utilização do

software. Este é acompanhado pelos diagramas de sequência, colaboração, atividade e máquinas de estado, que buscam apresentar o comportamento do software de acordo com a análise dos requisitos e do modelo de negócio.

Em seguida, a **fase de elaboração** (que envolve a análise e projeto) utiliza os diagramas de classe por estar mais próxima da implementação do sistema e, novamente, de sequência, colaboração, atividade e máquinas de estado, porém evoluídos de acordo com a evolução obtida pela construção do diagrama de classes.

Na **fase de construção** também é importante incluir o diagrama de instalação, já que conhecer o hardware e o ambiente de software que serão utilizados na implantação é importante durante o desenvolvimento.

Na **fase de implementação** os principais diagramas utilizados são os de classes, componentes, sequência e colaboração. Perceba que são diagramas que dispõem de informações diretamente relacionadas à implementação do sistema propriamente dito.

O fluxo de trabalho relacionado a testes utiliza a informação de todos os outros diagramas já desenvolvidos, para que seja possível testar todas as funcionalidades do sistema. Analisando a utilização dos diagramas em cada fase do ciclo de vida do UP (apresentada na Figura 1.5), é possível perceber que são escolhidos os diagramas que apresentam a informação necessária e importante para cada etapa.

Ao construir os diagramas UML para cada fase do processo, é importante que se mantenha a consistência dos diagramas desenvolvidos. Lembre-se de que os 14 diagramas UML apresentam diferentes aspectos do mesmo software e, sendo assim, não devem conter informações conflitantes. Os trabalhos de análise de consistência de diagramas UML seguem o padrão de definição de regras para criação dos diagramas que, quando utilizadas, têm por objetivo evitar a inconsistência.

O trabalho de Alsammak, Sahib e Itwee (2018) apresenta uma abordagem interessante das regras de consistência de diagramas UML, com 11 regras básicas, e o trabalho de Harza e Dey (2014) apresenta outras 11 regras, sendo que algumas são referentes a relações entre diagramas diferentes dos casos apontados no primeiro trabalho. Então é possível compilar essas regras de forma sintética no seguinte conjunto:

1. O número de objetos no diagrama de sequência deve ser o mesmo do número de classes presente no diagrama de classes. Isso significa que há inconsistência caso um objeto no diagrama de sequência não tenha classe correspondente.
2. Deve se atentar para as atualizações do diagrama de classes e reproduzi-las corretamente no diagrama de sequência.
3. Se uma relação de dependência é expressa no diagrama de classes, ela deve estar representada por ao menos uma passagem de mensagem entre os objetos dessas classes no diagrama de sequência. Caso não exista relação expressa entre objetos no diagrama de sequência, a relação de dependência no diagrama de classes deve ser removida.
4. O nome dos métodos deve ser respeitado entre os diagramas de classe e sequência; caso contrário uma iteração entre objetos representada no diagrama de sequência pode fazer referência a um método não existente. Caso isso ocorra, é complexo entender se o método deveria existir ou não, se foi um erro apenas de nome ou se foi erro de representação no diagrama de classes.
5. Os diagramas de classe e sequência devem ser sincronizados. As mudanças no diagrama de classe impactam o diagrama de sequência e, se não forem aplicadas corretamente e de modo síncrono, a implementação do sistema pode apresentar erros. Portanto, sempre que uma mudança no diagrama de classes ocorrer deve-se atualizar imediatamente o diagrama de sequência.

6. Os objetos representados no diagrama de comunicação e de sequência devem apresentar um padrão de nomenclatura; isso evita que um objeto da mesma classe tenha nomes diferentes nos dois diagramas. Por exemplo: um objeto chamado de “D” pode ser uma instância da classe X no diagrama de comunicação e da classe Y no diagrama de sequência. Isso cria inconsistência. Adotando-se uma regra de nomenclatura de objetos, esse problema pode ser facilmente evitado.
7. Cada uma das situações representadas no diagrama de casos de uso deve ter uma operação correspondente no diagrama de classes. Isso garante que os casos de uso serão tratados de forma correta na implementação em sua totalidade. Além disso, evita que haja erros de interpretação do tipo “esta operação representa quais casos de uso? ”.
8. Cada caso de uso deve ter um substantivo e um verbo associados. O substantivo do caso de uso corresponde ao nome da classe que o representa no diagrama de classes. O verbo deverá ser relacionado a uma operação representada no diagrama de classes, garantindo inclusive que os diagramas respeitem a regra anterior.
9. Cada caso de uso deve ocorrer pelo menos uma vez no diagrama de sequência, caso contrário há uma inconsistência entre os dois diagramas.
10. Para cada caso de uso representado no diagrama de sequência, todos os atores associados ao caso de uso devem ser representados como participantes. Caso isso não ocorra, o ator do diagrama de casos de uso pode estar representado de forma errada ou o diagrama de sequência pode ter sido construído de forma errada. Nos dois casos há inconsistência.
11. Para cada caso de uso deve existir ao menos um diagrama de sequência.

12. Para cada diagrama de sequência deve existir um diagrama de tempo.
13. Os participantes associados a um determinado diagrama de tempo devem estar representados no diagrama de sequência correspondente.

É importante entender que essas regras, ao serem seguidas, diminuirão o número de inconsistências nos diagramas UML. Porém, não há garantia que os diagramas gerados serão totalmente consistentes. Além disso, as ferramentas de geração de diagramas UML conseguem manter a consistência de diagramas em um mesmo projeto até um certo nível. Assim, em projetos maiores recomenda-se a utilização de softwares de desenvolvimento de diagramas. Existe também uma versão Ágil do UP, que pode ser vista em Ambyssoft (2020).

EXEMPLIFICANDO

Apesar de você ainda não conhecer a fundo os diagramas existentes na linguagem UML, é possível entender o quanto uma inconsistência nos diagramas pode prejudicar o desenvolvimento de software.

Imagine que ao desenvolver o diagrama de casos de uso de um software, o cliente tenha sido enfático com relação a um determinado caso e a equipe tenha negligenciado o ponto 7, sobre cada caso de uso ter uma operação correspondente no diagrama de classes, por achar que esse caso já estava sendo implementado em outra operação.

Ao final da implementação, o caso de uso em questão muito provavelmente não terá sido tratado da forma correta, e o cliente, ao testar o software, perceberá gerando uma necessidade de correção no produto final. O tempo gasto com essa correção vai alocar recursos de desenvolvimento da

empresa que poderiam estar em outros projetos gerando prejuízo financeiros para a empresa, além de deixar o cliente muito insatisfeito com a situação.

Esse caso poderia ter sido evitado caso a inconsistência apontada pelo ponto 7 tivesse sido levado em consideração, poupando tempo de correção do software, horas de programadores e deixando o cliente satisfeito.

Ver anotações

UTILIZAÇÃO DE DIAGRAMAS UML

Uma questão acerca da utilização de diagramas UML para o UP deve ser tratada de forma mais específica. Tanto o UP quanto a UML são independentes de objetivo, ou seja, foram desenvolvidos para o desenvolvimento de qualquer tipo de software. Porém, em alguns casos é preciso saber como utilizar essas ferramentas de forma a obter bons resultados. Um dos tipos de software que demandam uma utilização consciente dos diagramas são os sistemas de informação.

Segundo Laudon e Laudon (2014), um sistema de informação é um conjunto relacionado de componentes de software que apresentam as funções de processar, recuperar, armazenar e distribuir informações em uma organização. Essas informações serão utilizadas para controle, coordenação e tomada de decisões. Esse tipo de software é intimamente relacionado com as questões empresariais, o que deve ser levado em conta em seu desenvolvimento.

Como o foco está nos sistemas de software, é natural que a UML não enfatize os aspectos de um sistema de informações que visam ao valor e suporte que ele pode fornecer aos negócios, como estratégia (por exemplo, cadeias de valor e objetivos estratégicos) e organização (por exemplo, organogramas e processos de negócios). Esses problemas são tratados na "modelagem de negócios" (também chamada de modelagem corporativa). Mesmo assim, a UML dispõe de ferramentas que podem auxiliar nesses casos. Segundo Wazlawick (2014, p. 7), os diagramas que podem auxiliar no processo são os diagramas de casos

de uso de negócio e os de atividade de negócio. Ocasionalmente, os diagramas de máquinas de estado também podem ser utilizados, porém existem ferramentas mais apropriadas para esses casos, como o BPMN (BARBARÁ; VALLE, 2009).

Portanto, é importante saber que no caso de sistemas de informação, a utilização da UML deve ser feita de forma diferenciada em relação ao desenvolvimento de um software com outras finalidades. Além disso, a questão mostra a versatilidade da linguagem UML, que mesmo não tendo sido desenvolvida para aplicações específicas é capaz de apresentar soluções para os problemas enfrentados.

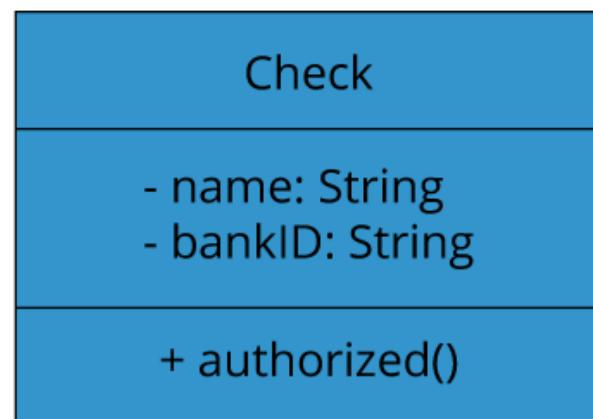
MECANISMOS COMUNS DA UML

O último tópico a ser abordado nesta seção é a questão dos mecanismos gerais da linguagem UML. Existem, portanto, 14 diagramas disponíveis para serem utilizados na linguagem UML e todos eles apresentam particularidades. Entretanto, há alguns mecanismos que são comuns a todos os diagramas e podem ser utilizados para melhorar a legibilidade, bem como incluir mais informações em determinado diagrama sobre o modelo.

Os mecanismos comuns são quatro: **especificações, adornos, divisões comuns e mecanismos de extensão**. Como sabemos, a UML é uma linguagem gráfica, porém associada a cada elemento dessa linguagem gráfica existe uma **especificação** que descreve exatamente aquele elemento. Por exemplo: relacionada a uma classe existe toda a especificação, que descreve todos os atributos, operações e comportamentos que a classe incorpora. Visualmente, o elemento da classe mostrará apenas parte dessa informação. Basicamente, a parte visual permite um entendimento rápido e global de cada parte do sistema, enquanto a especificação permite especificar os detalhes.

Cada diagrama UML começa com um símbolo e depois os adornos são adicionados ao diagrama para compor os elementos da modelagem. Um **adorno**, portanto, é um elemento do diagrama UML que tem uma arte gráfica única e direta, tornando-se uma representação visual objetiva daquele elemento. Veja o exemplo da classe na Figura 1.6. Neste adorno estão os elementos principais da classe, ou seja, seu nome, seus atributos e métodos. Nada mais é necessário para o entendimento global do diagrama de classes, porém essa representação é um adorno da UML.

Figura 1.6 | Adorno classe, em que o nome da classe é *Check*, os atributos são *name* e *bankID* e o método é *authorized()*.



Fonte: elaborada pelo autor.

Divisões comuns: os blocos de construção dos diagramas UML apresentam, em quase todos os casos, uma dicotomia entre sua interface e implementação (UML-DIAGRAMS, 2016). A interface é como um “contrato”, e a implementação é uma das possíveis realizações deste contrato, responsável por complementar a semântica da interface. Essa estrutura é a base do polimorfismo em linguagens orientadas a objetos, quando se define uma interface com vários métodos que serão implementados apenas nas subclasses.

Por fim, os **mecanismos de extensão** foram criados na UML para permitir que sejam feitas modificações na linguagem sem a necessidade de mudar toda a linguagem. Os três mecanismos de extensão da UML são (UML-DIAGRAMS, 2016):

- **Estereótipos:** é possível, na UML, utilizar o “desenho” de um determinado bloco e modificá-lo para um propósito específico, criando um novo objeto.
- **Restrições:** é possível, também na UML, alterar as restrições na construção de um diagrama. Em UML, as restrições são representadas pelas *strings* que acompanham as ligações entre elementos.
- **Valores predefinidos:** é possível predefinir valores específicos em um diagrama, para guiar a implementação do sistema ou gerenciamento de configurações do sistema.

Considerando o que foi colocado na questão dos sistemas de informação, é importante reforçar que cada desenvolvimento de software apresentará suas particularidades, e a linguagem UML está preparada para lidar com isso. Seus diagramas são versáteis a ponto de permitir sua utilização na modelagem de diversos tipos de sistemas. Além disso, a existência dos mecanismos de extensão da linguagem permite que ela seja modificada de acordo com as necessidades da modelagem de um software específico. Essas características tornam a UML uma linguagem extremamente útil e poderosa no desenvolvimento de sistemas.

Chegamos ao final da seção, e agora você já é capaz de relacionar a utilização dos diagramas UML com o método UP de desenvolvimento. Além disso, comprehende o que é necessário para manter a consistência dos diagramas desenvolvidos e o motivo dessa consistência ser tão importante durante o desenvolvimento. O caso do desenvolvimento de sistemas de informação também foi apresentado para demonstrar o que ocorre em sistemas que apresentam particularidades. Por fim, você conheceu os mecanismos comuns da linguagem UML, sendo capaz de aprender a fundo como construir seus diagramas e como modelar softwares utilizando a linguagem. Bons estudos!

REFERÊNCIAS

ALSAMMAK, I. L. H.; SAHIB, H. M. A.; ITWEE, W. H. A method of ensuring consistency between UML Diagrams. **Journal University of Kerbala**, v. 16, n. 1, 2018.

AMBYSOFT INC. **The Agile Unified Process**. Ambysoft, 2020. Disponível em: <https://bit.ly/2P8yikw>. Acesso em: 17 jun. 2020.

BARBARÁ, S.; VALLE, R. **Análise e Modelagem de Processos de Negócio**. São Paulo: GEN Atlas, 2009. 232 p.

HARZA, R.; DEY, S. Consistency between Use Case, Sequence and Timing Diagram for Real Time Software Systems. **International Journal of Computer applications**, v. 85, n. 16, 2014.

JACOBSON, I.; BOOCH, G.; RUMBAUGH, J. **The Unified Software Development Process**. Massachusetts, EUA: Addison-Wesley, 1999.

LAUDON, K.; LAUDON, J. **Sistemas de Informação Gerenciais**. 11. ed. Campinas: Pearson Universidades, 2014. 504 p.

PRESSMAN, R. S.; MAXIM, B. R. **Engenharia de Software**. 8. ed. Porto Alegre: AMGH, 2016.

UML – DIAGRAMS. **The Unified Modeling Language**. 2016. Disponível em: <https://bit.ly/3f7qM41>. Acesso em: 28 maio 2020.

WAZLAWICK, R. S. **Análise e Design Orientados a Objetos para Sistemas de Informação**. São Paulo: GEN-LTC, 2014. 488 p.

FOCO NO MERCADO DE TRABALHO

O PROCESSO DE DESENVOLVIMENTO DE SOFTWARE COM UML

Maurício Acconcia Dias

Ver anotações

A UML E OS PROCESSOS DE DESENVOLVIMENTO

Para cada fase do processo de desenvolvimento de software é possível relacionar diagramas específicos e a construção dos diagramas deve ser feita com consistência.



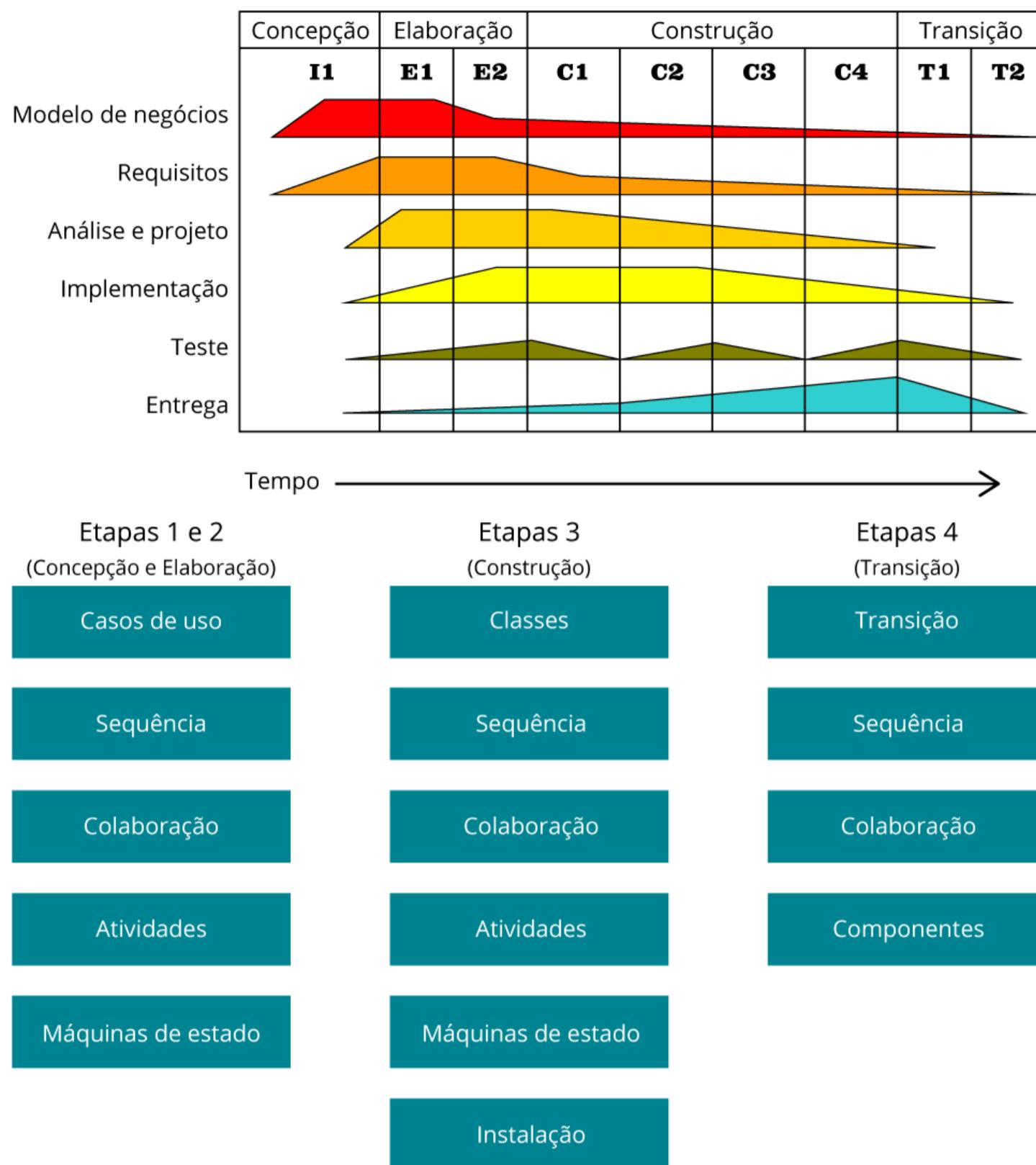
Fonte: Shutterstock.

Deseja ouvir este material?

Áudio disponível no material digital.

Existem diversas formas de apresentar a relação entre o UP e a linguagem UML. A principal delas é a apresentação de uma relação entre as fases do UP e qual diagrama utilizar em cada uma delas. Uma das melhores formas de resolver o problema é utilizando a Figura 1.5 e indicando os diagramas relativos a cada uma das etapas.

Figura 1.7 | Associação dos diagramas UML com as fases do fluxo de trabalho do UP



Fonte: elaborada pelo autor.

A figura demonstra os diagramas utilizados em cada uma das etapas e de acordo com os fluxos de trabalho. O fluxo de trabalho relacionado a testes utiliza todos os diagramas implementados anteriormente, para que seja possível verificar e validar a implementação, além do teste propriamente dito. Na entrega também é necessário utilizar os diagramas, já que um erro ou funcionalidade que não foi implementada deve ser verificada em todo o processo.

NÃO PODE FALTAR

MODELAGEM DE CASOS DE USO

Maurício Accocia Dias

Ver anotações 0

DIAGRAMA DE CASOS DE USO

Diagrama de fácil entendimento sobre o sistema, utilizado para apresentar a ideia geral do projeto aos donos do software, pois descrevem um conjunto de ações que o sistema deve executar em conjunto com os usuários externos.



Fonte: Shutterstock.

Deseja ouvir este material?

Áudio disponível no material digital.

Bem-vindo ao início do estudo da construção e da utilização dos diagramas da linguagem UML, a qual é utilizada para a modelagem de processos de software em forma de diagramas. Nesta unidade serão mostrados diferentes tipos de diagramas UML e, para isso, será preciso tratar, basicamente, dos principais componentes do sistema e de como eles se relacionam. Por exemplo, ao ir até o caixa de um banco pagar uma conta, você seria o usuário que interage com a pessoa que manipula o caixa (atores do sistema). Em seguida, você irá pagar a conta (transação que ocorre ou um caso de uso, mais especificamente), conta esta que pode ser paga com dinheiro, com cartão ou com seu próprio saldo do banco. Se essa relação fosse modelada, todos os componentes deveriam estar presentes. Para isso, temos os diagramas de casos de uso, que apresentarão essas relações; além disso, temos também o diagrama de classes, que apresentará as classes utilizadas para modelar os componentes desse sistema, como a conta bancária, o que será pago, a transação do pagamento, entre outros itens.

Aprender sobre os diagramas e sobre suas principais características é importante, porém outra parte fundamental do estudo de UML é saber como construir um determinado diagrama e como utilizá-lo para obter o melhor resultado no processo de desenvolvimento de sistemas.

Lembre-se de que a linguagem UML é uma ferramenta para a modelagem de sistemas, uma tarefa primordial e essencial a ser realizada antes do desenvolvimento de sistemas. Sendo assim, a construção dos diagramas é imprescindível para que, ao final do processo do desenvolvimento de software, os requisitos sejam atendidos e o produto final satisfaça as necessidades do cliente.

Ao estudar o conteúdo desta unidade, você será capaz de trabalhar com três dos principais diagramas existentes na linguagem UML: o **diagrama de casos de uso**, responsável por descrever um conjunto de ações que os sistemas devem executar em conjunto com usuários externos ao sistema; o **diagrama de classes**, que será utilizado para modelar a estrutura do sistema a ser desenvolvido considerando o nível de classes e interfaces, e, por fim, o **diagrama de atividades**, que apresenta o fluxo de controle dos objetos com ênfase na sequência em que os eventos ocorrem e nas condições para que ocorram.

Esses diagramas são amplamente utilizados para a modelagem de sistemas e, ao adquirir o conhecimento de quando e como construí-los da forma correta, você perceberá que eles são de grande importância para a aplicação no mercado de trabalho. O desenvolvimento de um software, em geral, inicia-se pelo levantamento de requisitos, no qual teremos listado tudo o que o software deverá fazer, ou seja, as suas funcionalidades. Por exemplo, um software para controle de vendas deve ter requisitos sobre as operações a serem realizadas (compra, venda, troca), sobre como elas devem ser feitas e sobre as informações importantes para a empresa. Os requisitos de usuário são utilizados para a construção dos diagramas de casos de uso. Além isso, a partir dos requisitos obtidos e dos diagramas de caso de uso gerados, posteriormente, serão construídos os demais diagramas da UML. Esse

diagrama é comumente utilizado pois o cliente normalmente é capaz de interpretá-lo e de identificar problemas nas relações. O diagrama de classes serve de base para todo o desenvolvimento do software, pois representa a estrutura do sistema como um todo. As classes são as unidades fundamentais do software e tudo que envolve sua implementação está representado no diagrama. Por fim, para a correta análise da relação de objetos, é necessário construir o diagrama de atividades.

Com o conhecimento apresentado nesta unidade, será possível compreender melhor a importância da linguagem UML e aprender como utilizá-la na construção de três de seus principais diagramas. Vamos lá? Bons estudos!

PRATICAR PARA APRENDER

Bem-vindo à seção de Modelagem de casos de uso. Nela você será apresentado aos conhecimentos necessários para a construção dos diagramas de casos de uso. A linguagem UML é uma ferramenta amplamente utilizada em empresas de desenvolvimento de software para a modelagem de sistemas. Ela possui quinze diagramas, que, juntos, apresentam diversas visões do software em desenvolvimento, auxiliando durante o processo e melhorando o resultado final. Além disso, os times envolvidos no desenvolvimento do software são capazes de obter as informações de forma coerente, diminuindo muito os problemas de interpretação das informações sobre o projeto.

Todo projeto de desenvolvimento de software inicia-se pela fase de obtenção dos requisitos. Os requisitos de software estão concentrados em um documento que possui tudo que o cliente espera do produto final. Nesse sentido, o primeiro diagrama UML desenvolvido é o diagrama de casos de uso, que irá modelar todas as possíveis utilizações do sistema de uma forma simples e de fácil entendimento. O diagrama de casos de uso é utilizado inclusive em reuniões com o cliente para verificação.

Sem as informações consistentes do diagrama de casos de uso fica mais complexa a tarefa de desenvolver os outros diagramas, já que é necessário revisar sempre o documento de requisitos para verificação e validação.

Com um diagrama de casos de uso bem feito, é possível obter êxito no desenvolvimento de vários outros diagramas da linguagem UML.

Nesta seção você irá aprender os principais conceitos do diagrama, como identificar seus componentes, como fazer a documentação suplementar dos casos de uso de forma correta e completa e, por fim,

irá aprender a elaborar diagramas com estudos de caso de problemas reais.

Você foi contratado por uma empresa de desenvolvimento de software e, ao iniciar seus trabalhos, percebeu que a empresa passava por dificuldades com os projetos. Os prazos sempre eram perdidos, o software final possuía uma taxa de erros muito elevada e, no final das contas, as correções e manutenções do software consumiam muito tempo dos programadores.

Após perceber toda essa situação e se atentar para o fato de que a empresa não utilizava nenhuma linguagem de modelagem de software, você propôs a seus diretores uma apresentação sobre a linguagem UML, seus benefícios, seus principais diagramas e como essa linguagem poderia solucionar diversos problemas encontrados por você no processo de desenvolvimento de software da empresa.

Os donos da companhia concordaram e então você pôde fazer uma série de apresentações. Todos da empresa gostaram da ideia e, após esse momento, a linguagem UML passou a integrar o desenvolvimento de softwares logo no primeiro projeto da empresa do qual você estava fazendo parte como desenvolvedor.

A tarefa designada ao projeto é desenvolver um sistema de gerenciamento para um novo banco, porém, como se tratava de uma instituição recém-criada, os proprietários da instituição propuseram que a empresa de desenvolvimento que você trabalha apresentasse uma primeira versão dos requisitos, a fim de que eles tivessem uma ideia das funcionalidades para, então, modificarem e acrescentarem outras necessidades de acordo com o sistema desejado.

Você achou estranho pois sabe que um documento de requisitos não seria a melhor opção de apresentação do sistema, e criar um diagrama de casos de uso sem um conjunto de requisitos seria uma tarefa complexa. Então teve a seguinte ideia: procurar por sistemas de bancos

já existentes e tentar obter um diagrama de casos de uso que apresentasse ao menos as funcionalidades básicas esperadas para esse tipo de sistema.

Sendo assim, sua tarefa é apresentar um diagrama de casos de uso de um sistema que irá gerenciar um banco simples, o qual só possui contas correntes de clientes de pessoas físicas e jurídicas. Bom trabalho!

DICA

Pesquise sobre sistemas de gerenciamento de bancos e suas funcionalidades.

Pronto para conhecer melhor o diagrama de casos de uso e suas características? Bons estudos!

CONCEITO-CHAVE

A linguagem UML é uma poderosa ferramenta de modelagem e pode ser aplicada em diversas etapas do desenvolvimento de software com o objetivo de obter um resultado melhor ao final do processo. A criação dos diagramas possibilita o detalhamento visual de diversos aspectos do software, auxiliando a equipe de desenvolvimento e reduzindo os erros provenientes do entendimento equivocado sobre os aspectos do sistema. Com isso, o uso da UML é amplamente utilizado no processo de desenvolvimento de software de muitas empresas.

Apesar de apresentar os resultados mencionados, é importante que os diagramas sejam construídos de maneira correta para que as informações necessárias sobre o sistema estejam neles expressas.

Nesta seção iremos abordar o diagrama de casos de uso, que é um diagrama simples e importante no desenvolvimento do software por apresentar o comportamento do sistema em relação a seus possíveis usuários.

DIAGRAMA DE CASOS DE USO

Normalmente o diagrama de casos de uso é construído após o levantamento dos requisitos dos usuários, pois nesse momento os desenvolvedores possuem todas as informações necessárias para sua elaboração. Além disso, esse diagrama é uma ferramenta visual de fácil entendimento sobre o sistema, podendo, em muitos casos, ser utilizada para apresentar a ideia geral do projeto aos donos do software.

De forma sucinta, é possível dizer que o diagrama de casos de uso captura a funcionalidade e os requisitos do sistema usando **atores**, **casos de uso** e os **relacionamentos** entre eles (RUBAUGH; JACOBSON; BOOCHE, 2004).

- **Atores:** são os usuários do sistema, representam quem irá interagir com ele.
- **Casos de uso:** representam as diferentes formas através das quais o usuário pode interagir com o sistema, ou seja, os modos como cada usuário poderá utilizá-lo.
- **Relacionamentos:** representam como cada caso de uso interage com outros casos de uso ou com os atores.

Os casos de uso modelam os serviços, tarefas e funções que um sistema precisa executar. Representam, também, funcionalidades de alto nível e como um usuário manipula o sistema. Vamos agora conhecer os principais componentes desse diagrama e o que eles representam.

O diagrama de casos de uso tem definições complementares com relação à sua natureza. Nas definições da linguagem UML de 2.0 a 2.4, era considerado tanto um diagrama comportamental quanto estrutural (UML-DIAGRAMS, 2016). A classificação estrutural do diagrama de casos de uso era definida com base em sua relação com o diagrama de classes, que é estrutural. Já na UML 2.5, o diagrama de casos de uso foi retirado do conjunto de diagramas comportamentais e colocado em conceitos suplementares, criando um dilema em sua classificação (UML-DIAGRAMS, 2016).

O primeiro componente do diagrama que iremos estudar é o “caso de uso”. Representado por uma elipse preenchida pelo nome do caso de uso, como apresentado na Figura 2.1, é usado para representar funcionalidades de alto nível e para demonstrar como o usuário manipulará o sistema. Um caso de uso representa uma funcionalidade diferente de um sistema, componente, pacote ou classe. Segundo as regras da UML, um caso de uso deve ser nomeado obrigatoriamente, porém nenhuma definição sobre como isso deve ser feito é apresentada, deixando a tarefa a cargo do criador do diagrama. Entretanto, é recomendado pelas diretrizes da linguagem que se utilize uma frase curta no formato VERBO+SUBSTANTIVO, como “sacar dinheiro”.

Figura 2.1 | Notação de caso de uso

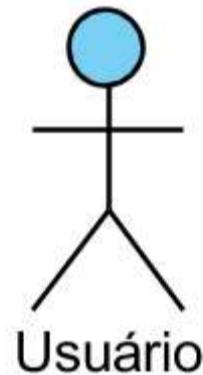


Fonte: elaborada pelo autor.

ATOR

O segundo componente do diagrama de casos de uso que iremos discutir é o ator, componente de hardware ou software, externo ao sistema, com quem interage. Esse componente é chamado de ator por representar um papel de entidade externa ao software e por interagir com o sistema. Um usuário seria o melhor exemplo de ator, sendo este uma entidade que inicia o caso de uso fora do escopo de um caso de uso. Pode ser qualquer elemento que acione uma interação com o caso de uso. Um ator pode ser associado a vários casos de uso no sistema. A notação desse elemento na UML é apresentada na Figura 2.2.

Figura 2.2 | Notação de ator



Fonte: elaborada pelo autor.

As especificações da UML, até a UML 2.5, exigem que a funcionalidade do caso de uso seja iniciada por um ator. Na UML 2.5, isso foi removido, o que significa que pode haver algumas situações em que a funcionalidade do sistema seja iniciada pelo próprio sistema, enquanto ainda fornece resultados úteis a um ator. Por exemplo, o sistema pode notificar um cliente que o pedido foi enviado, agendar limpeza e arquivamento de informações do usuário, solicitar algumas informações de outro sistema etc. (UML-DIAGRAMS, 2016).

ASSIMILE

É importante destacar que o diagrama de casos de uso é o primeiro a ser construído no processo de modelagem de sistemas, e as informações para a sua construção são obtidas

diretamente do documento de requisitos. Imagine um requisito de um sistema de vendas que indique que “o operador deve poder escolher entre um pagamento de cartão de crédito ou dinheiro”. Nessa circunstância será criado um caso de uso no qual o ator seja o operador do caixa, usuário do sistema, e em que existam dois casos de uso. Os nomes dos casos de uso, seguindo o padrão da UML, poderiam ser “pagamento cartão” e “pagamento dinheiro”.

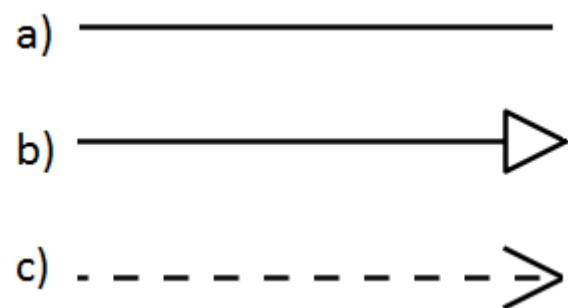
o

Ver anotações

RELACIONAMENTOS

Outros componentes de um diagrama de casos de uso são os seus relacionamentos. Os relacionamentos expressam o tipo de interação entre outros componentes do diagrama (atores e casos de uso). Na figura 2.3 são ilustradas as notações dos três principais tipos de interações.

Figura 2.3 | Relações do diagrama de casos de uso



Legenda: a) associação; b) generalização; c) extensão/inclusão.

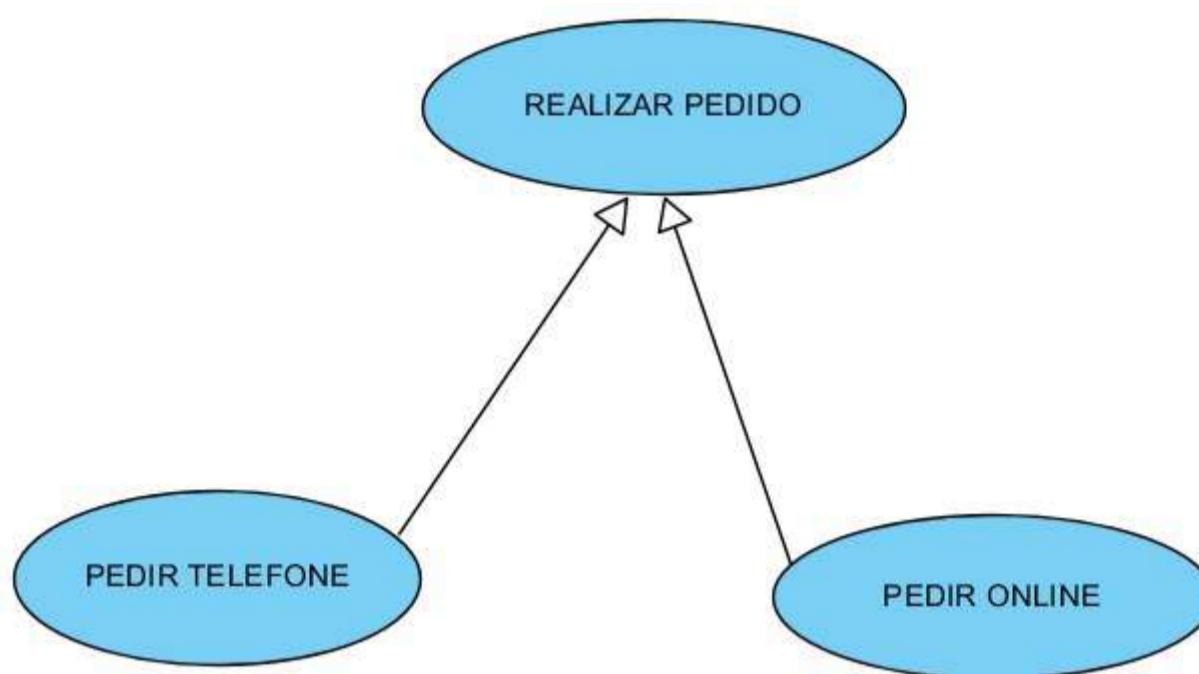
Fonte: elaborada pelo autor.

A **Associação** indica um relacionamento ou a comunicação entre um ator e um caso de uso. A notação da associação é uma linha que liga os elementos relacionados, a qual pode ser vista na Figura 2.3(a). É importante destacar que o relacionamento de associação pode ser direcionado pela inclusão de uma seta que indica a direção na linha da associação. Importa destacar que não existe no modelo um caso de uso que inicia por dois atores.

A **generalização** entre casos de uso é a mesma relação que se tem de generalização de classes. Na questão da herança de classes, a classe filha herda as características da classe pai, sendo que a classe filha possui todos os seus atributos e métodos visíveis. A representação nos casos de uso define que um caso de uso filho herda propriedades e o comportamento do caso de uso pai e deve inclusive redefinir seu comportamento. Imagine a situação de um login em um determinado sistema, ele pode ser feito de diversas formas, como: digitando o login e a senha, utilizando a função “lembrar” do navegador, ou inscrevendo-se na página. Nesse cenário teríamos um caso de uso pai, chamado de autenticação, e casos de uso filho, utilizados para redefinir a forma de autenticação do caso de uso pai (UML-DIAGRAMS, 2016). Considere o seguinte exemplo: em um sistema de pedidos, há duas maneiras de se realizar um pedido, ou pela internet ou pelo telefone.

Independentemente de qual meio foi utilizado para realizar esse pedido, ele deve ser feito da mesma forma. A Figura 2.4 apresenta esse exemplo com os casos de uso e sua generalização.

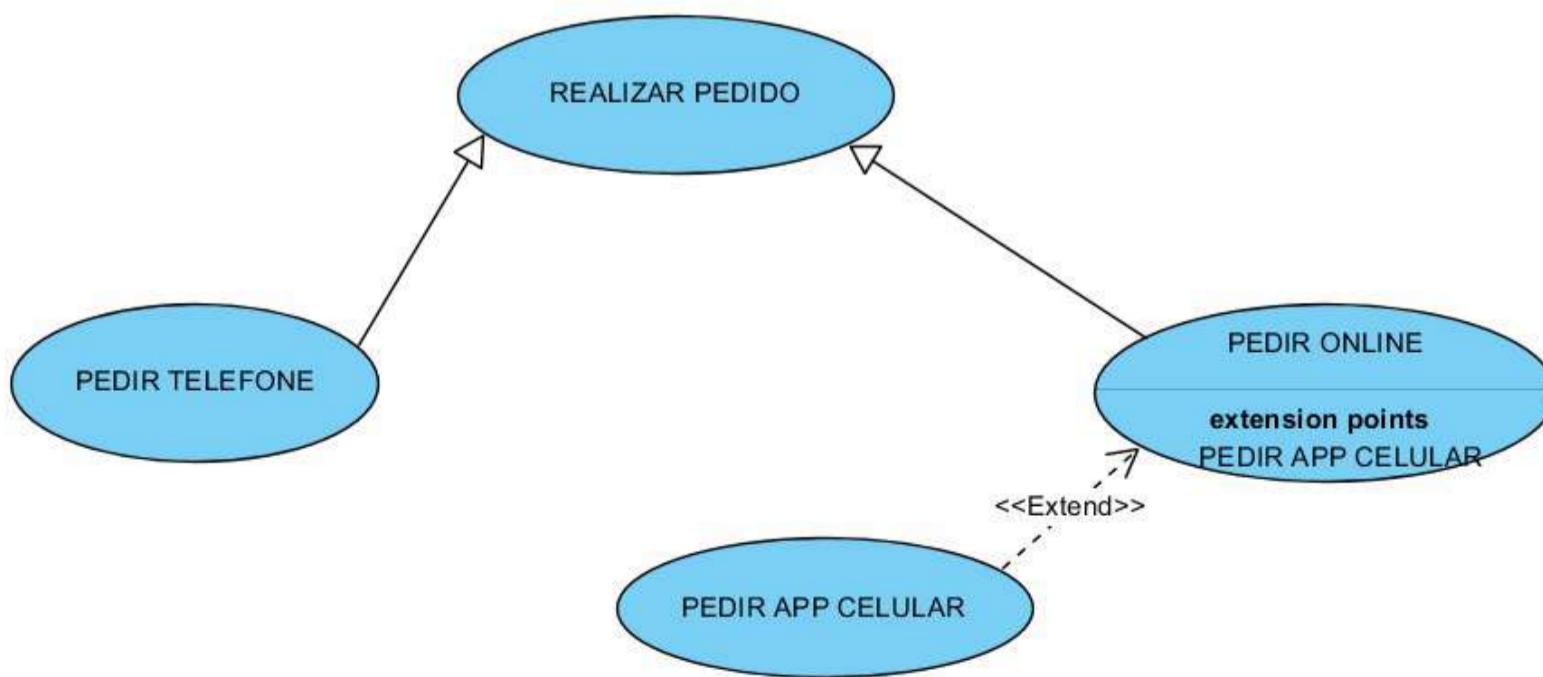
Figura 2.4 | Exemplo de generalização de casos de uso



Fonte: elaborada pelo autor.

O relacionamento do tipo “**extend**” de um caso de uso indica um comportamento opcional/adicional, ou seja, é executado em apenas determinadas situações. Esse tipo de relacionamento especifica como e quando o comportamento definido no caso de uso extensivo pode ser inserido no comportamento definido no caso de uso estendido (UML-DIAGRAMS, 2016). Trata-se de um comportamento adicional a um caso de uso, sem criação de dependência. Porém, o caso de uso extensivo não faz sentido se analisado isoladamente. Um exemplo geral é a opção de ajuda em qualquer parte do sistema: imagine um sistema que possua uma forma correta de preencher um formulário e possua um botão de ajuda no preenchimento. Esse caso de uso “ajuda no preenchimento” pode estender o caso de uso “preencher formulário”, já que não torna o caso de uso principal dependente, mas apenas complementar à sua funcionalidade. Repare que, se analisarmos o caso de uso extensivo “ajuda no preenchimento” isoladamente, ele não fará sentido. Considerando o exemplo anterior e adicionando um caso de extensão, é possível que um pedido on-line seja feito de um computador pessoal ou do aplicativo do celular. A Figura 2.5 apresenta a extensão necessária para modelar o comportamento do pedido on-line, considerando que a forma “padrão” seria pedir de um computador.

Figura 2.5 | Exemplo de relação de extensão



Fonte: elaborada pelo autor.

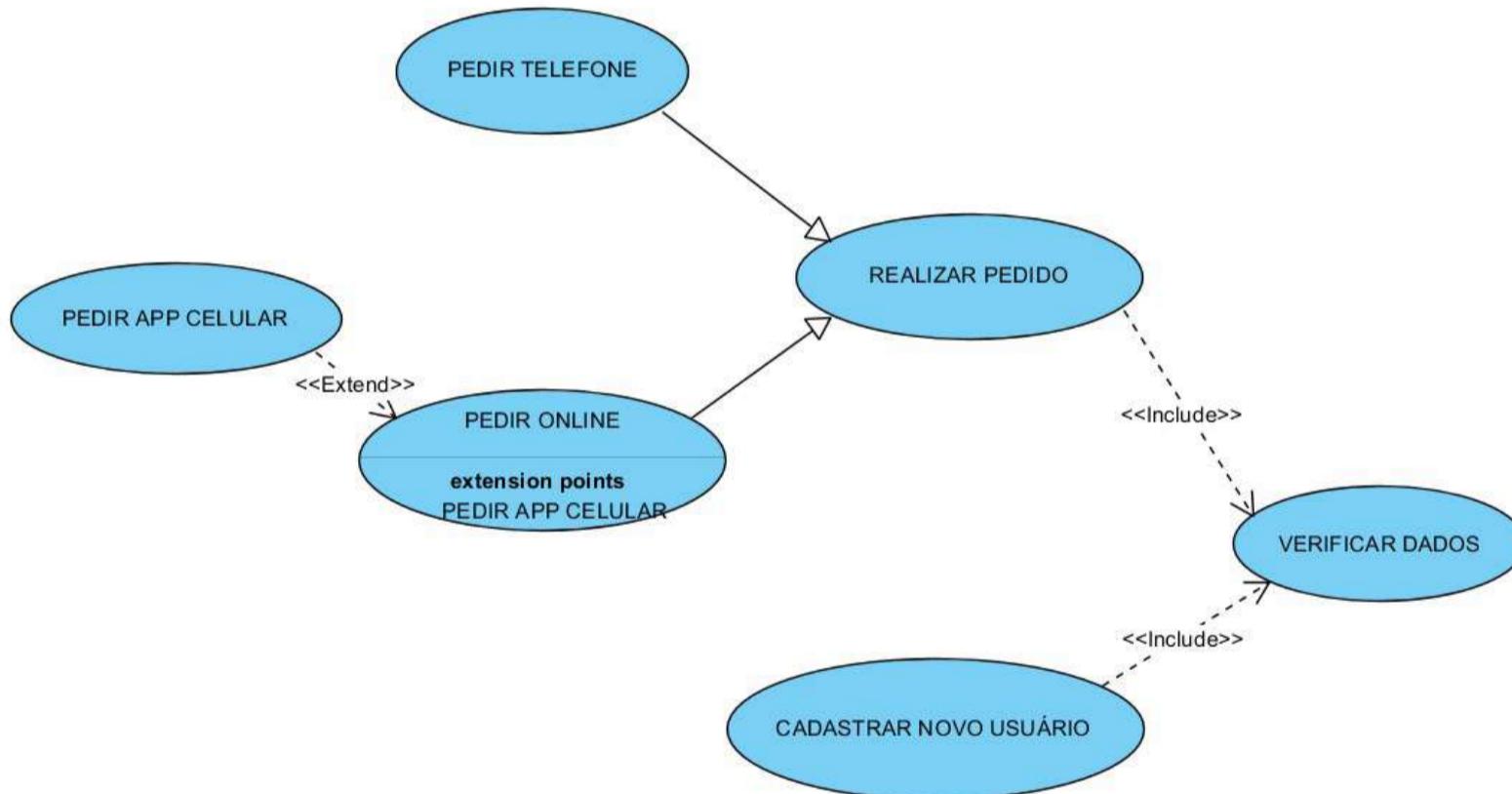
O relacionamento do tipo “**include**” (inclusão) é utilizado para representar comportamentos parciais que são comuns para outros casos de uso, ou seja, são divididos em casos de uso menores que compartilham funcionalidades. Outro exemplo de utilização é quando deseja-se decompor um caso de uso complexo. Em casos de usos menores e mais simples, esse tipo de tratativa ajuda no entendimento do caso de uso. A relação de inclusão indica obrigatoriedade, a execução de um primeiro obriga a execução de um segundo caso de uso associado.

o

Ver anotações

O comportamento do caso de uso a ser incluído deve ser inserido no comportamento do caso de uso inicial. Imagine o sistema de um supermercado, no qual, tanto para pagar um produto quanto para saber seu preço, é preciso escanear o código de barras. Sendo assim, é interessante definir um caso de uso “escanear item” para ser incluído em outros dois casos de uso: “pagamento” e “consulta preço” (UML-DIAGRAMS, 2016). Considere ainda que, no exemplo do pedido realizado, exista uma funcionalidade do sistema que verifica dados; isso fará com que o pedido tenha seus dados verificados, assim como o cadastro de um novo usuário no sistema. A Figura 2.6 apresenta um diagrama com a representação desses casos de uso.

Figura 2.6 | Exemplo de relacionamento de inclusão



Fonte: elaborada pelo autor.

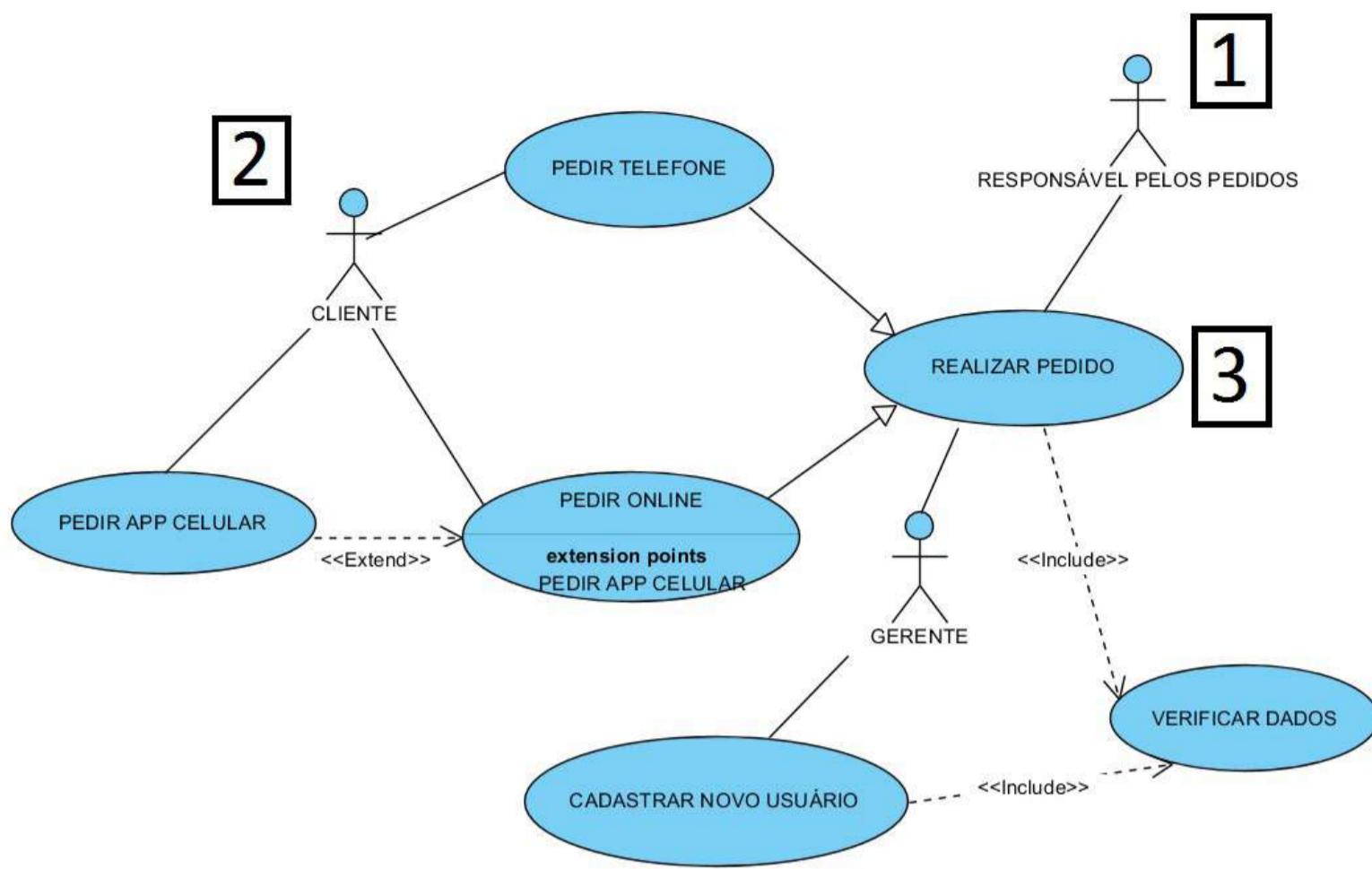
Analisando, ainda, o mesmo problema do pedido e colocando um contexto mais abrangente, foi possível identificar possíveis relacionamentos de generalização, extensão e inclusão. Após analisar os relacionamentos entre casos de uso, vamos verificar a relação entre atores e casos de uso. A associação é um relacionamento que envolve um ator e pode ocorrer de três maneiras distintas.

Considere, agora, a inclusão de três atores no diagrama da Figura 2.6 e suas relações de associação. Pode haver uma associação entre:

- Um ator e um caso de uso (caso 1 presente na Figura 2.7).
- Um ator e múltiplos casos de uso (caso 2 presente na Figura 2.7).
- Múltiplos atores e um caso de uso (caso 3 presente na Figura 2.7).

De acordo com a UML, deve-se ter no mínimo um ator associado a um caso de uso.

Figura 2.7 | Exemplos de associações com atores



Fonte: elaborada pelo autor.

Também é possível que um ator seja associado a diversos casos de uso, por exemplo, quando um usuário utiliza um programa de edição de texto e existem diversas opções de utilização. Nesse caso, o ator que representa o usuário no diagrama desse sistema vai estar associado a todos os possíveis casos de uso. Por fim, considere uma empresa que possui um sistema que classifica tipos diferentes de usuário. Todos estes tipos de usuário irão utilizar o mesmo caso de uso de login configurando a associação de múltiplos atores e um caso de uso.

O relacionamento de generalização também pode ser aplicado a atores. A generalização de um ator significa que ele pode herdar o papel de outro ator, herdando inclusive todos os casos de uso do pai. Também é possível definir mais casos de uso específicos para o ator filho.

EXEMPLIFICANDO

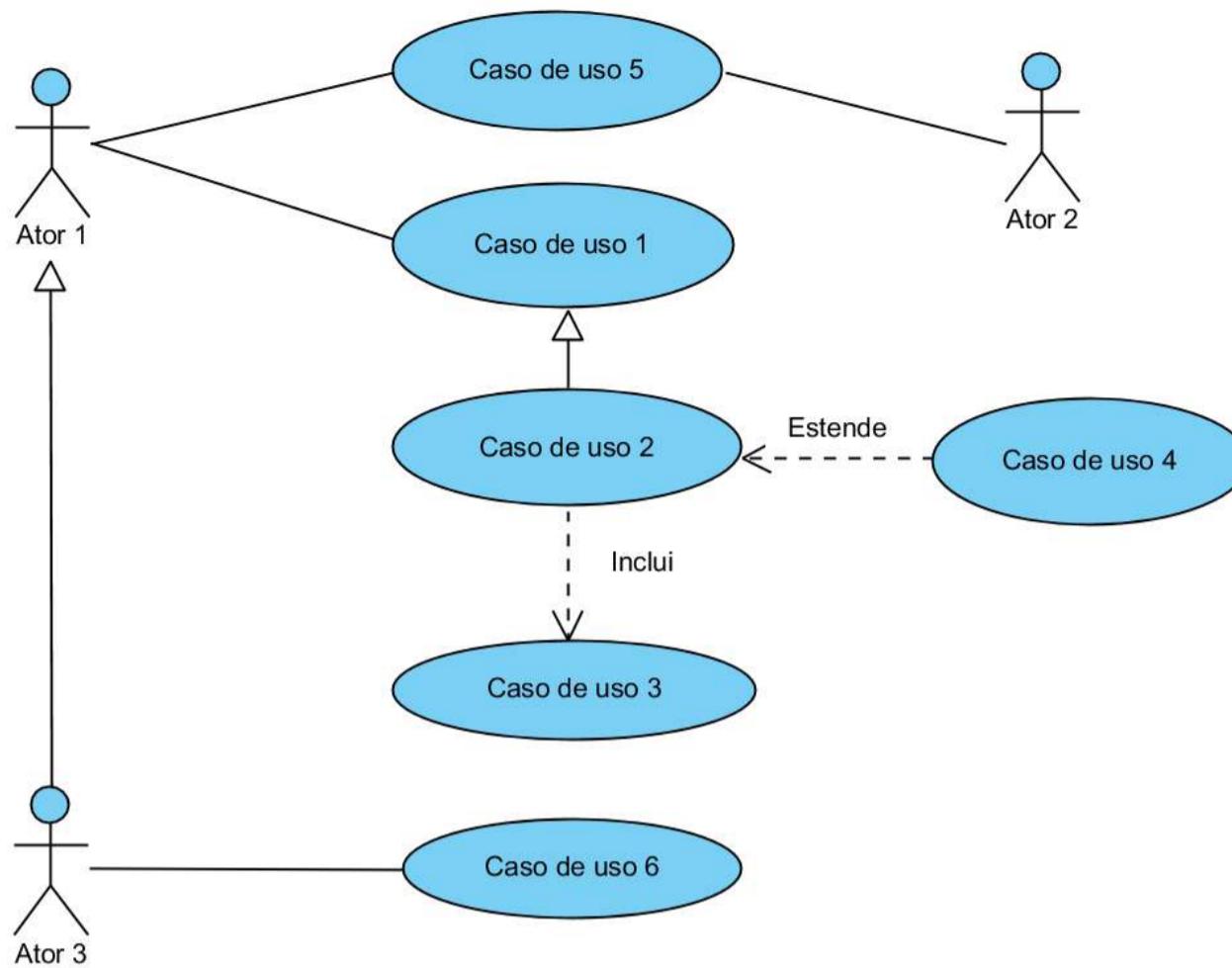
Foram definidos até aqui os componentes básicos de um diagrama UML de casos de uso, sendo eles:

- Os atores.

- Os casos de uso.
- Os relacionamentos – associações, generalizações, inclusões e extensões.

Sendo assim, é possível construir um exemplo de diagrama de casos de uso com as relações apresentadas anteriormente.

Figura 2.8 | Exemplo de diagrama de casos de uso



Fonte: elaborada pelo autor.

Na Figura 2.8 são apresentados exemplos de todos os relacionamentos e componentes apresentados até o momento:

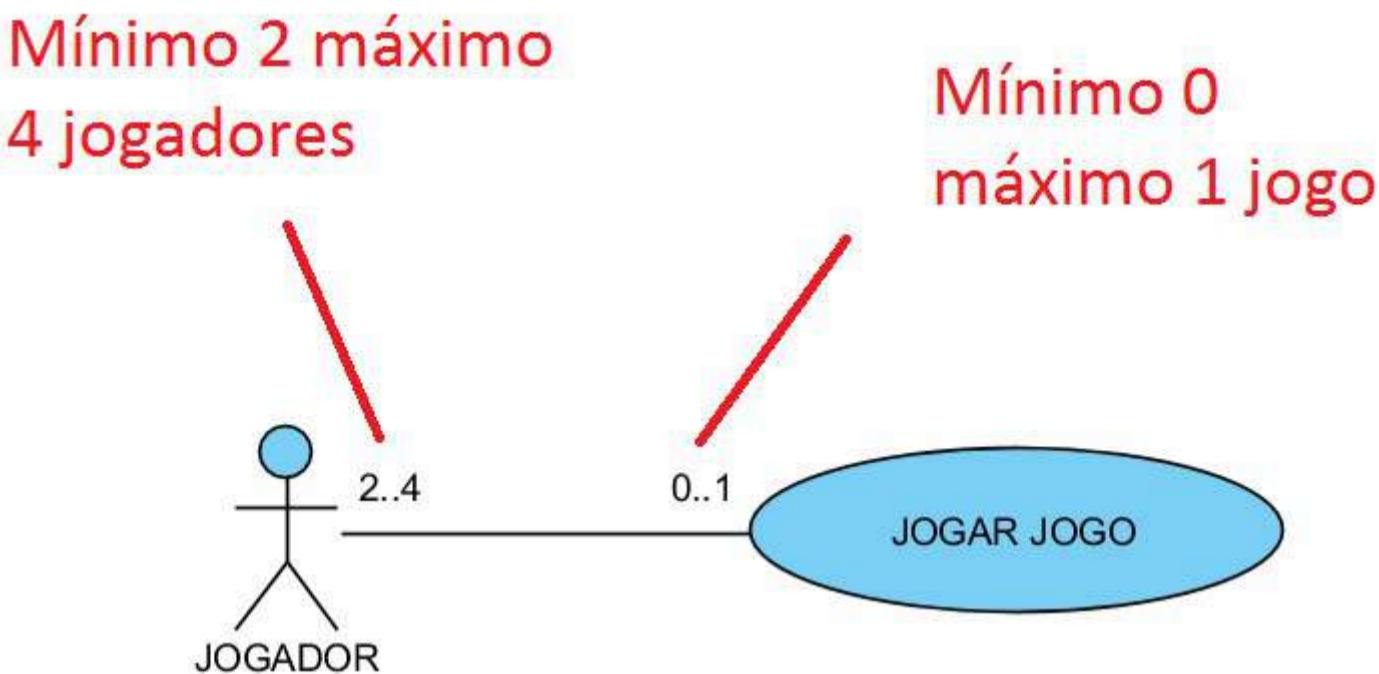
- Generalização entre os casos de uso 1 e 2.
- Generalização entre os atores 1 e 3.
- Inclusão do caso de uso 3 no caso de uso 2.
- Extensão do caso de uso 2 pelo caso de uso 4.
- Associação simples do ator 3 com o caso de uso 6.
- Associação múltipla dos atores 1 e 2 com o caso de uso 5.
- Associação múltipla do ator 1 com os casos de uso 1 e 5.

MULTIPLICIDADE

Os diagramas de caso de uso também podem incluir a questão da multiplicidade de associações. Nessa situação é possível indicar que mais de um ator esteja executando determinado caso de uso ao mesmo

tempo. Um bom exemplo disso é a questão de jogos. Imagine o ator jogador e o caso de uso jogar um jogo. Cada jogador pode estar jogando nenhum jogo ou um jogo. Cada jogo pode ter um número determinado de jogadores. Um exemplo desse tipo de relação aplicada ao diagrama de casos de uso é ilustrado na Figura 2.9, na qual um jogo pode ser jogado por, no mínimo, dois e, no máximo, quatro jogadores, e um jogador pode estar jogando um ou nenhum jogo. A forma correta de indicar multiplicidade está indicada na Figura 2.9.

Figura 2.9 | Exemplo de multiplicidade



Fonte: elaborada pelo autor.

Caso exista apenas uma possibilidade, ao invés de utilizar a notação “n..n”, utilize apenas um número. Por exemplo, se o jogo aceitar apenas um jogador, coloque “1” no lugar do “2..4” da Figura 2.9.

Existem basicamente dois tipos de requisitos: os **funcionais** e os **não funcionais**. Os requisitos funcionais são relativos às funcionalidades e informações do sistema, mostrando o que deve ser feito. Os requisitos não funcionais definem restrições de tempo, espaço, softwares de apoio, sistema operacional etc. (SOMMERVILLE, 2011). O diagrama de casos de uso é a melhor ferramenta para representar requisitos funcionais, porém os requisitos não funcionais são um problema. Neste caso é possível utilizar o artefato chamado documentação suplementar para os casos de uso. Essa documentação normalmente possui um formato padrão, que varia de acordo com a empresa. Um modelo desse formato pode ser encontrado em Helm ([s.d.]).

Para elaborar adequadamente um diagrama de casos de uso, é importante seguir alguns critérios básicos:

- O diagrama deve ser o mais simples possível.
- O diagrama também deve ser o mais completo possível.

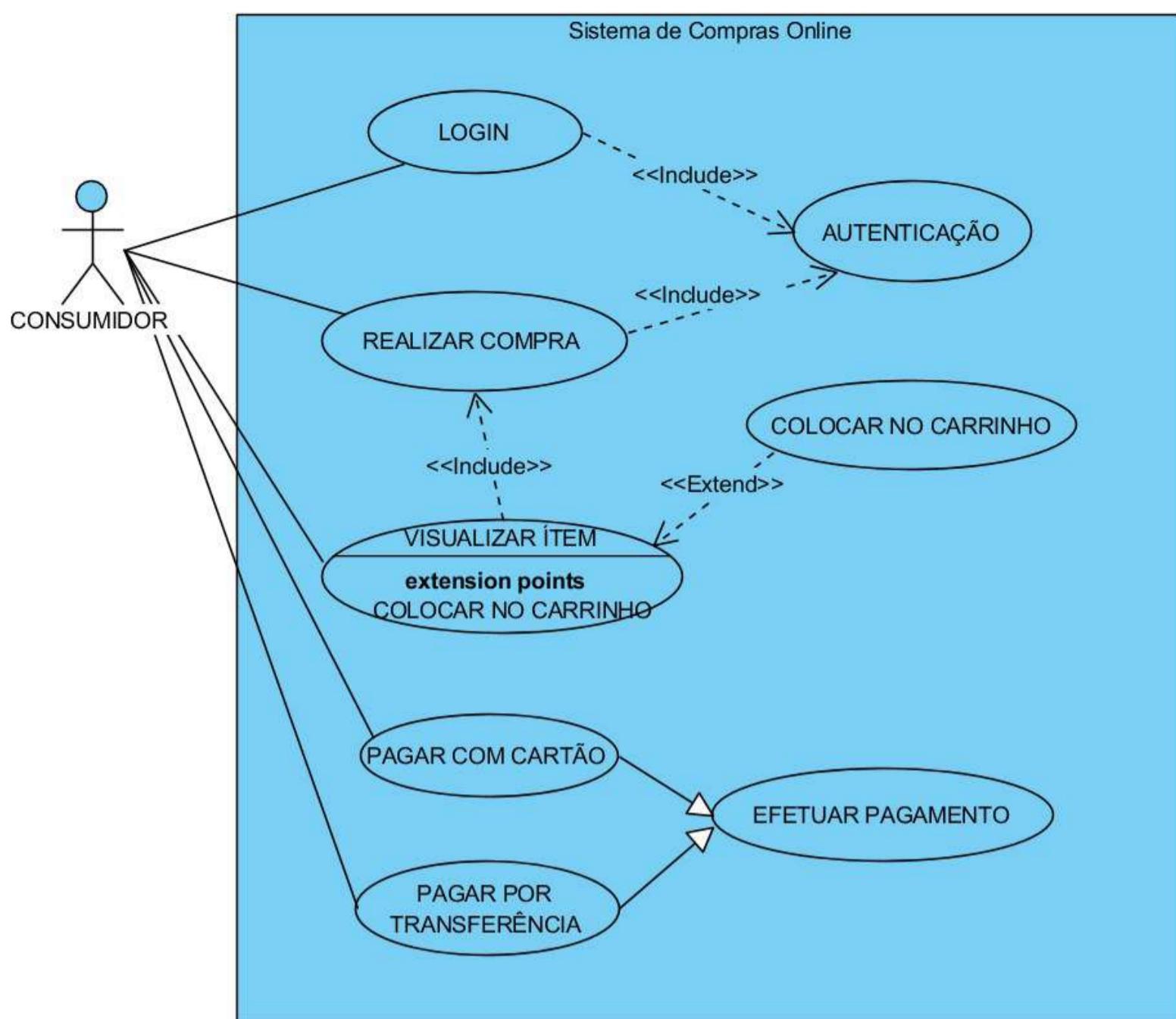
- Um caso de uso deve ter todas as suas interações representadas.
- Se o diagrama é muito grande, é possível representar apenas as partes essenciais.
- Um diagrama de casos de uso deve modelar ao menos um módulo do sistema de forma completa.

Vamos agora fazer um exemplo de construção de diagramas de casos de uso. Iniciaremos com um sistema simples de biblioteca. Considere os seguintes requisitos de projeto:

- Trata-se de um sistema de compras on-line.
- Todo consumidor deve logar no sistema para fazer uma compra.
- O login passa por um sistema de autenticação.
- O consumidor pode ver os itens que deseja comprar; ao ver um item, pode comprá-lo diretamente ou colocá-lo no carrinho.
- O pagamento pode ser feito por meio de cartão ou de transferência bancária.

O conjunto de requisitos apresentados pode ser desenvolvido de várias formas possíveis. Vamos analisar a solução proposta na Figura 2.10 e entender como os relacionamentos, atores e casos de uso alocados atendem aos requisitos do projeto. O diagrama da Figura 2.10 foi construído com a ferramenta *Visual Paradigm*.

Figura 2.10 | Possível solução para o problema da compra on-line



Fonte: elaborada pelo autor.

Os casos de uso apresentados na Figura 2.10 atendem aos requisitos do projeto. A relação de inclusão do sistema de autenticação, nos casos login e realizar compra, garantem que o consumidor esteja logado. Ao ver um item, o consumidor pode comprar diretamente, por isso a inclusão do caso de uso realizar compra. Além disso, ele pode optar por colocar no carrinho, o que estende o caso de uso por adicionar uma funcionalidade específica ao caso de uso visualizar item. O pagamento pode ser efetuado por cartão ou transferência, porém o resultado é o mesmo, e essa é a justificativa da generalização entre esses casos de uso. Pode ser visto também na Figura 2.10 o limitador de escopo do sistema, que é o retângulo presente em volta dos casos de uso, deixando os atores na parte externa.

Quando o sistema apresenta um cenário conhecido e estabelecido, é possível, em uma primeira versão do diagrama de casos de uso, que se construa uma “proposta” de um sistema mais funcional ainda que os requisitos do cliente não cubram todas partes necessárias. Isso é

importante pois o cliente muitas vezes não tem ideia das características do sistema e o diagrama de casos de uso completo pode ser apresentado a ele para que decida se é valido ou não.

o

REFLITA

A linguagem UML é uma ferramenta de modelagem que pode ser aplicada juntamente com os métodos de desenvolvimento de software. Considerando esse fato, é possível dizer que toda a experiência adquirida no desenvolvimento de software pode ser aplicada na elaboração dos diagramas já pensando nos possíveis erros a serem evitados e funcionalidades essenciais que devem ser implementadas e não foram definidas nos requisitos. O sistema da biblioteca da Figura 2.5 está muito simples ainda apesar de permitir a implementação de todos os requisitos. Você conseguiria pensar em novos casos de uso para serem adicionados ao diagrama?

Dica: considere os casos de uso adicionais que podem ser estendidos e incluídos nos que já estão presentes.

O diagrama de casos de uso é importante pois, além de apresentar o sistema em relação ao usuário, faz isso de forma simples. Essa simplicidade permite que o diagrama seja utilizado em reuniões com o cliente para visualização do sistema como um todo e para validação dos requisitos apresentados. Com a teoria apresentada nesta seção, foi possível ver as partes que compõem um diagrama de casos de uso, uma versão genérica com diversas relações apresentada na Figura 2.4 e um caso real de sistema de biblioteca apresentado na Figura 2.5. Agora que você já sabe os princípios básicos, é hora de começar a praticar. Bons estudos!

REFERÊNCIAS

Ver anotações

HELM, J. C. Supplementary Specification. **University of Houston** – Clear Lake, Houston, [s.d.]. Disponível em: <https://bit.ly/32LXdTd>. 2020. Acesso em: 12 jun. 2020.

SOMMERVILLE, I. **Engenharia de Software**. 9. ed. [S.I.]: Pearson Universidades, 2011.

RUBAUGH, J.; JACOBSON, I.; BOOCH, G. **The Unified Modeling Language Reference Manual**. 2. ed. [S.I.]: Pearson Higher Education, 2004.

UML-DIAGRAMS. **The Unified Modeling Language**, [S.I.], 2016. Disponível em: <https://bit.ly/3f7qM41>. Acesso em: 19 maio 2020.

UNHELKAR, B. **Software engineering with UML**. [S.I.]: CRC Press, 2018.

WAZLAWICK, R. S. **Análise e design orientados a objetos para sistemas de informação**. [S.I.]: GEN-LTC, 2014. 488p.

FOCO NO MERCADO DE TRABALHO

MODELAGEM DE CASOS DE USO

Maurício Accocia Dias

Ver anotações

ATORES, CASOS DE USO E RELACIONAMENTOS

No diagrama de casos de uso, os atores, casos de uso e relacionamentos são usados para representar a funcionalidade e os requisitos do sistema.



Fonte: Shutterstock.

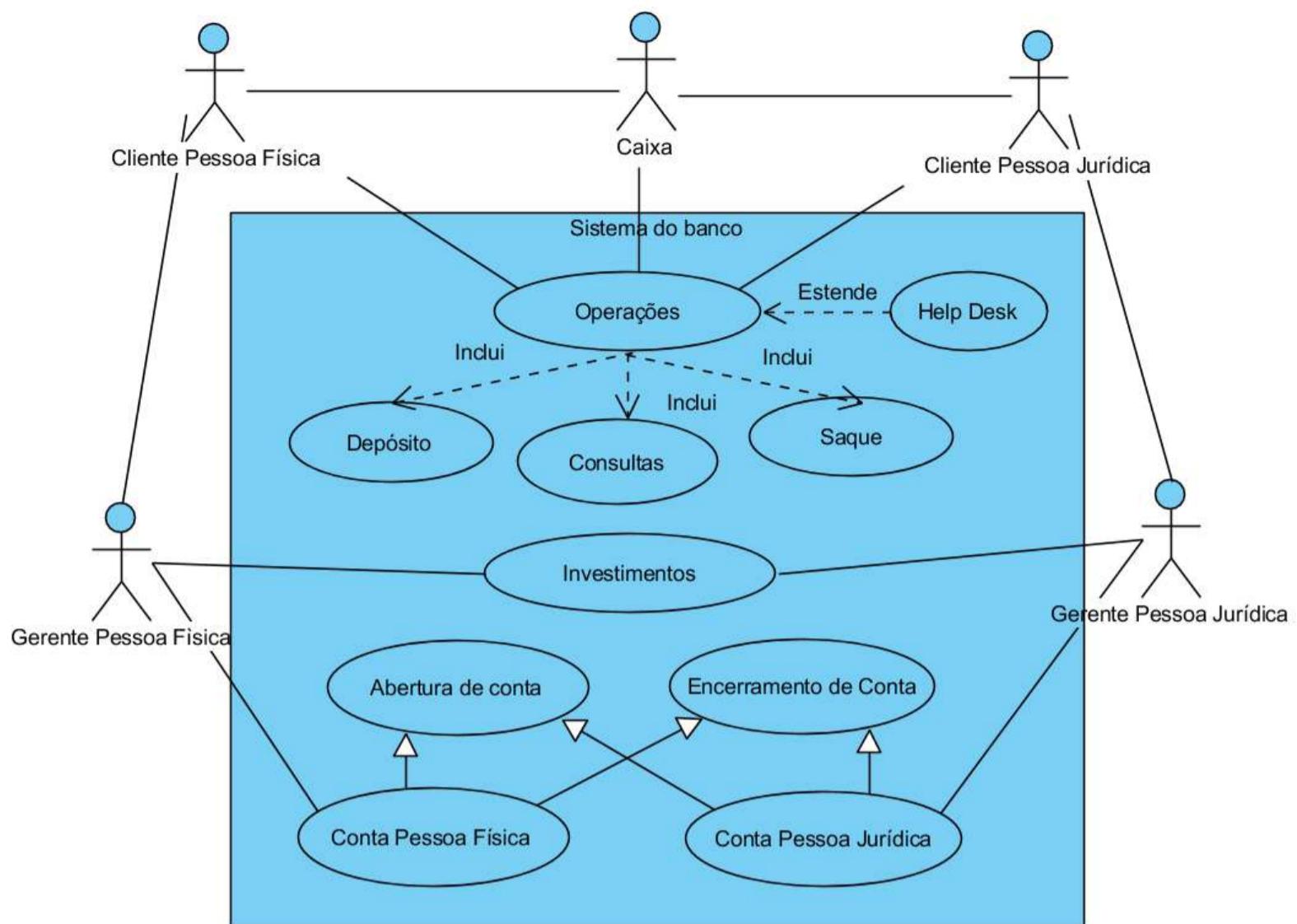
Deseja ouvir este material?

Áudio disponível no material digital.

SEM MEDO DE ERRAR

A solução para esse problema envolve um sistema que considera, no mínimo, o ator cliente, os diferentes tipos de gerentes (no caso dois) e o caixa, que realiza funções diferentes das do cliente. Além disso, um banco permite que sejam feitas operações de depósito, saque, pagamento de contas, investimentos, criação de contas, encerramento de contas, consultas, entre outras operações. Existem diversas maneiras de elaborar esse diagrama, porém seria interessante abordá-lo da forma mais simplificada possível utilizando o máximo de elementos. Portanto, uma das propostas de solução para o problema é apresentada na Figura 2.11.

Figura 2.11 | Diagrama de casos de uso de um sistema de banco



Fonte: elaborada pelo autor.

NÃO PODE FALTAR

MODELAGEM DE CLASSES

Maurício Accocia Dias

Ver anotações 0

DIAGRAMA DE CLASSES E OBJETOS

Utilizados para modelar a estrutura do sistema a ser desenvolvido, considerando o nível de classes e interfaces, também mostra o relacionamento entre os componentes do sistema.



Fonte: Shutterstock.

Deseja ouvir este material?

Áudio disponível no material digital.

Seja bem-vindo à seção que trata do diagrama de classes. Nela iremos trabalhar um dos diagramas mais utilizados da linguagem UML. Ao modelar um problema do mundo real, como um sistema de vendas, um controle de estoque ou um sistema bancário, é mais intuitivo utilizar uma linguagem orientada a objetos visto que a abstração dos objetos do mundo real facilita o entendimento da modelagem de sistemas, a qual pode ser feita de maneira mais simples em relação ao que estamos acostumados a encontrar no nosso dia a dia.

Um exemplo de problema do mundo real que podemos abordar é o de um sistema de controle bancário. Bancos geralmente possuem sistemas que devem considerar tanto a parte de dados dos clientes e de suas contas quanto a hierarquia da distribuição das agências por cidades. Além disso, existem os diversos serviços oferecidos por essas instituições, como contas e empréstimos.

É importante destacar que a abstração das linguagens orientadas a objetos inicia-se com as classes, que nada mais são do que a abstração dos objetos do mundo real. Essas classes são “materializadas” por meio de um software. Portanto, é possível criar a classe banco e depois especificá-la para cada parte em seu sistema. Por exemplo, pode-se criar uma classe extrato, que armazena diversos modelos, como conta corrente e poupança, e que controla suas quantidades, sua data de entrada e saída, entre outras informações.

Dessa forma, pode-se observar que as classes e objetos estão presentes na modelagem de vários sistemas presentes no mercado. Sendo assim, entender como criar o diagrama de classes é muito importante, pois permite a visualização do sistema em relação à sua estrutura.

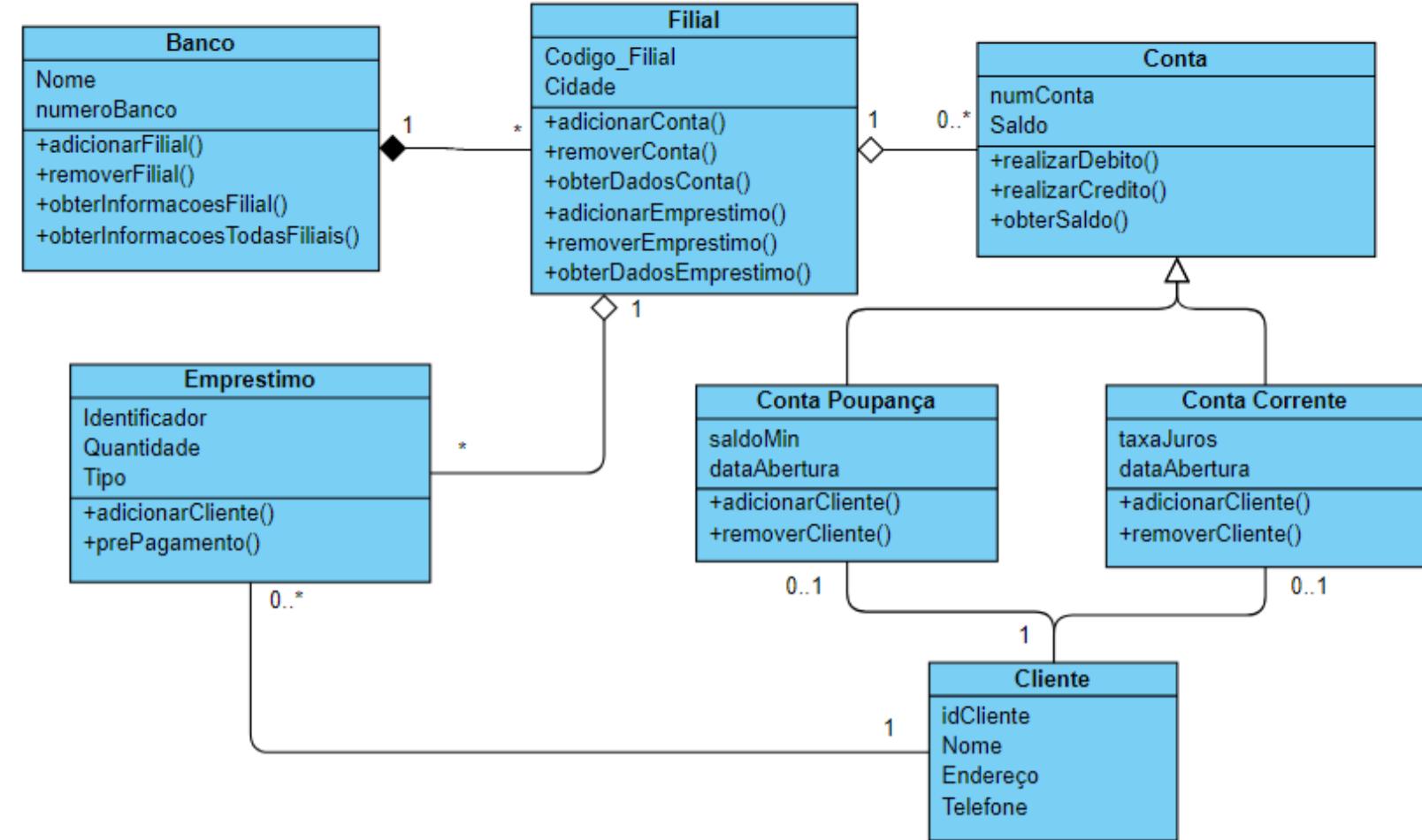
O diagrama de classes também se relaciona diretamente com o diagrama de objetos, que irá apresentar uma instância específica do diagrama de classes em um determinado momento da execução o software.

Imagine agora que, em um sistema bancário, temos uma classe inicial responsável por criar o banco. Além dessa classe outras são necessárias, como a que lida com as filiais do banco e suas agências; as responsáveis por modelar os tipos de contas oferecidas pelo banco e as que modelam clientes e serviços.

É possível, nesse caso, modelar um sistema de forma que a estrutura seja feita por classes mais simples e que a especificação seja realizada por subclasses herdeiras das características gerais. Os relacionamentos entre todas as classes citadas também podem ser explorados, por exemplo composições e agregações gerando um diagrama que modela situações reais ocorridas em uma agência bancária.

Você agora é um profissional que já apresenta diversas habilidades em UML. Ao ser contratado por uma empresa, foi alocado na equipe de engenharia de software e, devido à sua experiência com a linguagem UML, ficou responsável por realizar a modelagem de sistemas. A questão é que, para que seja testado seu nível de entendimento a partir de um diagrama de classes, foi-lhe mostrado um diagrama de um problema real desenvolvido pela empresa. Esse problema foi modelado a partir do diagrama de casos de uso e sua equipe irá desenvolver o software. Como você está chegando nesse momento, deve demonstrar ao líder de sua equipe que é capaz de entender o diagrama e as relações apresentadas nele.

Figura 2.12 | Diagrama de classes de um banco



Fonte: elaborada pelo autor.

Com base no diagrama apresentado, explice as classes e as relações existentes, evidenciando os detalhes existentes entre elas, os atributos e os relacionamentos apresentados na modelagem.

Agora, com os conhecimentos sobre os diagramas de classes e objetos, você será capaz de modelar grande parte dos sistemas exigidos pelo mercado utilizando a linguagem UML. Preparado? Então, bons estudos.

CONCEITO-CHAVE

Caro aluno, agora chegamos à seção de diagrama de classes e objetos da UML. Tais conhecimentos são fundamentais para a modelagem e, posteriormente, para o desenvolvimento de softwares. Para o completo entendimento desse conteúdo, é fundamental compreender o que são classes e objetos e quais são suas relações.

Um dos aspectos fundamentais para o uso do paradigma orientado a objetos é a necessidade constante de reutilização de código (DEITEL; DEITEL, 2016). Ao se programar de forma estruturada, por meio da construção de funções, existe uma separação entre os dados e seu processamento, ou seja, uma função recebe dados de qualquer parte do programa e esses dados são processados. Todavia, ao se utilizar o

paradigma orientado a objetos, os dados são integrados ao seu processamento, possibilitando um maior controle sobre eles e seu processamento.

CONCEITOS BÁSICOS DE ORIENTAÇÃO A OBJETOS

Existem alguns conceitos básicos do paradigma de orientação a objetos que devem estar claros para você entender os diagramas que serão apresentados. São eles:

- **Atributos:** são variáveis que armazenarão as características do objeto que se está modelando. Por exemplo, ao modelar um objeto carro, podemos pensar nas seguintes características (atributos): ano, modelo, cor, número de portas, manual, automático.
- **Métodos:** são as ações/comportamentos que podem ser desempenhados pelo objeto que se está modelando. No caso do objeto carro, podemos citar: acelerar, frear, mudar a marcha, dar a partida, desligar.
- **Classes:** representam o elemento abstrato de um conjunto de objetos; em outras palavras, uma classe possui toda a especificação do objeto, como as características (atributos) e as ações/comportamentos (métodos) dele. O carro, que utilizamos como exemplo até aqui, pode ter os atributos cor, ano, modelo, capacidade do motor; além disso, pode conter os métodos: acelerar, frear, dar a partida e desligar. Nesse caso, estamos lidando com a classe carro.
- **Objetos:** o objeto carro seria uma instância da classe carro, ou seja, um carro azul do ano de 1998 modelo GT2 com motor V8 de 4.0 litros. Este carro especificamente descrito é um objeto da classe carro com os atributos descritos.

EXEMPLIFICANDO

Para entender melhor os conceitos atributos, métodos, classes e objetos, vamos pensar no caso da bicicleta. Ao criar uma classe bicicleta, temos como atributos:

- Ano.
- Marca.
- Modelo.
- Aro.
- Tipo de freio.

E como métodos podemos pensar em:

- Pedalar para frente.
- Pedalar para trás.
- Virar.
- Frear.

Portanto, esta seria a classe bicicleta. Agora, para exemplificar a instância dessa classe, vamos pensar em bicicletas específicas:

- Ano: 2018; marca: trovão; modelo: sx45; aro: 18; freio: disco.
- Ano: 1963; marca: bike; modelo: a23; aro: 20; freio: normal.

Os atributos de cada uma das bicicletas são específicos, porém elas possuem os mesmos métodos já que são bicicletas.

Herança também chamada de generalização é um dos relacionamentos entre classes e pilares da orientação à objetos, a qual indica que uma classe é criada a partir de uma classe existente apenas adicionando atributos e métodos à sua definição original (DEITEL & DEITEL, 2016).

Nesse sentido, pode-se afirmar que quando uma classe herda de outra, estamos definindo que a nova classe terá tudo que a anterior (ou superclasse) tinha em adição ao quer for definido na nova classe (ou subclasse).

EXEMPLIFICANDO

Para exemplificar o conceito de herança, vamos definir a classe bicicleta_motorizada a partir da classe bicicleta anteriormente definida, que possui os atributos:

- Ano, marca, modelo, aro, tipo de freio.

E os métodos:

- Pedalar para frente, pedalar para trás, virar e frear.

É possível definir que uma bicicleta motorizada tenha todas as características de uma bicicleta comum e, além disso, tenha como atributo adicional o motor e como método adicional acelerar. Sendo assim, em vez de definir toda uma nova classe bicicleta_motorizada com todos esses atributos e métodos, é definida a classe nova apenas com um atributo (motor) e um método (acelerar), herdando todos os outros métodos da classe bicicleta.

Outra vantagem da herança é que tudo que for melhorado, modificado ou corrigido na superclasse impacta diretamente na subclasse. Isso remove a necessidade de correção em diversas partes de um mesmo código tornando o desenvolvimento mais rápido e simples.

Com os conceitos de orientação a objetos revisados, iniciamos os diagramas de classes, os quais são elementos importantes para o processo de modelagem de sistemas.

■ DIAGRAMA DE CLASSES

O diagrama de classes da UML é um diagrama estrutural, que tem como objetivo principal ilustrar graficamente a estrutura do software, em níveis mais e menos abrangentes. Além disso, o diagrama de classes mostra como se dá a interligação entre os componentes da estrutura do sistema (UML-DIAGRAMS, 2016).

Para compreender o diagrama de classes, é preciso entender os relacionamentos entre as classes que poderão ser representadas no diagrama de classes e objetos.

■ RELACIONAMENTO DO TIPO ENCAPSULAMENTO

Um importante tipo de relacionamento é o encapsulamento, que tem como objetivo garantir a proteção e o controle de acesso dos atributos e métodos de uma classe, estabelecendo diferentes níveis de acesso aos usuários de uma classe para que apenas usuários autorizados tenham visualização, ação, modificação de algum atributo ou método da classe (DEITEL; DEITEL, 2016).

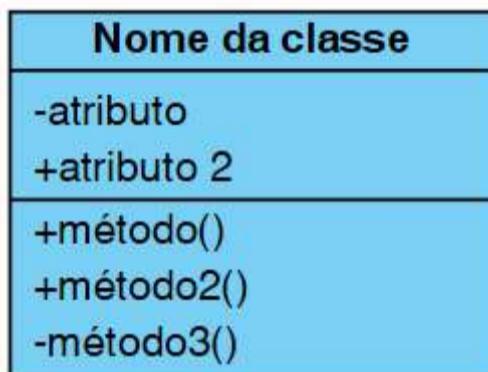
Para encapsular informações em uma classe, são utilizados os operadores de escopo em atributos e métodos, que são:

- **Public:** nível de acesso mais permissivo. O método ou atributo da classe é livre para qualquer usuário manusear. É utilizada a notação (+) para representar atributos e métodos públicos.
- **Private:** nível de acesso mais protegido (restritivo). Apenas usuários definidos têm acesso. É utilizada a notação (-) para indicar atributos e métodos privados.
- **Protected:** nível intermediário (entre *public* e *private*). Permite acesso externo restrito e os atributos e métodos podem ser públicos

dentro de uma classe específica. Envolve conceitos de herança abordados a seguir. A notação (#) é utilizada para indicar atributos e métodos protegidos.

Em geral, um diagrama de classes possui três informações: o **nome** da classe, os **atributos** (características) e os **métodos** (comportamento, operações). Os atributos são mostrados na primeira partição e os métodos na segunda, como mostrado na Figura 2.13 a seguir.

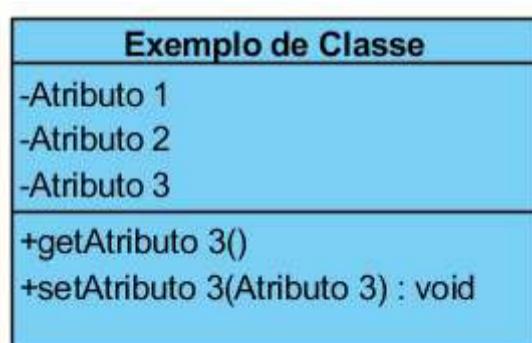
Figura 2.13 | Notação diagrama de classes



Fonte: elaborada pelo autor.

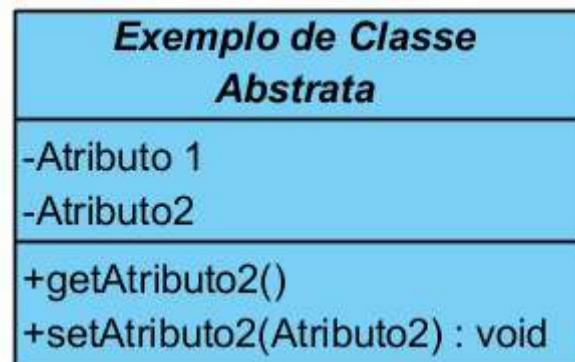
Os diagramas de classes UML apresentam dois tipos básicos de classe: as **classes comuns** (Figura 2.14) e as **classes abstratas** (Figura 2.15). As comuns possuem o nome da classe, os atributos e os métodos separados por um divisor; já as abstratas são representadas da mesma maneira, porém com o nome da classe em itálico (Figura 2.15). Cada classe possui seus próprios atributos e métodos.

Figura 2.14 | Exemplo de classe



Fonte: elaborada pelo autor.

Figura 2.15 | Exemplo de classe abstrata



Fonte: elaborada pelo autor.

Outro componente presente nos diagramas de classe são as **interfaces**. Elas apresentam um conjunto de operações e elementos públicos, que devem ser implementados por outras classes (UML-DIAGRAMS, 2016). Esse procedimento faz com que as informações de implementação fiquem protegidas em outras classes privadas ou protegidas enquanto a interface é acessada. Nos diagramas de classes UML, as interfaces são representadas de maneira similar às classes, porém possuem um identificador, como apresentado na Figura 2.16.

Figura 2.16 | Exemplo de interface



Fonte: elaborada pelo autor.

Como uma classe abstrata ou interface não instanciam objetos, é necessário que alguma outra classe implemente o que está descrito na interface ou que alguma classe herde os atributos e métodos da classe abstrata e implemente seu comportamento.

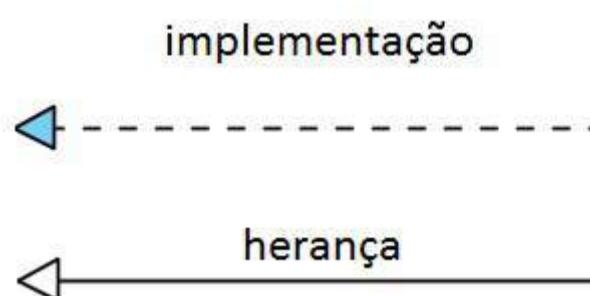
| TIPOS DE RELACIONAMENTO

Outro ponto importante são os tipos de relacionamentos entre as classes. A **generalização** (herança) é representada por uma seta contínua, que sai da subclasse e vai em direção à superclasse, enquanto

que a **implementação**, também chamada de realização, é representada por uma seta pontilhada, que sai da classe que implementa e aponta para a classe implementada. Ambas estão representadas na Figura 2.17.

0

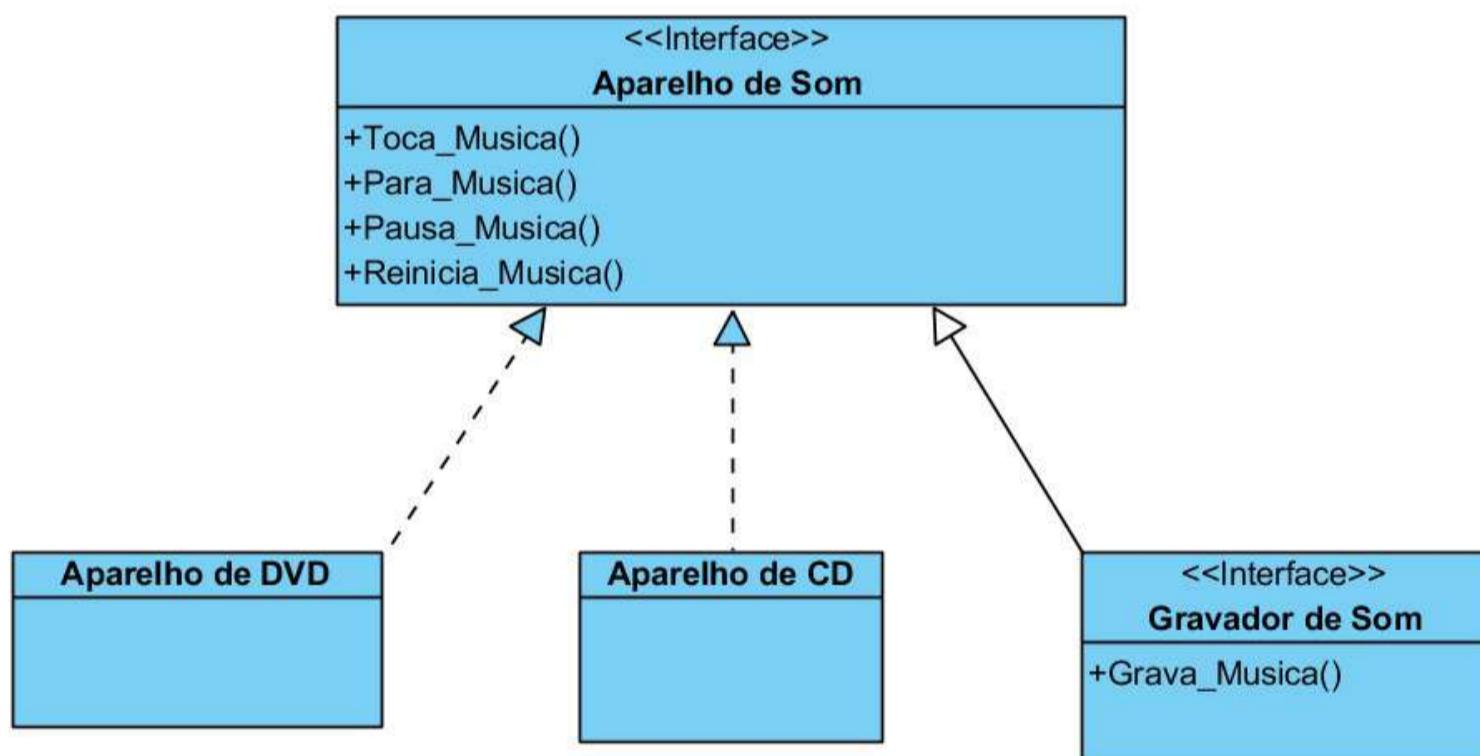
Figura 2.17 | Relacionamentos de implementação e herança



Fonte: elaborada pelo autor.

Para melhor compreensão, observe o exemplo de um diagrama de classes simples apresentado na Figura 2.18.

Figura 2.18 | Diagrama de classes para um sistema de música



Fonte: elaborada pelo autor.

Na Figura 2.18, é apresentado um diagrama de classes para um sistema de música. Nela existe uma interface que apresenta as funções básicas que devem estar disponíveis para o sistema: tocar, parar, pausar e reiniciar uma música. As classes aparelho de DVD e aparelho de CD implementam a interface, cada uma para uma mídia diferente. A herança nesse caso é representada entre a classe aparelho de som e gravador de som, já que se entende um gravador como um aparelho de

Ver anotações

som que possui a função de gravar. Observe que nenhuma classe implementa o gravador ainda, porém iremos adicionar funcionalidades a esse diagrama ao longo da seção.

É importante destacar que existem diferentes perspectivas, as quais estão relacionadas à quantidade de detalhes e os tipos de relacionamentos entre as classes. As três perspectivas que podem ser apresentadas são as seguintes:

- **Conceitual:** que define um relacionamento entre entidades conceituais, conceito no domínio.
- **Especificação:** que define funções/responsabilidades entre duas classes. Existem métodos que tratam essas operações/responsabilidades. Esta perspectiva é utilizada para entendimento da arquitetura em nível de interface.
- **Implementação:** é a mais completa, pois possui diversos detalhes, como visibilidade de atributos e métodos, parâmetros, tipos de atributos e retornos. Além disso, há a possibilidade de visualizar a navegabilidade (seta no fim do relacionamento) na representação gráfica e as responsabilidades.

É importante conhecer as todas as perspectivas que podem ser utilizadas em diferentes etapas do desenvolvimento do software mesmo sendo a implementação a perspectiva que envolve mais elementos e mais detalhes.

RELACIONAMENTO DE MULTIPLICIDADE

Ainda sobre os diagramas de classes, existem outros relacionamentos que devem ser descritos. O relacionamento “papel” descreve a ligação entre as classes, facilitando o entendimento. Além disso, é importante destacar o relacionamento de multiplicidade entre as classes, que diz respeito à quantidade de instâncias de uma classe associada com um ou mais instâncias de outra classe.

- ***n..m*** - significa *n* para *m* instâncias entre classes, ou seja, *n* classes A se relacionam com *m* classes B.
- ***n..**** - quando se utiliza o asterisco (em qualquer um dos lados da expressão *n..m*) indica-se “*muitos*”, ou seja, muitas instâncias da classe.
- **1** - quando se utiliza o número 1, quer-se dizer exatamente uma instância associada.
- **1..*** - esta relação, em que temos o número *1..**, apresenta o conceito de um ou mais instâncias que podem estar relacionadas.

As notações dos tipos de relacionamento multiplicidade são ilustrados na Figura 2.19 a seguir.

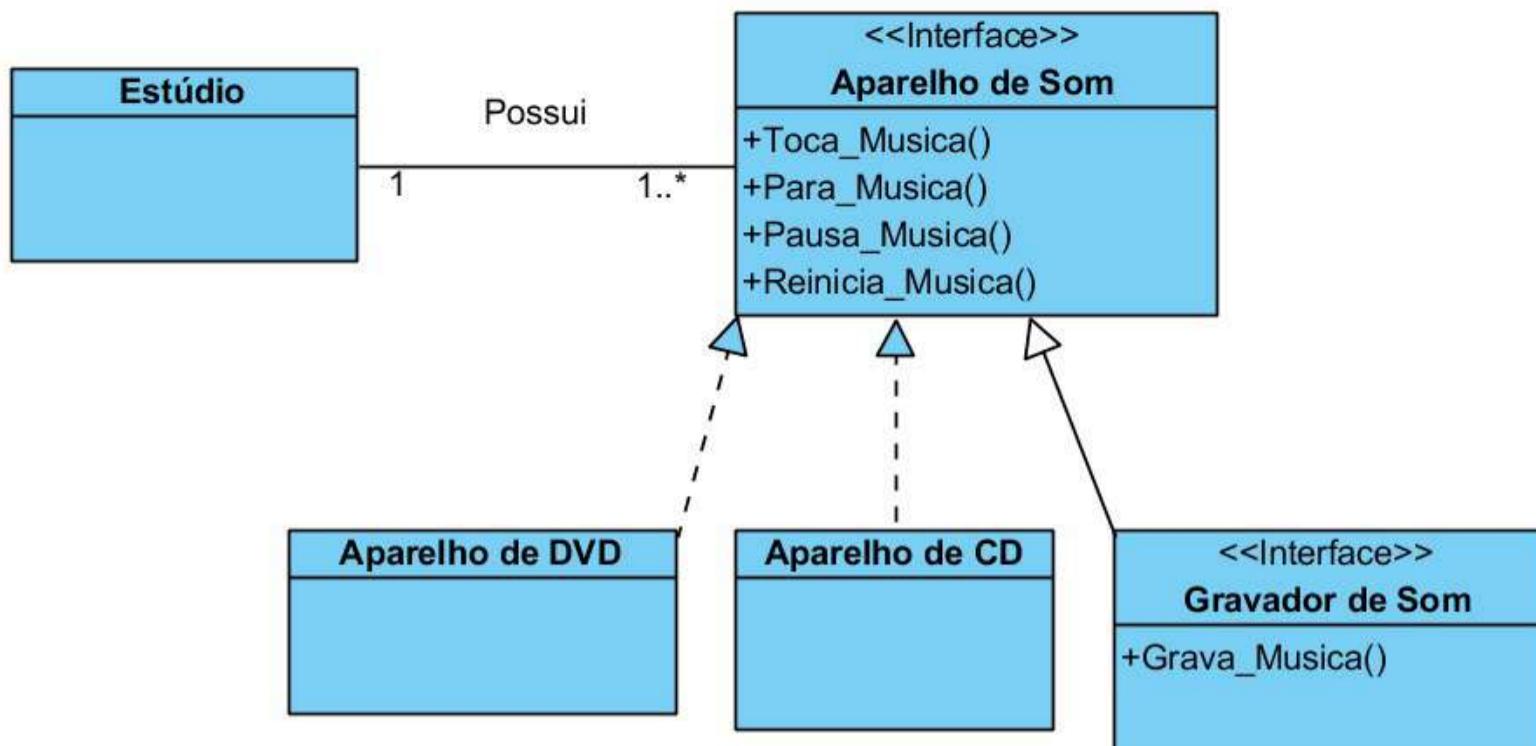
Figura 2.19 | Notação de multiplicidade



Fonte: adaptada de Visual Paradigm (c2020).

Para exemplificar a multiplicidade, o diagrama do sistema de som, apresentado anteriormente, foi modificado e é apresentado na Figura 2.20. Neste caso foi adicionada a classe estúdio e os relacionamentos de multiplicidade, que representam que um estúdio (1) possui um ou mais (1..*) aparelhos de som.

Figura 2.20 | Exemplo de diagrama de classe com relacionamento multiplicidade



Fonte: elaborada pelo autor.

Outros dois relacionamentos que podem ser representados em diagramas de classe são: **agregação** e **composição**.

A agregação é considerada um caso especial de associação; ela representa o fato de que uma das classes do relacionamento é uma parte de ou está contida em outra classe. A associação agregação é do tipo todo-parte quando um objeto é parte de outro; porém, é preciso salientar que a parte existe independentemente do todo. Sendo assim, é considerada como um relacionamento fraco. Por exemplo, se uma classe 2 é parte da classe 1, ambas as classes existem separadamente.

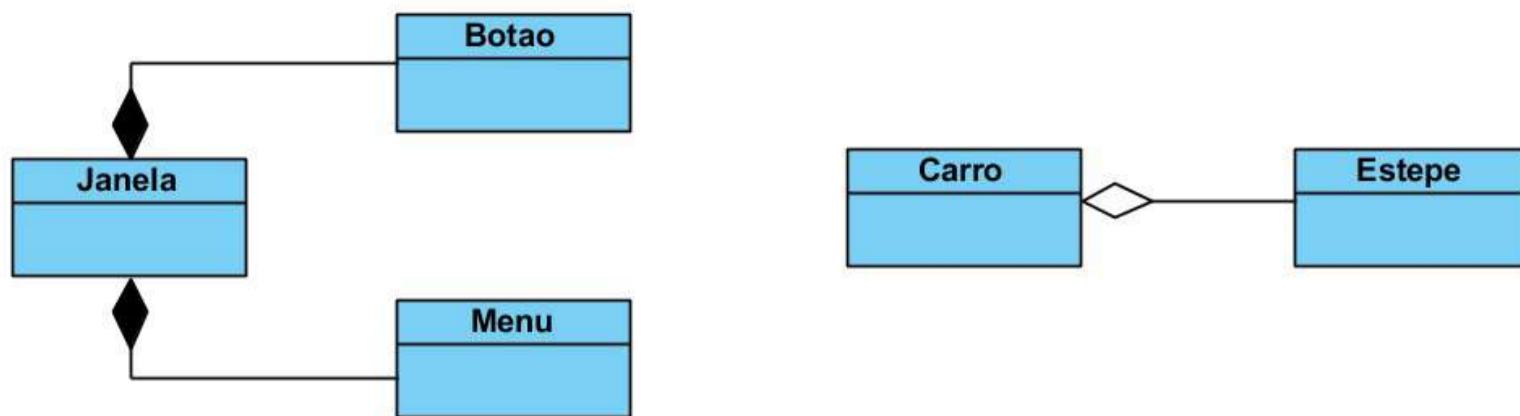
A composição é uma variação da agregação que apresenta um vínculo mais forte entre as associações. No caso, objetos-parte devem obrigatoriamente pertencer ao objeto-todo. Logo, na composição, o todo não existe sem as partes e as partes não existem sem o todo.

A Figura 2.21 ilustra um exemplo com os relacionamentos agregação e composição. A imagem à esquerda é um exemplo de composição: o menu e os botões não existem sem as janelas para acomodá-los. Portanto, a composição implica uma relação parte-todo configurando

uma relação mais forte no sentido de que uma classe não existe sem a outra. A notação da composição é um losango preenchido ao lado do todo.

No caso da agregação, relação à direita na Figura 2.21, é apresentado um carro e seu estepe. Um carro pode existir sem o estepe ainda que possa apresentar problemas ao usuário caso um pneu fure. O relacionamento de agregação é representado por um losango sem preenchimento na extremidade da classe que contém os objetos-todo.

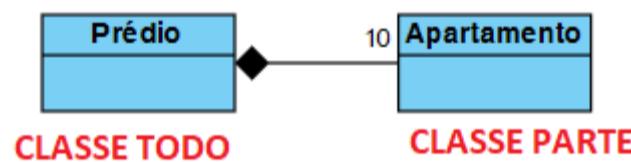
Figura 2.21 | Exemplos de composição e agregação



Fonte: elaborada pelo autor.

Vamos agora apresentar um outro exemplo de composição que envolve multiplicidade. Na Figura 2.22, temos um exemplo de composição entre prédio e apartamentos. Assim, temos duas classes: prédio e apartamento. O prédio é considerado o todo, ou agregada, e a classe apartamento é parte da classe prédio.

Figura 2.22 | Exemplos de composição e agregação



Fonte: elaborada pelo autor.

No exemplo, pode-se observar que os apartamentos apenas existem se o prédio existe também. Já a multiplicidade apresentada no diagrama representa que é necessário, obrigatoriamente, um prédio possuir dez apartamentos.

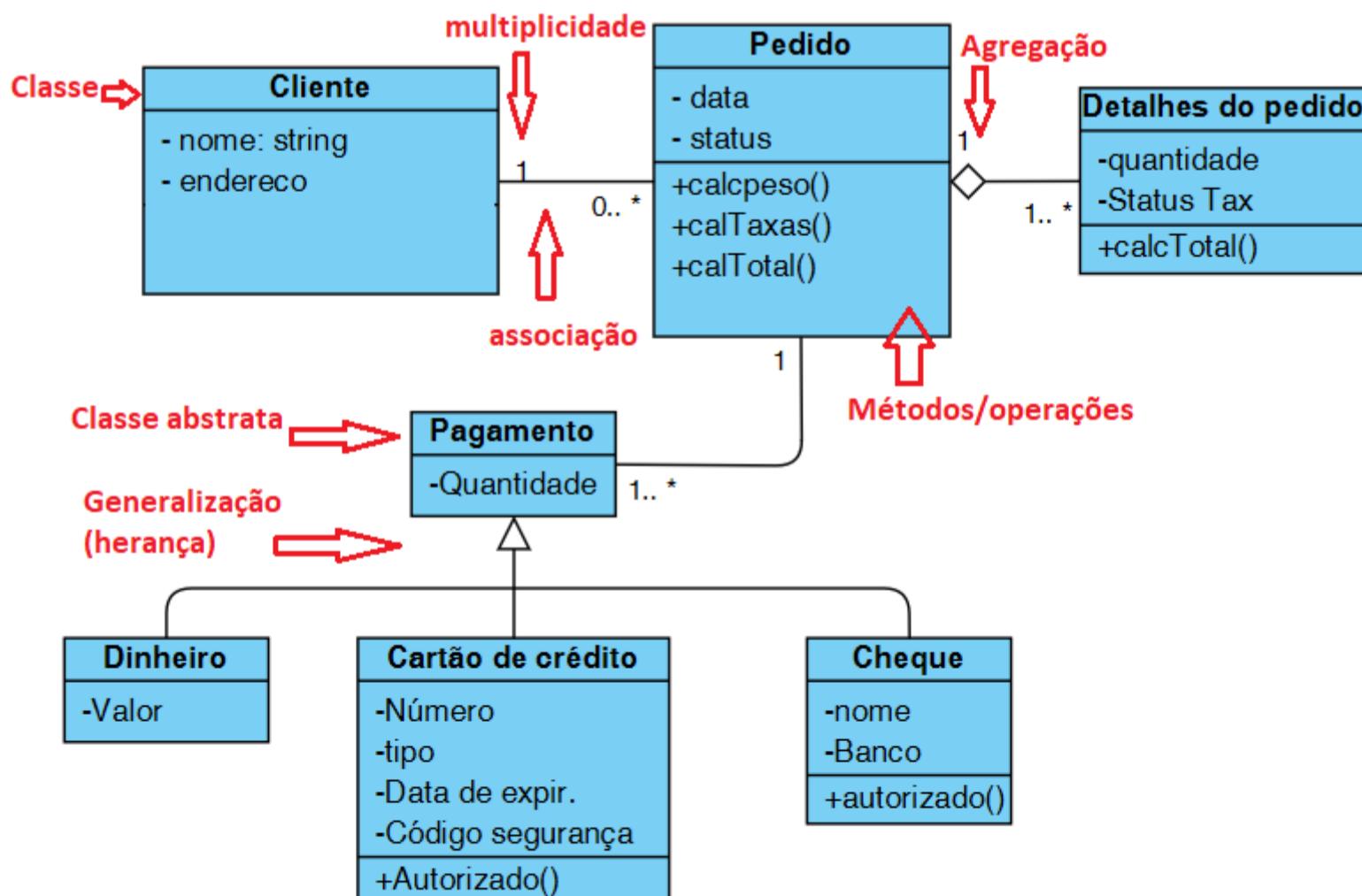
ASSIMILE

Importa destacar os conceitos de agregação e composição relacionados às necessidades de representação do diagrama de classes. Quando, na modelagem, existe uma relação de todo-parte entre duas classes, é necessário definir se o relacionamento é do tipo agregação ou composição. Na agregação, o todo e a parte existem independentemente e, no caso da composição, a parte depende do todo para existir.

Algumas palavras utilizadas para reconhecer uma agregação são as seguintes: "consiste em", "contém", "é parte de". É importante lembrar que essas relações são abstrações, ou seja, quando implementamos uma relação de composição, não necessariamente seria impossível instanciar um objeto da parte sem ter instanciado o todo, porém conceitualmente isso não deve ser feito no código.

Para sintetizar os conceitos apresentados sobre diagrama de classes e os seus relacionamentos, a seguir é apresentado um exemplo de aplicação de um sistema de compras. A Figura 2.23 ilustra esse sistema no qual temos as classes cliente, pedido, detalhes do pedido, a classe abstrata pagamento e as classes dinheiro, cartão de crédito e cheque. Pode-se observar que temos as associações com multiplicidades, em que um cliente faz nenhum ou mais pedidos. Para cada pedido deve-se realizar os pagamentos, ação que está associada ao pedido, pago através de cartão, dinheiro ou cheque. Observe que temos três subclasses: dinheiro, cartão de crédito e cheque herdadas da superclasse chamada pagamento.

Figura 2.23 | Exemplo de diagrama de classes de um sistema de compras



MODIFICADORES DE ACESSO

Para encerrar o tópico sobre os elementos de um diagrama de classes, outro conceito importante é o dos modificadores de acesso, que foram introduzidos anteriormente nesta seção quando tratamos dos operadores de escopo em encapsulamento, na revisão de orientação a objetos. Agora vamos tratar deles no contexto de aplicação em um diagrama de classes.

Os modificadores de acesso são modelos de visibilidade para acesso às classes, atributos e métodos.

Em UML é possível representar itens com visibilidade *public*, *protected*, *private* e *package* (que apenas é visível em um determinado *namespace*). A Figura 2.24 apresenta um exemplo de classe com métodos de visibilidade diferentes.

Figura 2.24 | Classe com exemplos de visibilidade



Fonte: elaborada pelo autor.

A classe da Figura 2.24 apresenta um método configurado com cada tipo de visibilidade. O primeiro é *public*, indicado com um '+'; o segundo é do tipo *package*, indicado por um '~'; o terceiro é o *protected*, indicado por um '#' e o último é o *private*, indicado por um '-'. Portanto, ao analisar um diagrama, é necessário considerar como os atributos e métodos estão indicados quanto a sua visibilidade.

Sabe-se que as classes são “modelos” que podem ser instanciados em objetos. A modelagem normalmente ocorre criando-se as classes necessárias e todos os relacionamentos existentes. Porém, existem alguns problemas que podem aparecer ao se fazer a modelagem das classes. Caso não sejam analisados os objetos instanciados e suas relações, as classes podem apresentar problemas de implementação, que só irão aparecer quando o programa for executado. Você consegue imaginar esse tipo de problema? Tente pensar em algum que não apareceria em uma implementação de classes, mas que causaria erros na execução do programa ao instanciar os objetos e ao executar o programa.

0

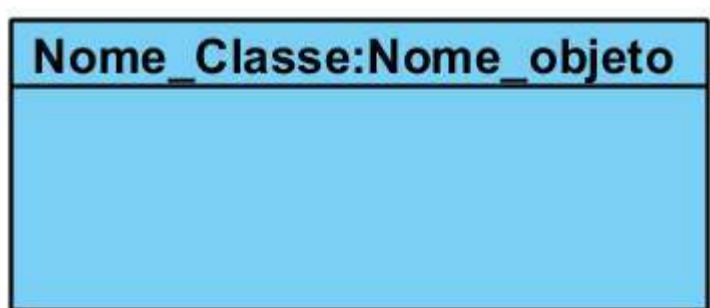
Ver anotações

| DIAGRAMA DE OBJETOS

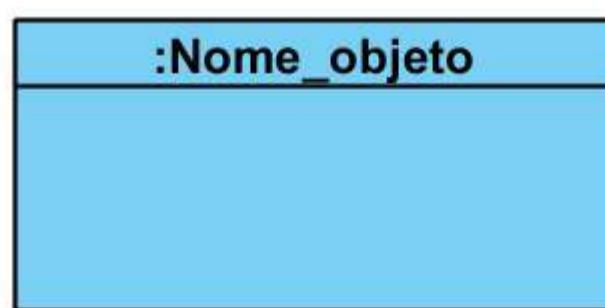
Além do diagrama de classes, temos o diagrama de objetos; este é uma variação daquele, com uma notação semelhante à do diagrama de classes. O diagrama de objetos é utilizado para representar os objetos instanciados de uma classe. Ele utiliza notação de objetos para a própria construção.

Esse diagrama tem por objetivo representar os objetos e os relacionamentos entre eles e nada mais é do que um grafo de instâncias, incluindo os objetos e os valores de seus atributos (UML-DIAGRAMS, 2016). As representações de objetos são feitas de forma similar às de classes, porém com a diferença de que eles só possuem atributos, e o nome irá indicar a classe da qual foram instanciados ou então apresentará uma instância de qualquer classe. Esses dois exemplos são apresentados na Figura 2.25.

Figura 2.25 | Tipos de objetos do diagrama de objetos



Objeto de classe específica



Objeto de classe genérica

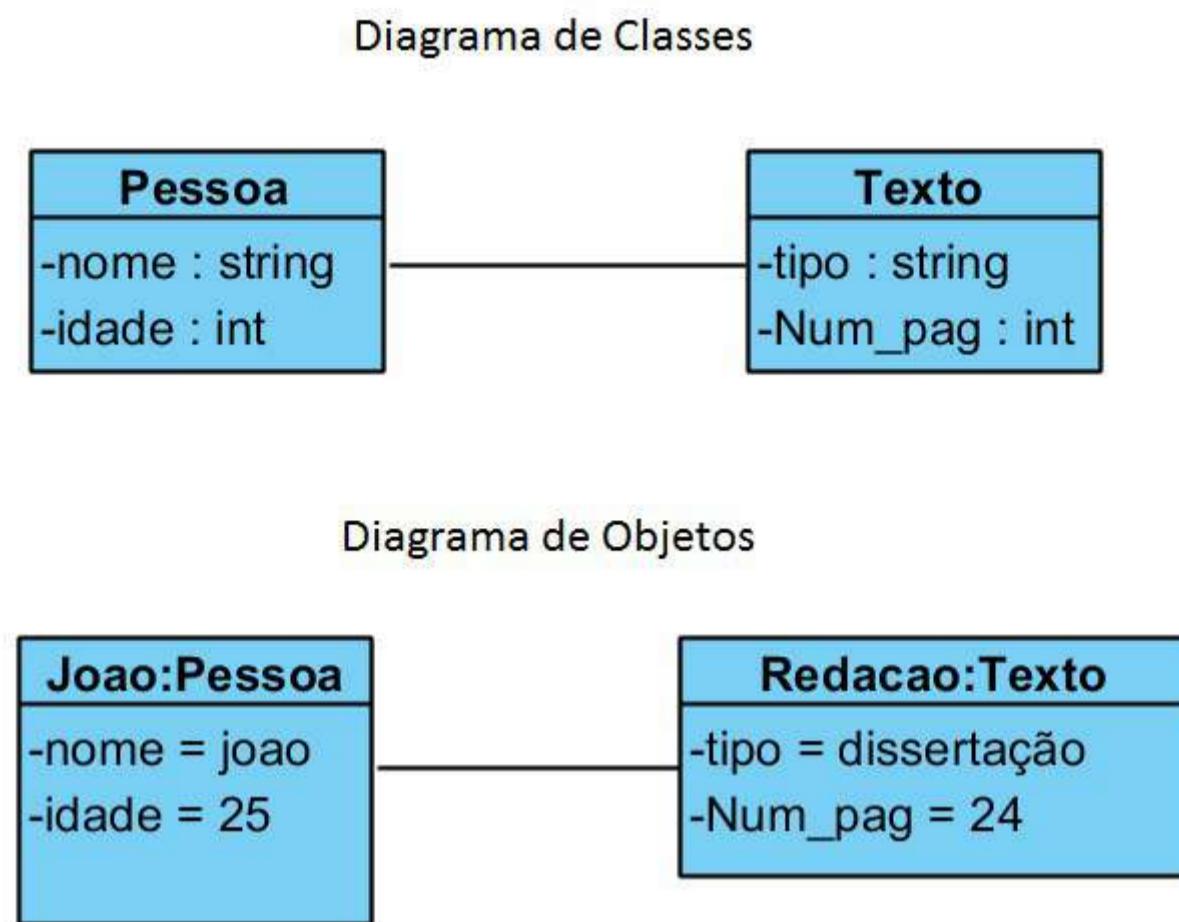
Fonte: elaborada pelo autor.

Além dos objetos, são representadas as associações e os valores de seus atributos.

É importante ressaltar que um diagrama de objetos deve ser gerado a partir de um diagrama de classes.

A Figura 2.26 ilustra um exemplo de diagrama de classes e um possível diagrama de objetos gerado a partir dele. Temos as classes pessoa e texto. A partir do diagrama de classes, geramos o diagrama de objetos.

Figura 2.26 | Exemplo de diagrama de classes e objetos



Fonte: elaborada pelo autor.

Observe que são apresentados apenas os nomes e os atributos no diagrama de objetos. Os valores dos objetos, obedecendo ao tipo de variável (int), também são apresentados. Nesse caso é possível entender como gerar um diagrama de objetos a partir de um diagrama de classes.

Caro aluno, nesta seção foram abordados alguns conceitos de orientação a objetos para que você conhecesse os diagramas de classes e objetos, bem como os principais componentes e relacionamentos. Além disso, tratou-se dos diagramas de objetos a partir de um determinado diagrama de classes. Agora você pode aplicar os seus conhecimentos para elaborar um diagrama de classes e de objetos para algum exemplo de aplicação do seu dia a dia. Então vamos lá? Bons estudos e mãos à obra!

REFERÊNCIAS

BELL, D. Fundamentos básicos de UML: O diagrama de classes: Uma introdução aos diagramas de estrutura em UML 2. IBM Developer Works, 2016. Disponível em: <https://bit.ly/2DccZMC>. Acesso em: 6 jul.

2020.

DEITEL, P.; DEITEL, H. Java: como programar. 10. ed. Campinas: Pearson Universidades, 2016. 968p.

RUBAUGH J.; JACOBSON, I.; BOOCH, G. The Unified Modeling Language Reference Manual. 2. ed. [S.I.]: Pearson Higher Education, 2004.

SOMMERVILLE, I. Engenharia de Software. 9. ed. Campinas: Pearson Universidades, 2011.

UML-DIAGRAMS. The Unified Modeling Language, [S.I.], 2016. Disponível em: <https://bit.ly/3f7qM41>. Acesso em: 6 jul. 2020.

UNHELKAR, B. Software engineering with UML. [S.I.]: CRC Press, 2018.

VISUAL PARADIGM. UML Class Diagram Tutorial. Visual Paradigm, Hong Kong, c2020. Disponível em: <https://bit.ly/2QBc1N4> . Acesso em: 6 ago. 2020.

FOCO NO MERCADO DE TRABALHO

MODELAGEM DE CLASSES

Maurício Acconcia Dias

Ver anotações

MODELAGEM DE UM SISTEMA

A modelagem da estrutura de um sistema deve ser iniciada com classes mais simples e especificadas em subclasses herdeiras das características gerais, bem como devem ser definidos os relacionamentos entre todas as classes.



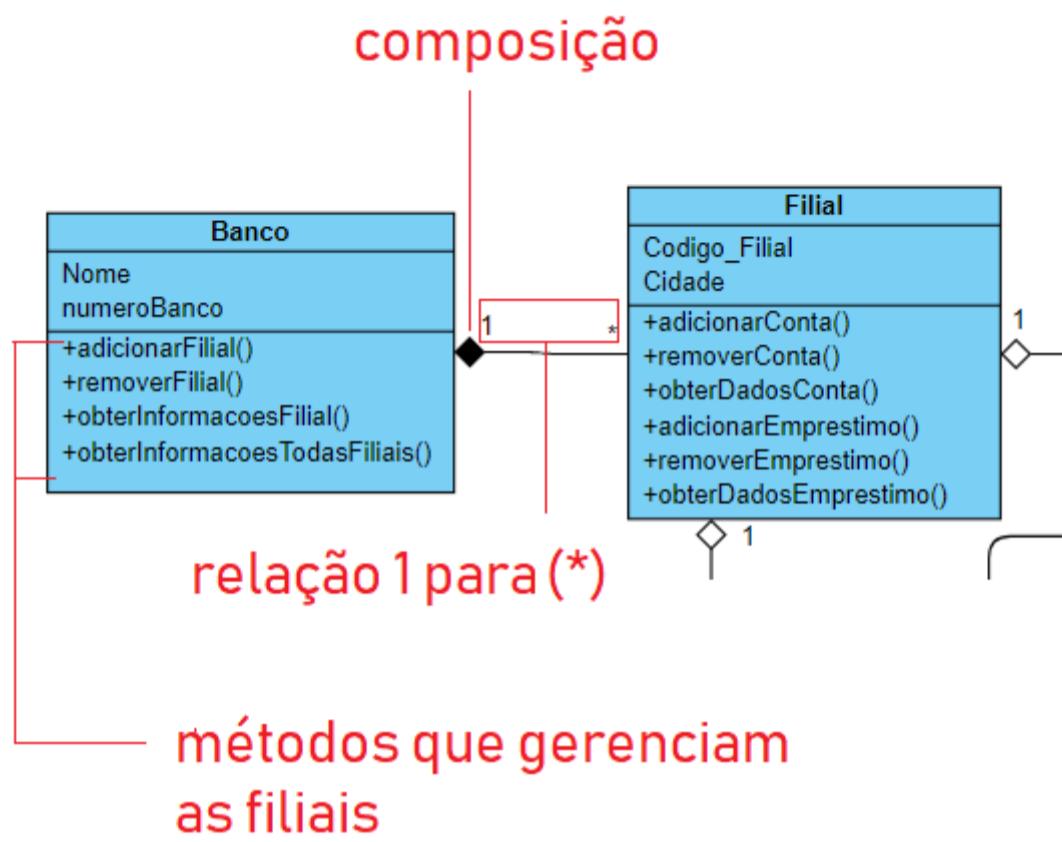
Fonte: Shutterstock.

Deseja ouvir este material?

Áudio disponível no material digital.

Este diagrama de classes está representando a relação entre um banco e suas filiais. É possível notar as seguintes estruturas e relações:

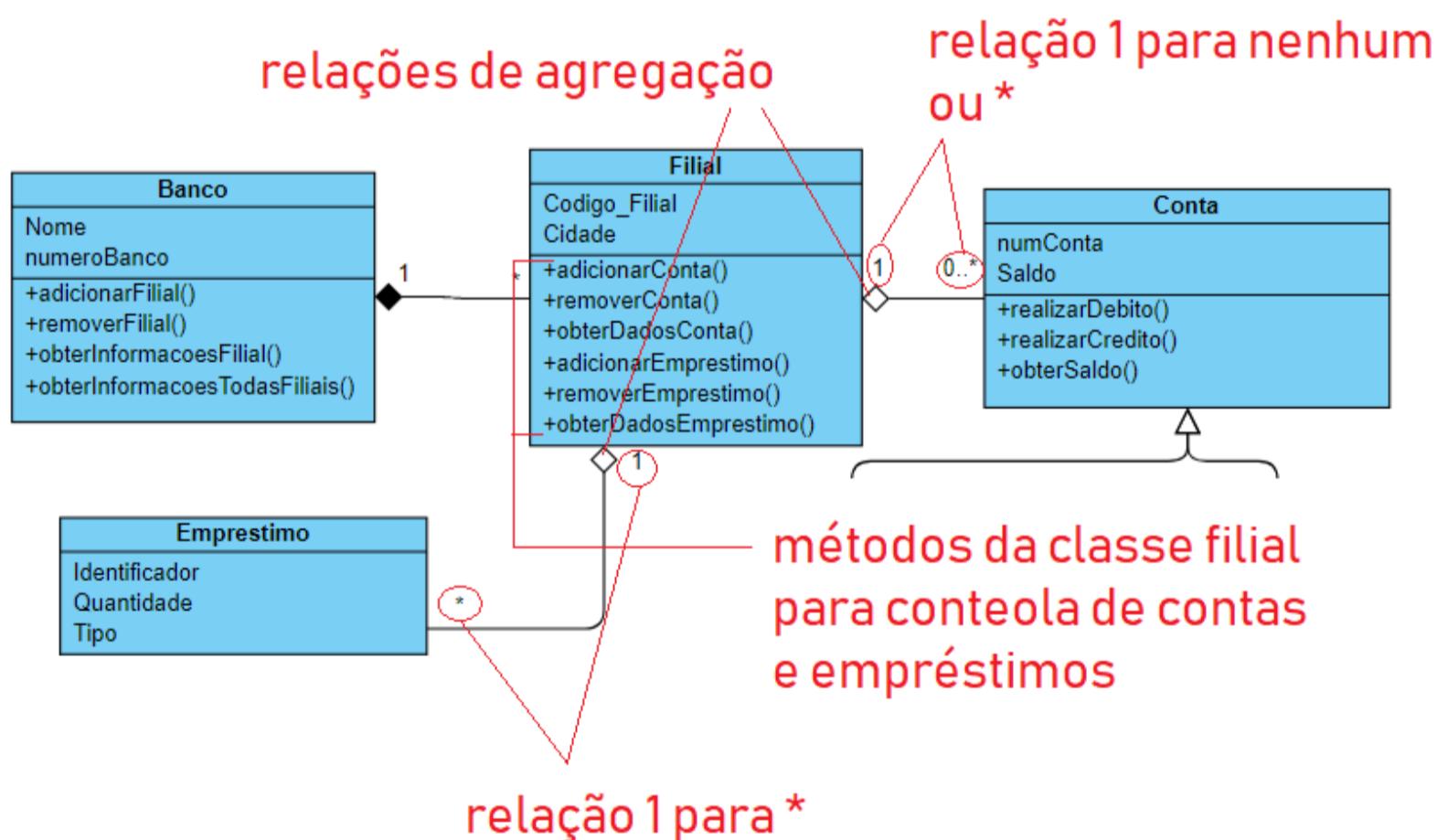
Figura 2.27 | Especificação da classe banco



Fonte: elaborada pelo autor.

- O banco possui um conjunto de filiais em uma relação de composição, ou seja, as filiais não existem sem a matriz do banco. Além disso, um banco pode ter um número não definido de filiais (definido pela relação 1 para *).
- A classe banco gerencia as filiais com funções de adicionar, remover, obter dados de uma filial específica ou obter dados de todas as filiais.
- A classe filial possui um código e uma cidade. Em uma filial é possível adicionar uma conta corrente, remover uma conta corrente ou obter informações dessa conta. A filial pode fazer empréstimos, remover os empréstimos e consultá-los. Uma filial também possui uma relação de agregação com as contas dos clientes, ou seja, elas podem existir mesmo que não tenham nenhuma conta.

Figura 2.28 | Especificação da classe filial



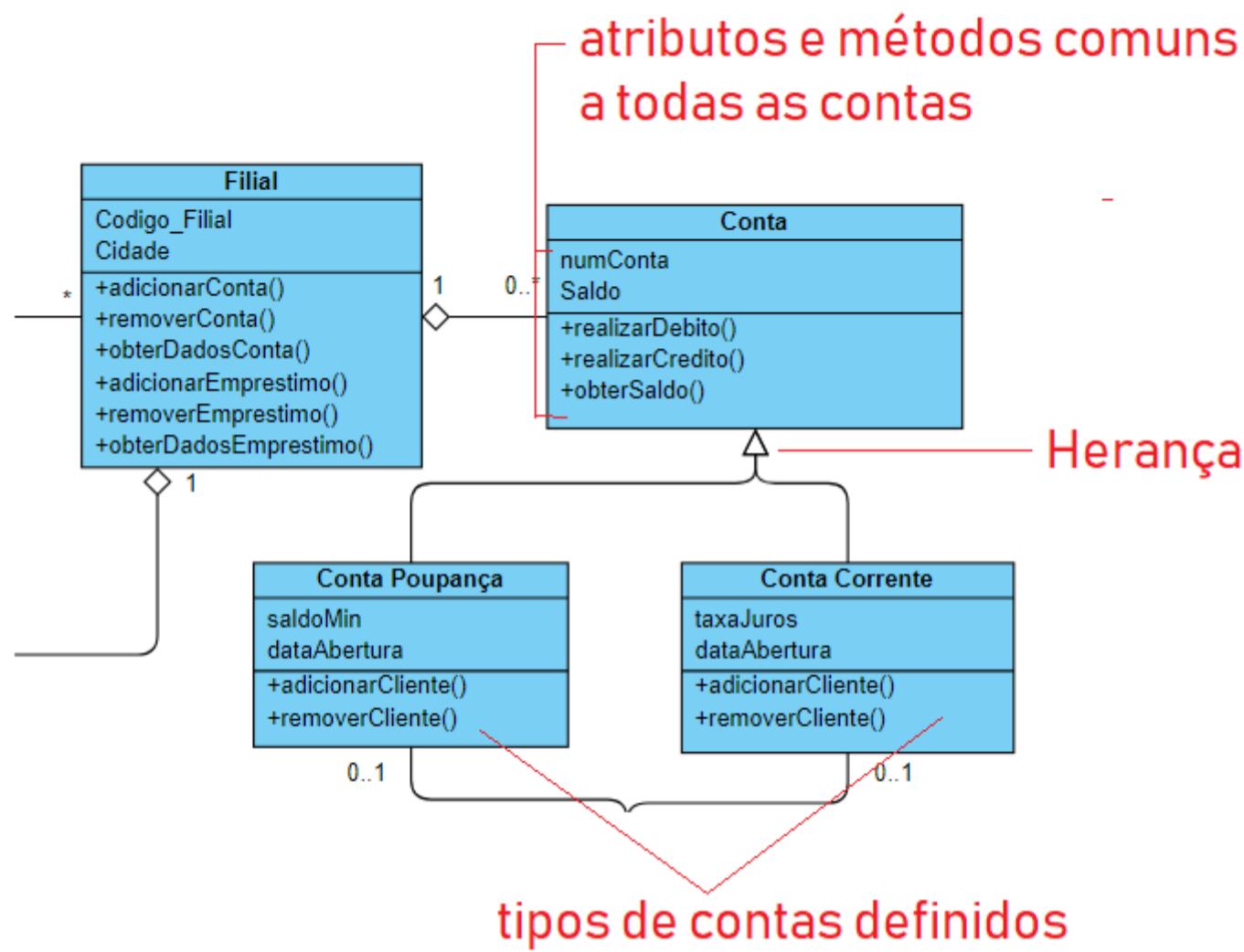
Fonte: elaborada pelo autor.

- Uma conta possui um número e um saldo como seus atributos. É possível realizar transações de débito e crédito além de consultar o saldo.
- Dois tipos de conta são definidos por herança: a conta poupança e a conta corrente. A conta poupança tem como atributos um saldo

mínimo e sua data de abertura. Nessa conta é possível adicionar ou remover um cliente. Já a conta corrente possui, relacionada a ela, uma taxa de juros e uma data de abertura como atributos. Os métodos permitem adicionar e remover um cliente.

o

Figura 2.29 | Especificação da classe conta



Fonte: elaborada pelo autor.

- Um cliente pode ter 0 ou 1 conta de cada tipo, sendo que possui como atributos um número de identificação, nome, endereço e telefone.
- A última classe é a classe de empréstimo. Os atributos de um empréstimo são um número, o tipo e a quantia emprestada. Em um empréstimo é possível adicionar um cliente, obter os pagamentos realizados e o estado atual do empréstimo. No caso do empréstimo um cliente pode ter de 0 a infinitos empréstimos. Uma filial pode também conceder infinitos empréstimos, porém sua relação é de agregação já que pode ser que exista uma filial sem que nenhum empréstimo tenha sido realizado.

NÃO PODE FALTAR

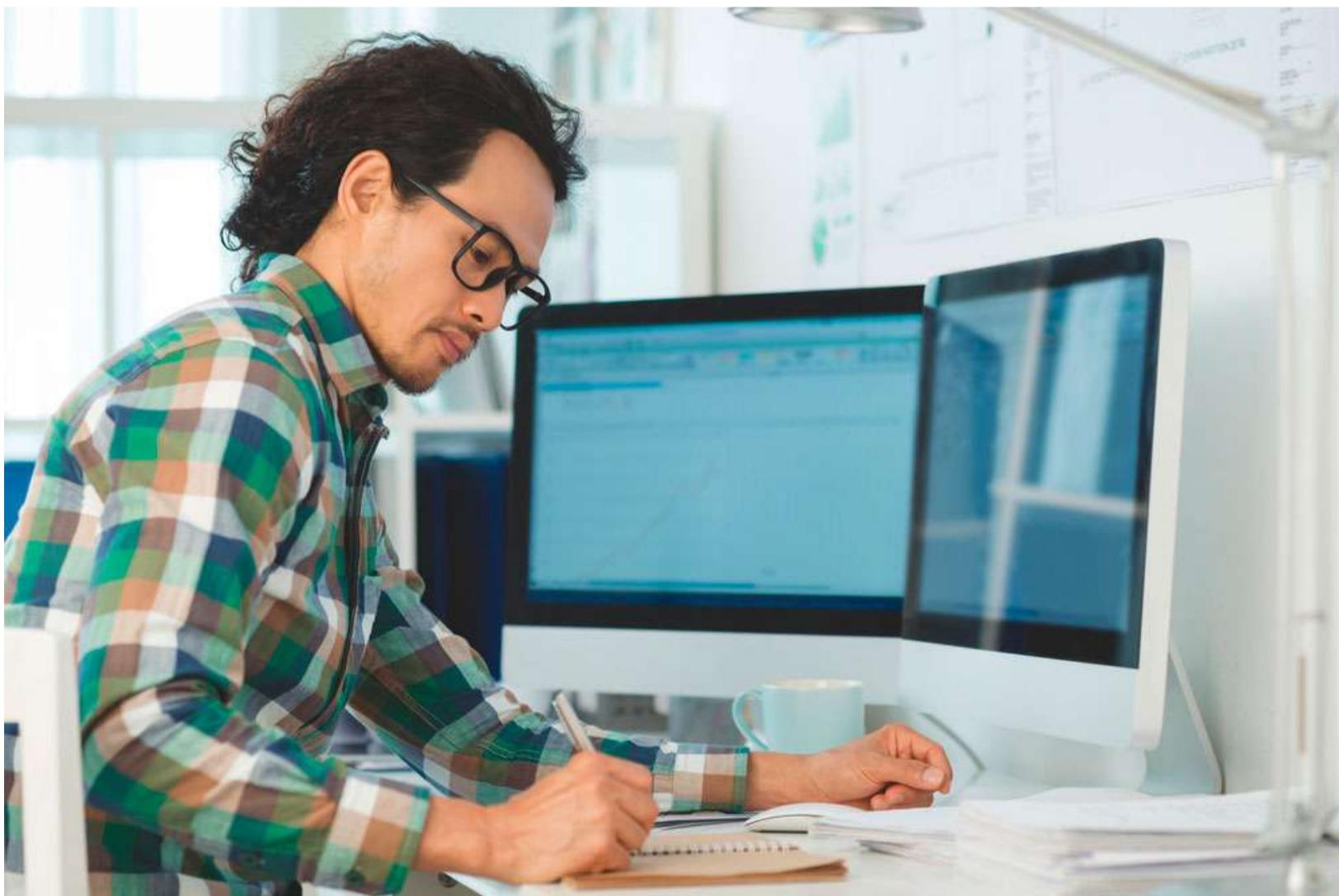
MODELAGEM DE ATIVIDADES

Maurício Acconcia Dias

Ver anotações

DIAGRAMA DE ATIVIDADES

Permite que as atividades e os fluxos de controle do sistema sejam representados em um diagrama e, posteriormente, possam auxiliar no desenvolvimento do sistema.



Fonte: Shutterstock.

Deseja ouvir este material?

Áudio disponível no material digital.

Seja bem-vindo à última seção desta unidade. Neste espaço, o diagrama UML abordado é o diagrama de atividades. A partir dele, pode-se representar o conjunto de atividades a serem executadas em um processo do sistema, assim como os fluxos de controles.

Como exemplo, imagine que você está em um terminal de um caixa eletrônico de um banco. Neste terminal, é necessário que exista um software capaz de gerenciar todas as operações possíveis em relação a saques, depósitos, transferência e gerenciamento da conta em geral.

Pense sobre o que deve ser envolvido no processo: você, enquanto cliente, executará algumas ações; o computador do terminal será responsável por interpretar essas ações e responder de acordo com o que for solicitado; por fim, um servidor do banco informará os resultados das operações solicitadas, se obtiveram sucesso ou não, e conferirá os dados finais da conta, para que possa salvar essas informações.

Imagine, agora, todas as atividades que você pode realizar nesta situação: iniciar o acesso, digitar a senha, escolher a operação desejada, confirmar sua identidade (geralmente, com mais de um método, por exemplo, senha numérica e biometria), realizar a operação provendo as informações necessárias para o sistema, pegar o comprovante da operação gerado após a conclusão e encerrar seu acesso ao sistema, levando seu cartão com você. Essas são as atividades realizadas por você, porém, enquanto cada operação é processada, um acesso ao servidor do banco é feito para consulta de saldo e outras informações da sua conta. Esse acesso é efetuado pelo programa, o qual é executado no terminal de autoatendimento.

As ações descritas são modeladas pelo diagrama de atividades, que cria um fluxograma com início e final definidos para cada sequência de ações.

O diagrama de atividades é importante porque permite que as atividades e os fluxos de controle do sistema sejam representados em um diagrama e, posteriormente, podem auxiliar no desenvolvimento do sistema.

Nesta seção, você aprenderá os principais conceitos do diagrama de atividades, seus fluxos de controle, componentes, partições de atividades e, principalmente, como elaborar um diagrama a partir de um problema.

Você foi contratado como desenvolvedor de software em uma empresa que pudesse utilizar seus conhecimentos de UML e programação. Ao se candidatar às vagas que encontrou, foi selecionado e admitido para fazer parte do time de desenvolvimento de um banco.

Ao estudar as principais atribuições de um desenvolvedor em uma equipe de um banco, descobriu que existem diversos tipos de sistemas de informação sendo executados diariamente em diferentes partes de uma agência bancária. Além disso, existem os softwares de gerência, os quais são executados em servidores e são responsáveis pela atualização de informações dos movimentos diários de cada agência.

Para o início de seu trabalho, é necessário que você passe pelas fases iniciais do desenvolvimento do sistema, para, então, estar habilitado a contribuir em outros projetos. Os sistemas mais simples dentre os existentes no banco são os que envolvem o caixa eletrônico. Nele existem diversas operações que podem ser realizadas, dentre elas, a mais simples é a operação de sacar dinheiro com o cartão de crédito.

Você foi selecionado para modelar o diagrama de atividades do saque de dinheiro com cartão de crédito. Imagine uma situação em que uma pessoa vai ao caixa eletrônico de posse de seu cartão para sacar dinheiro. Pense em todas as operações que ocorrem com o cliente, as consultas ao servidor e as respostas da máquina. Para guiar o desenvolvimento, alguns requisitos foram apresentados:

- O diagrama deve conter as *swimlanes* do cliente, da máquina ATM e do servidor do banco.
- A senha é digitada apenas uma vez para verificação.
- Após o saque, a máquina mostra o saldo na tela.
- Ao finalizar a operação, a máquina libera o cartão, e a atividade é encerrada.

Pronto para aprender como transformar a ideia de fluxo de ações em um diagrama? Bons estudos!

CONCEITO-CHAVE

DIAGRAMA DE ATIVIDADES

Dentre os diagramas da linguagem UML, o diagrama de atividades desempenha um papel fundamental por mostrar aspectos comportamentais na modelagem de sistemas. Basicamente, ele consiste em um gráfico na forma de fluxo que mostra as características dinâmicas do sistema, podendo ser visualizado o fluxo de ações que serão realizadas em uma determinada atividade relacionada ao sistema que será desenvolvido.

Esta perspectiva sobre o sistema é importante ser considerada, pois, muitas vezes, o produto de software desenvolvido pode apresentar melhorias no fluxo de ações das atividades necessárias em um sistema. Além disso, a sequência das atividades pode não ter sido implementada de forma otimizada e poderia ter melhor performance se realizada de outra forma, por exemplo, concorrentemente. Nesse sentido, o diagrama de atividades, em conjunto com outros diagramas UML, torna-se uma alternativa capaz de evitar esse tipo de problema, podendo auxiliar no desenvolvimento e na melhoria da qualidade do produto de software.

Dentre os **principais objetivos** do diagrama de atividades, podemos ressaltar: torna mais claro o fluxo de controle em operações complexas de casos de uso do sistema; mostra, com relação à lógica de execução,

como uma tarefa deve ser feita e pode decompor atividades maiores em subatividades, facilitando o entendimento do sistema e suas operações necessárias.

Algumas de suas **características** também são interessantes de se discutir antes de mostrarmos as partes do diagrama. Podemos entender o diagrama de atividades como casos especiais do diagrama de casos de uso. Uma vez que uma atividade representa um possível caso de uso do sistema de forma detalhada, os diagramas estão relacionados de certa forma. É possível elaborar diagramas de atividades com base nos casos de uso já descritos para o sistema.

Esse diagrama modela atividades que ocorrem de forma concorrente durante a execução do sistema, isto é, ações que ocorrem paralelamente em diferentes partes do sistema a partir do início da atividade. Esta visão completa do sistema auxilia no desenvolvimento e na procura por causas de erros, visto que apresenta todas as partes necessárias para a execução de cada atividade.

Assim, como podemos representar o sistema? O diagrama de atividades exibirá passo a passo as ações do sistema, considerando cada uma das partes que está processando alguma operação enquanto a atividade é realizada. Este diagrama é orientado a fluxos de controle, tendo uma similaridade com os fluxogramas de programas. Todavia, essa semelhança está apenas na aparência, devido a uma maior quantidade de informações, já que, em um diagrama de atividades, pode-se englobar representações concorrentes e sincronização de fluxos de controles. É importante destacar que o diagrama de atividades possibilita expandir a análise de um caso de uso.

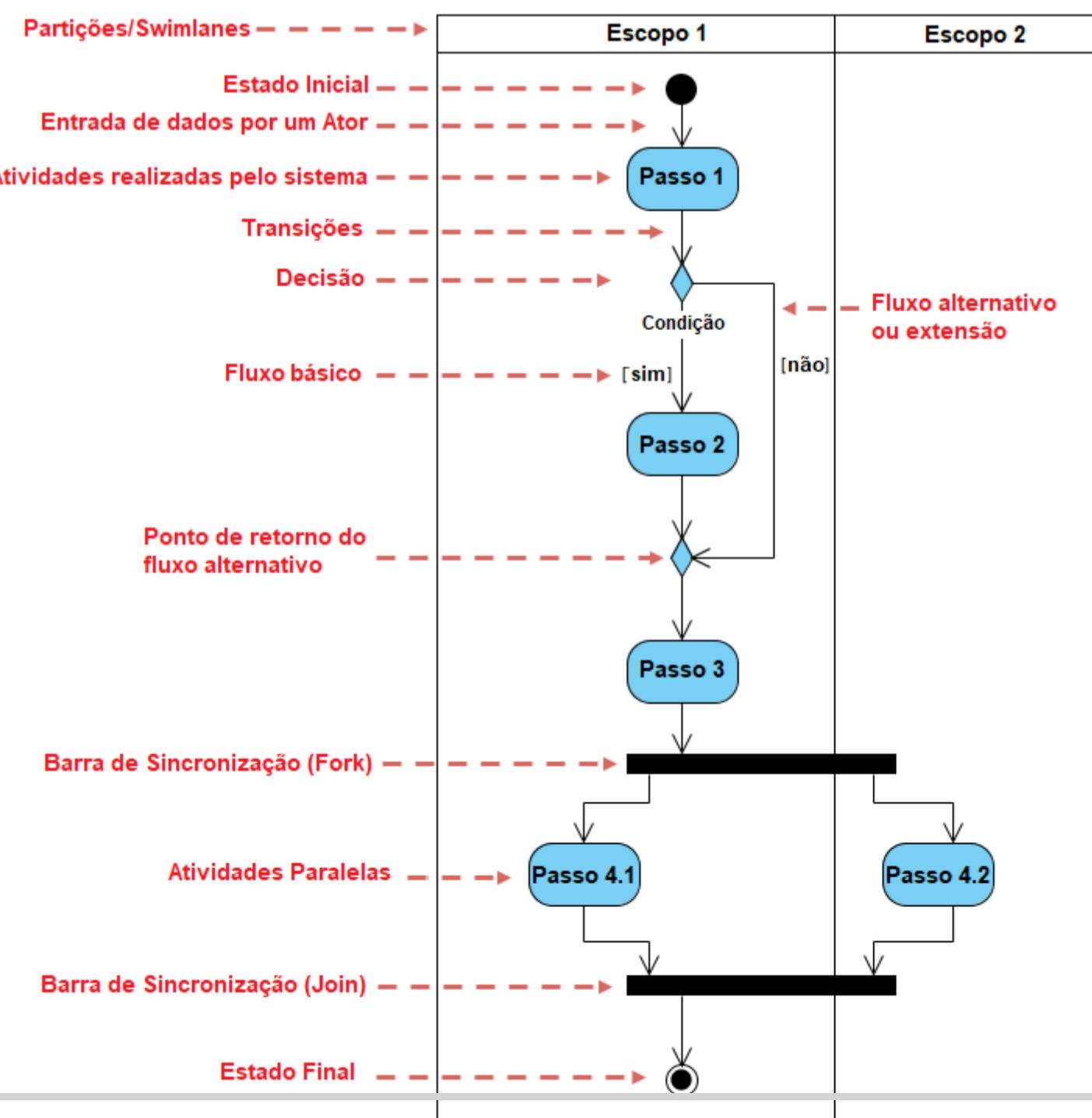
ELEMENTOS DO DIAGRAMA DE ATIVIDADES

Em geral, a criação dos diagramas de atividades é realizada do topo para o final e, normalmente, possui os seguintes elementos principais:

- Estados iniciais e finais.
- Atividades.
- Decisões.
- Transições.
- Barras de sincronização.
- *Swimlanes* (partições).

Todos os elementos listados serão abordados na sequência, trazendo alguns exemplos de diagramas. Antes desse detalhamento, a Figura 2.30 apresenta uma visão do diagrama com os elementos a serem descritos.

Figura 2.30 | Exemplo de um diagrama de atividades com exemplos de seus elementos



ESTADOS INICIAIS E FINAIS

Todo diagrama possui os estados iniciais e finais em sua notação, conforme notação ilustrada na Figura 2.31. Em um diagrama de atividades, pode-se ter um símbolo inicial e um símbolo final, porém o símbolo terminal pode ser utilizado mais de uma vez.

Figura 2.31 | Símbolos inicial, final e terminal de um diagrama de atividades



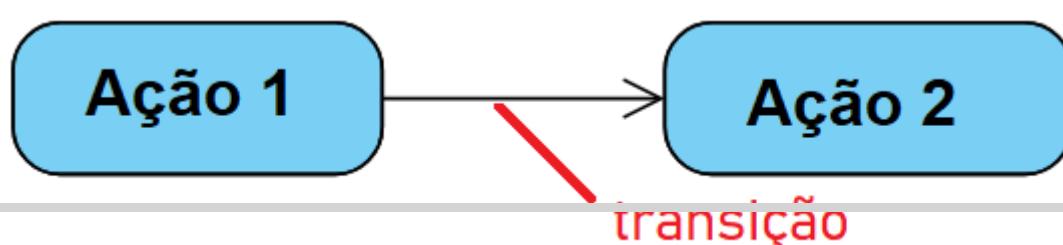
Fonte: elaborada pelo autor.

A diferença entre o símbolo final e o símbolo terminal é que, para um determinado fluxo de controle, se for utilizado o símbolo terminal, significa o final de um fluxo de processos específicos. Nesse caso, não representa o final de todos os fluxos de uma atividade. Por exemplo, caso tenha mais de um fluxo em paralelo, deve ser executado em seguida até que se atinja o símbolo final. Dessa forma, se utilizarmos o símbolo final, representará a finalização de todos os fluxos de um processo. Ao encontrar o símbolo final, a atividade está encerrada e não é necessário que nenhuma outra ação seja executada.

AÇÕES INTERNAS DAS ATIVIDADES

As ações internas em uma atividade são representadas por um retângulo de bordas arredondadas, e as setas entre eles, chamadas de transições, representam a mudança de uma atividade para a outra. Esses símbolos estão representados na Figura 2.32.

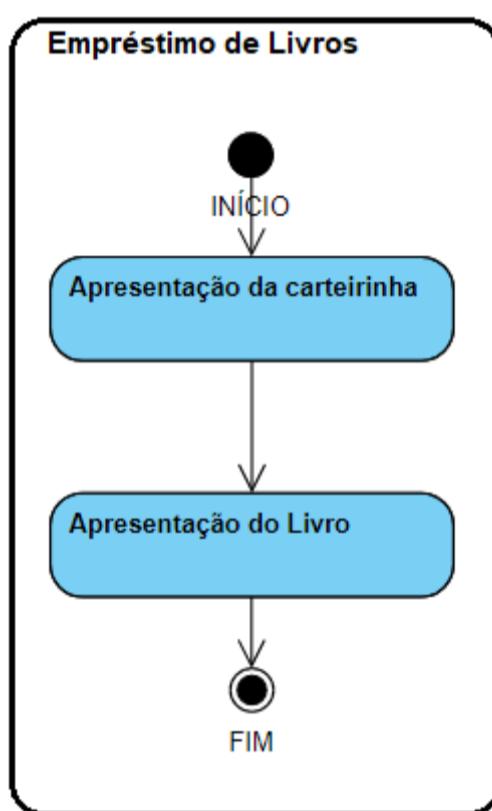
Figura 2.32 | Exemplos de atividades e transições



EXEMPLIFICANDO

Para exemplificar, imagine uma situação em que uma pessoa retira um livro de uma biblioteca. A atividade tratada nesse caso é o empréstimo de um livro. A ideia é que a pessoa apresente o cartão e o livro, então a atividade é encerrada. O diagrama da Figura 2.33 apresenta a atividade de retirar o livro.

Figura 2.33 | Diagrama de atividade simplificado de empréstimo de livros



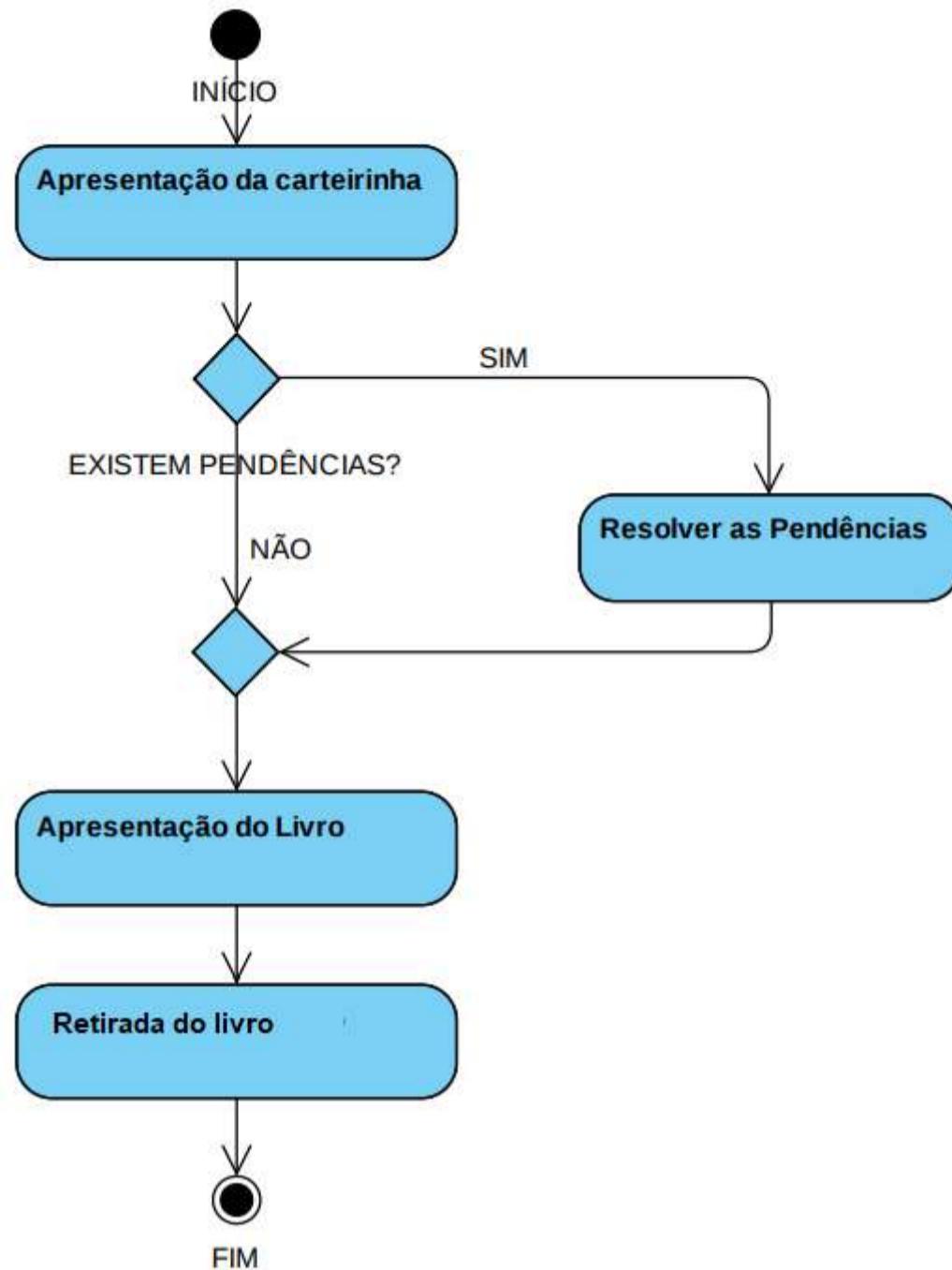
Fonte: elaborada pelo autor.

Neste exemplo, veja que as ações internas, normalmente, possuem nomes que resumem o que estão representando no fluxo de controle.

Agora, considere que existe uma condição a ser expressa no diagrama de atividades, ou seja, existe um teste a ser realizado e, se ele for verdadeiro, o fluxo de controle executará uma ação, mas, se for falso, outra ação. As estruturas condicionais são expressas nos diagramas de atividades como losangos, os quais tanto podem ser utilizados para representar a divisão do fluxo devido a uma condição como para unir dois fluxos para resultarem em uma mesma atividade.

Para entender como construir esta situação, imaginaremos, novamente, a questão do empréstimo de um livro de uma biblioteca. Pense que, ao apresentar a carteirinha para retirar o livro, existe uma verificação para saber se existe alguma pendência no cadastro do usuário (um livro sem devolução ou multas a pagar, por exemplo). Caso não haja, é possível retirar o livro, porém, caso haja, é necessário resolver as pendências antes de retirar o livro. A Figura 2.34 apresenta uma possibilidade de diagrama de atividades considerando esta situação nova que aumenta a complexidade do diagrama.

Figura 2.34 | Empréstimo de livros com condicional



Fonte: elaborada pelo autor.

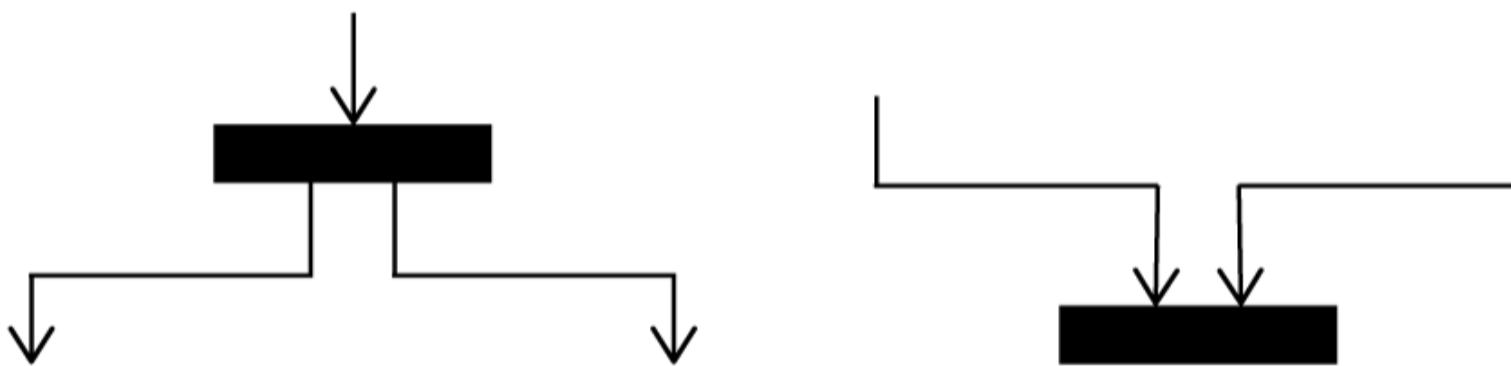
Nota-se, na Figura 2.34, que o primeiro losango representa a condicional se existem ou não pendências no cadastro do usuário. Caso haja, o usuário deve resolvê-las para, depois, pegar o livro; caso não haja, é possível pegar o livro de qualquer forma. O segundo losango é responsável por reunir os fluxos que foram divididos, já que resultam na mesma atividade seguinte.

Observe que a utilização de operadores condicionais está condicionada à execução de um determinado requisito por vez, no entanto, em muitos casos, há a necessidade de representar dois ou mais fluxos de controle que podem ser executados concorrentemente.

BARRAS DE SINCRONIZAÇÃO

Além disso, podemos dividir um fluxo de controle em múltiplos fluxos executados concorrentemente. Nesse caso, pode-se utilizar as barras de sincronização. Para isso, existem dois componentes: um chamado **FORK** e outro chamado **JOIN**. Ambos são representados por uma barra transversal sólida. A diferença é que o FORK representa a bifurcação (divisão) de um fluxo de controle em múltiplos fluxos que podem ser executados simultaneamente. Já o JOIN mostra a união de dois ou mais fluxos executados concorrentemente. A Figura 2.35 ilustra a notação de ambos, respectivamente.

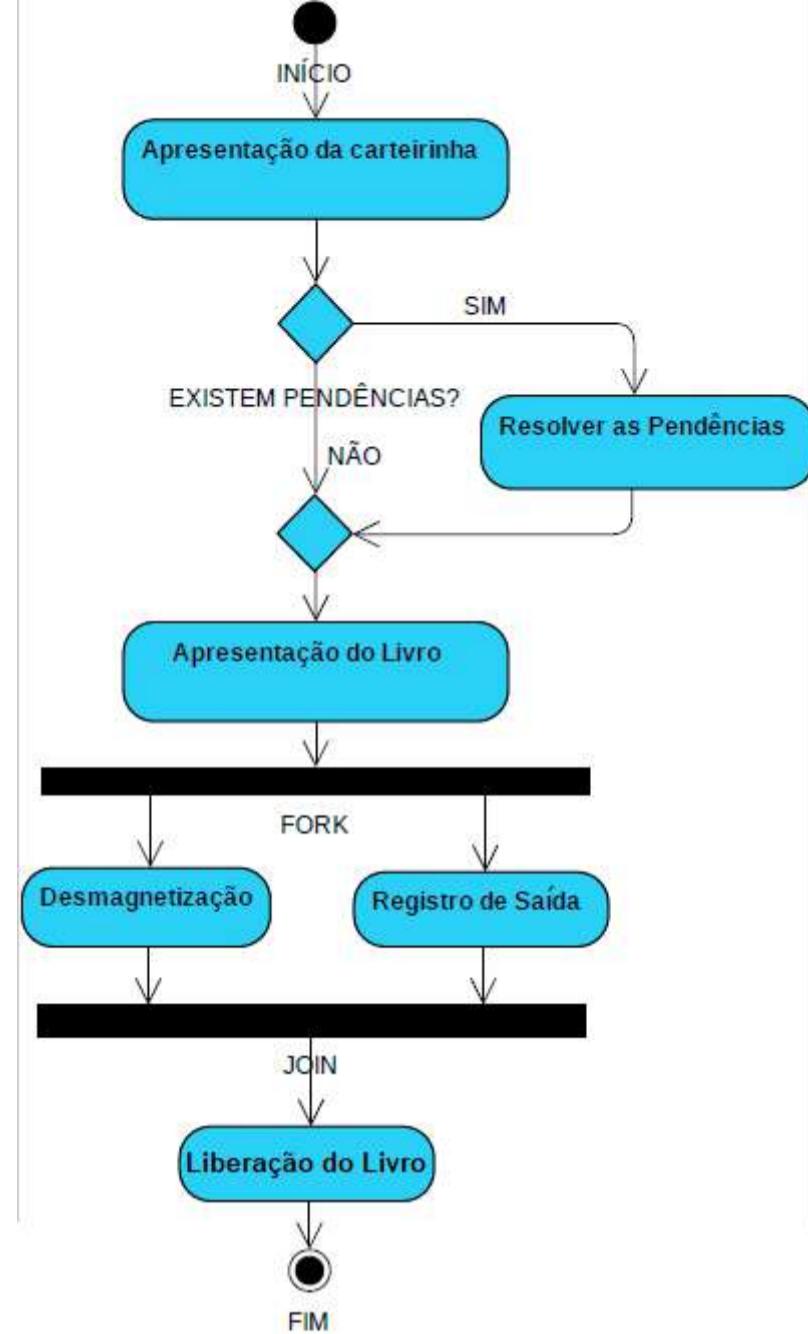
Figura 2.35 | Notação dos componentes FORK e JOIN



Fonte: Visual Paradigm (2020).

Um exemplo de aplicação de FORK e JOIN pode ser apresentado na retirada do livro. Imagine que, após a verificação, temos duas operações para serem realizadas: a **desmagnetização do livro** e o **cadastro da saída no sistema**. Pode-se considerar que ambas as operações podem ser executadas simultaneamente para a **liberação do livro**, assim temos a aplicação do JOIN. O FORK pode ser visto na bifurcação (divisão) da apresentação do livro para a desmagnetização e o registro de saída. A Figura 2.36 apresenta o diagrama com estas mudanças.

Figura 2.36 | Empréstimo de livros com FORK e JOIN



Fonte: elaborada pelo autor.

ASSIMILE

As operações representadas por componentes FORK e JOIN são paralelas e ocorrem de forma concorrente. Isso quer dizer que ocorrem ao mesmo tempo em dois fluxos de controle diferentes. Esse processo pode também ser entendido como uma sincronização.

Sincronizar operações é o ato de iniciá-las ao mesmo tempo e somente continuar o fluxo após todas terem terminado.

No exemplo da Figura 2.36, ao realizar um JOIN antes de liberar o livro, estamos querendo dizer que o livro só será liberado após o cadastro de saída do sistema e a desmagnetização terem terminado. Não importa se uma operação demorar mais tempo do que outra, a questão é que o fluxo só continua quando as duas terminarem, portanto é necessário sincronizá-las.

Ao realizar um FORK e um JOIN, pode-se afirmar que naqueles pontos deve existir uma barreira que sincroniza operações tanto de início de fluxos paralelos quanto de junção de fluxo após o término de operações.

Agora que aprendemos os componentes básicos de um diagrama de atividades, precisamos entender algumas questões sobre os fluxos.

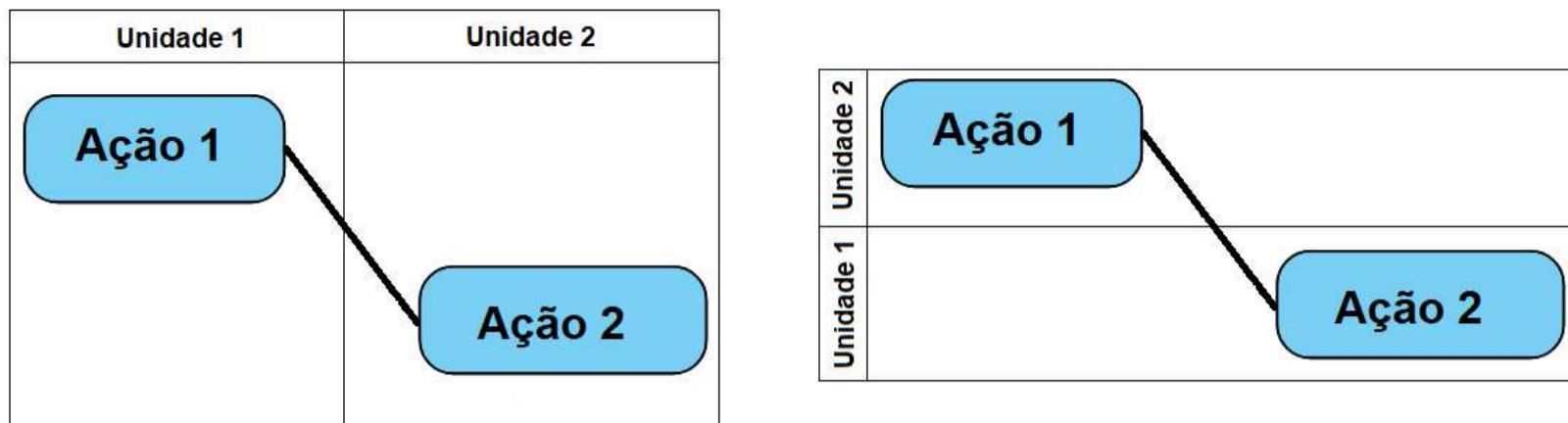
■ SWIMLANES

O fluxo de controle visto até o momento era sequencial até que sofria uma divisão em um FORK e uma junção em um JOIN. Essa divisão tornava o fluxo de controle paralelo em certo ponto. Entretanto, é possível pensar em um fluxo sequencial e/ou paralelo que ocorra em mais de um local.

Imagine, ainda no caso do sistema de empréstimo de livros, que existe um servidor que roda o banco de dados da biblioteca e pode ser tratado em três dimensões no diagrama de atividades: do usuário, do atendente e do sistema. Para esse caso, podemos organizar de uma forma que possibilite que o diagrama de atividades possua as três visões.

O componente utilizado para realizar essa divisão é chamado de *swimlane*, que é uma partição lógica. Existem diversas possibilidades de se organizar o diagrama de atividades com relação aos casos de uso, aos processos e até mesmo aos objetos. Em cada uma das “pistas” (do inglês, *lane*) do *swimlane* é colocado o nome da unidade que se deseja mapear as atividades e, logo abaixo, são organizadas as atividades relativas àquela entidade/objeto/unidade. A Figura 2.37 apresenta duas formas de se indicar as *swimlanes*.

Figura 2.37 | Exemplos de utilização de *swimlanes*



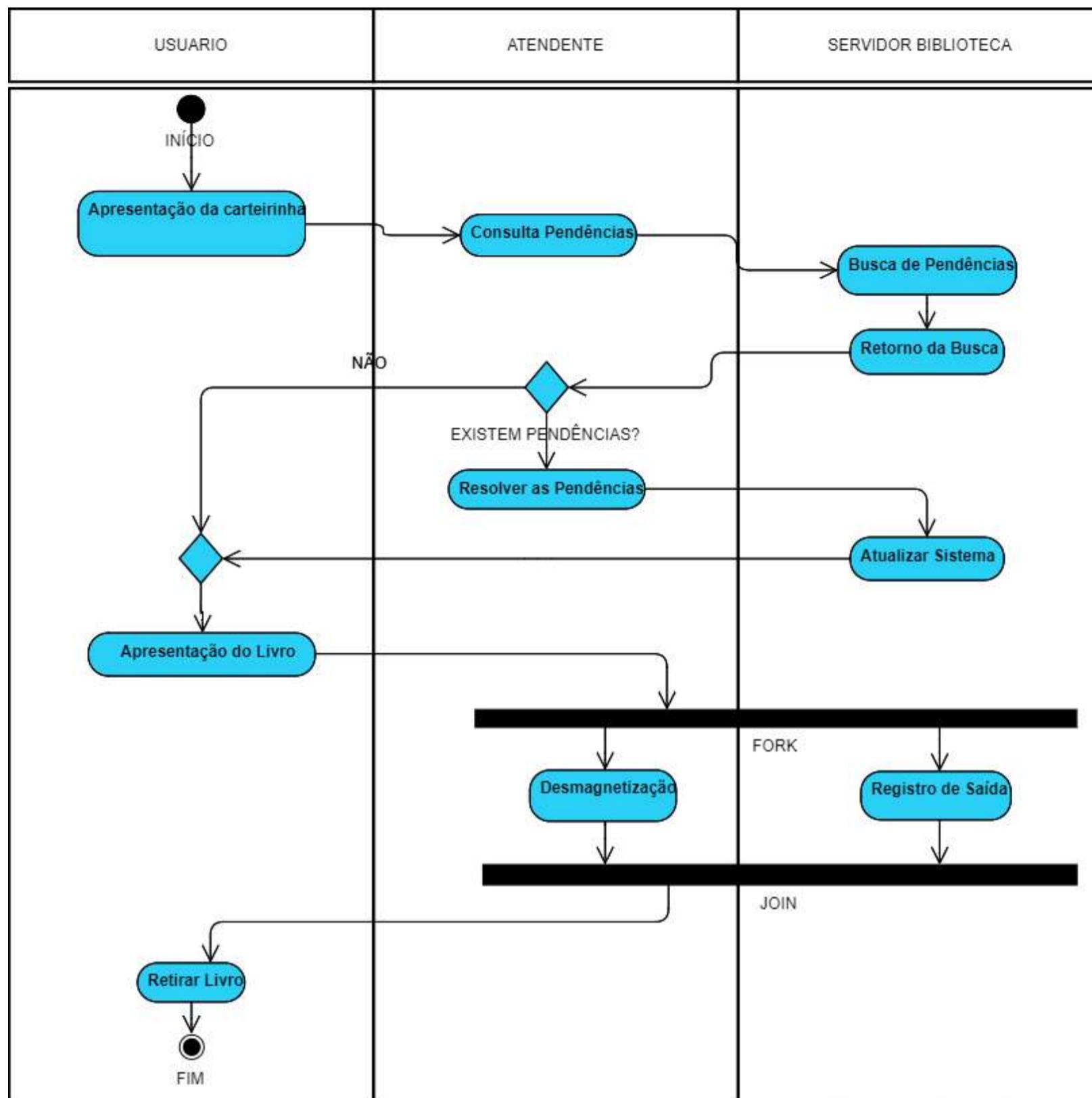
Fonte: elaborada pelo autor.

As *swimlanes* apresentadas na Figura 2.37 estão indicadas por retângulos que envolvem as ações da atividade relativas a cada unidade. Podem ser descritas de forma horizontal e vertical, devendo ser escolhida a disposição que apresenta o melhor entendimento do diagrama. Ao se construir *swimlanes* no diagrama de atividades, é necessária a reorganização das atividades de modo que cada unidade possua aquelas relativas a ela sem prejudicar o fluxo de controle.

Aplicaremos o conceito ao diagrama de atividades do empréstimo de livros. Considerando o que foi proposto, seriam necessárias três dimensões: uma para o usuário, uma para a atendente (ou para o sistema da biblioteca) e uma para o servidor que contém o banco de dados da biblioteca.

A Figura 2.38 ilustra uma das possibilidades de representação com o diagrama de atividades. No caso do usuário, ele apresenta a carteirinha; em seguida, o livro; e, por fim, realiza a retirada. A atendente consulta no sistema se existem pendências que poderiam impedir a realização do cadastro do usuário para o empréstimo. Caso existam pendências, ela poderá resolver e, logo depois, o usuário poderá apresentar, desmagnetizar e, finalmente, retirar o livro. Além disso, pode-se descrever os passos do sistema relacionados à busca sobre as pendências retornando o resultado para a atendente, a atualização dos dados do usuário, caso as pendências tenham sido resolvidas, e o registro da saída do livro.

Figura 2.38 | Utilizando *swimlanes* para organizar o diagrama de empréstimo de livro



Fonte: elaborada pelo autor.

ATENÇÃO

Importante observar que, ao adicionar as *swimlanes* neste diagrama de atividades, algumas partes foram modificadas, sendo necessárias algumas adições para a completude do diagrama. Essas mudanças mostram que a representação anterior possuía diversas omissões que poderiam comprometer o desenvolvimento do sistema. Este caso mostra claramente que, quando o diagrama de atividades é implementado de forma correta, pode-se obter uma representação completa do sistema, sendo possível até visualizar possíveis falhas na etapa de desenvolvimento.

No diagrama da Figura 2.38, também é possível observar que existem partes do fluxo que são sequenciais até que se encontra o FORK, o qual transforma o fluxo de controle em paralelo. A parte paralela do diagrama ocorre na desmagnetização do livro pela atendente, na qual o sistema automaticamente registra a saída do material. A sequência de ações é retomada após o JOIN e resulta na retirada do livro pelo usuário. Todas as outras atividades apresentadas no diagrama ocorrem de forma sequencial, apesar de estarem em partições diferentes. Repare na representação dos dois componentes, cujas barras estão entre duas pistas, indicando que cada atividade ocorre em um local diferente.

Ver anotações

REFLITA

Nesta seção, discutimos a relação entre o diagrama de atividades e os diagramas de casos de uso. Sabemos que estes sofrem modificações ao longo do processo de desenvolvimento, principalmente no início. Reflita sobre qual é o momento certo de fazer os diagramas de atividades. Seria no início, para auxiliar no desenvolvimento dos casos de uso? Após a primeira versão do diagrama de casos de uso validada pelo cliente, para que não haja trabalho desnecessário? Ou seria interessante fazer os diagramas de atividades em todas as etapas do processo e ver suas mudanças? Lembre-se de que o tempo necessário para fazer os diagramas pode impactar no tempo de desenvolvimento de software.

Chegamos ao final da seção sobre os diagramas de atividades. Neste momento, você é capaz de identificar os principais componentes de um diagrama, analisar e construir um diagrama a partir de um problema e fazer o processo de melhorias até que se obtenha um diagrama completo, cuja representação é o que se esperava. Aliando este diagrama aos diagramas de casos de uso e de classes, agora, você

possui conhecimento sobre três importantes representações existentes na UML. Chegou a hora de testar seus conhecimentos e praticar o que você aprendeu. Preparado? Bons estudos!

REFERÊNCIAS

REGGIO, G.; LEOTTA, M.; RICCA, F.; CLERISSI, D. What are the used activity diagram constructs? A survey. *In: INTERNATIONAL CONFERENCE ON MODEL-DRIVEN ENGINEERING AND SOFTWARE DEVELOPMENT*, 2., 2014, Lisboa. **Anais [...]**. Lisboa: MODELSWARD, 2014.

UML-DIAGRAMS. **UML Activity Diagram Examples**. 2016. Disponível em: <https://bit.ly/2EXl88e>. Acesso em: 8 jul. 2020.

VISUAL PARADIGM. **What is Activity Diagram?**. 2020. Disponível em: <https://bit.ly/3bjs9wh>. Acesso em: 27 ago. 2020.

FOCO NO MERCADO DE TRABALHO

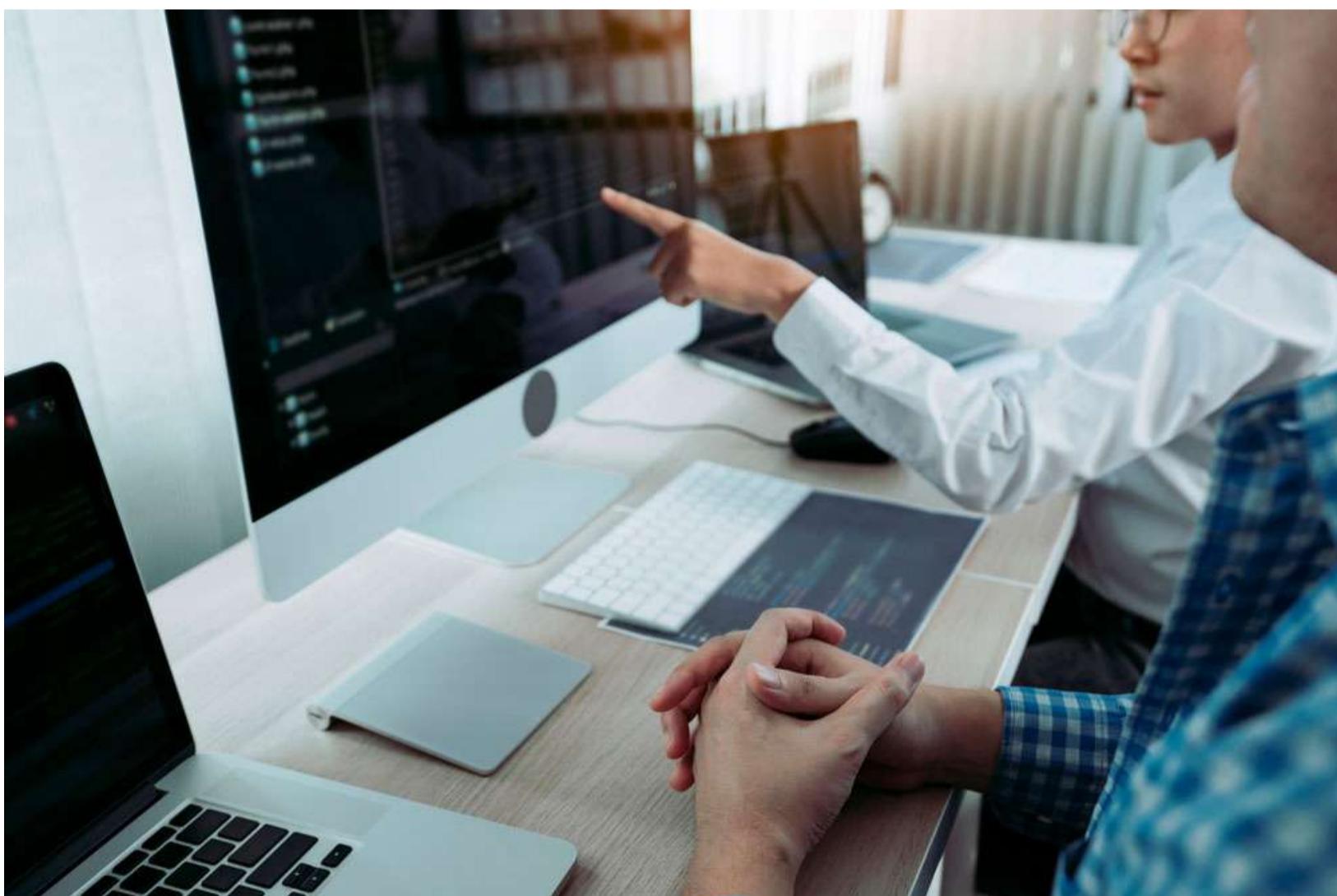
MODELAGEM DE ATIVIDADES

Maurício Acconcia Dias

Ver anotações 0

REPRESENTAÇÃO DO SISTEMA

O diagrama de atividades exibe passo a passo as ações do sistema, considerando cada uma das partes que está processando alguma operação enquanto a atividade é realizada.



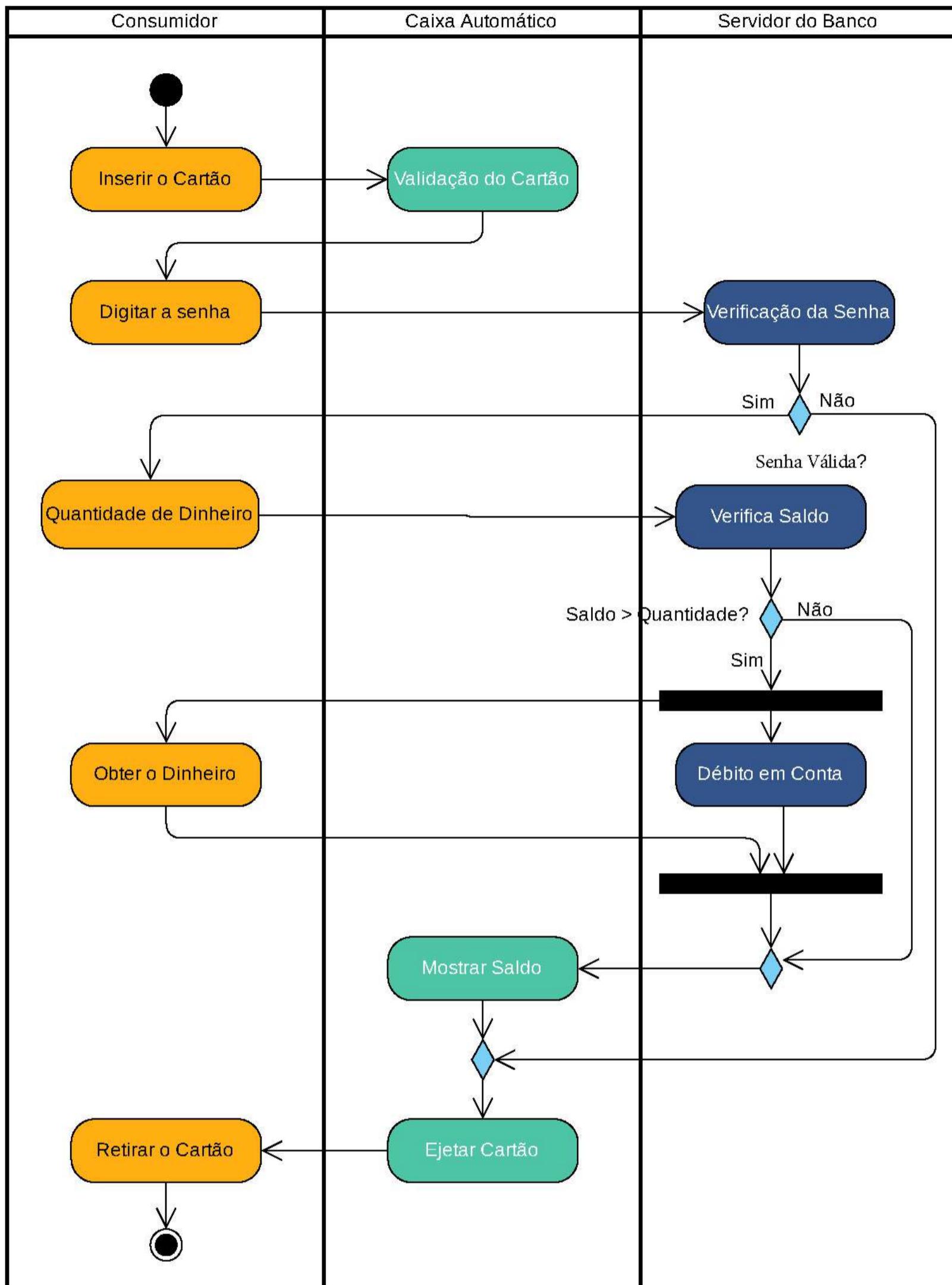
Fonte: Shutterstock.

Deseja ouvir este material?

Áudio disponível no material digital.

Considerando as orientações apresentadas na situação-problema, é possível solucionar o problema com um diagrama de atividades contendo três partições/*swimlanes*, como ilustrado na Figura 2.39 a seguir.

Figura 2.39 | Diagrama de atividades para um caixa eletrônico



Fonte: elaborada pelo autor.

Este diagrama apresenta três *lanes*, considerando que é possível dividir as ações desta atividade entre o consumidor, o caixa automático e o servidor do banco. As ações do consumidor são comuns na utilização do caixa automático para operações de saque envolvendo a autenticação, a solicitação e a obtenção do dinheiro. Neste caso, é importante lembrar que algumas operações são feitas no caixa do banco, porém outras são

realizadas em servidores que ficam mais distantes. No caixa eletrônico propriamente dito, são feitas ações com dois objetivos: identificar o cliente e retornar informações a ele. Toda parte da operação de saque em si é feita nos servidores do banco por motivos de segurança e consistência dos dados.

o

Ver anotações

NÃO PODE FALTAR

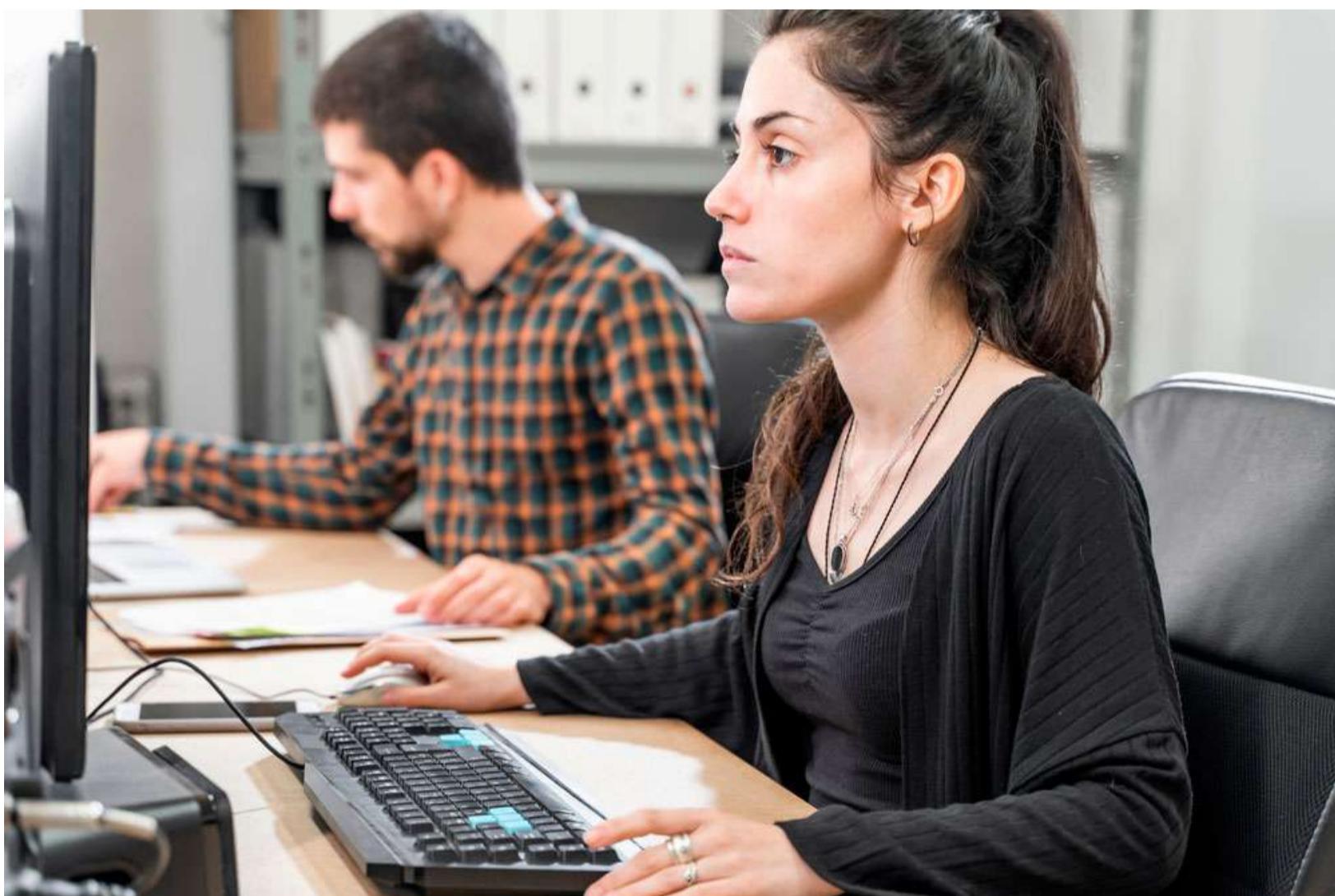
MODELAGEM DE ESTADOS

Iolanda Cláudia Sanches Catarino

Ver anotações 0

DIAGRAMA DE MÁQUINA DE ESTADOS

O diagrama de máquina de estados permite descrever o ciclo de vida de objetos de uma classe, os eventos que causam a transição entre os estados e a realização de operações resultantes.



Fonte: Shutterstock.

Deseja ouvir este material?

Áudio disponível no material digital.

Olá! Seja bem-vindo a esta unidade do livro de Análise Orientada a Objetos.

Independentemente do modelo de processo de engenharia de software a ser adotado no desenvolvimento de um sistema, cada empresa ou gerente de projeto define a sua melhor metodologia de desenvolvimento de sistemas em consonância com as boas práticas da engenharia de software, abrangendo todo o processo de software, os métodos e as ferramentas, a fim de atenderem aos diferentes domínios e complexidades de sistemas de softwares.

As atividades iniciais de análise de requisitos e análise de um processo de desenvolvimento de software exigem maiores esforços, comunicação e interação do analista de sistemas com os usuários responsáveis pelas atividades dos processos de negócio que envolvem a área da solução a ser desenvolvida. Conhecer e compreender o domínio do sistema, bem como delimitar o escopo do projeto de software, são ações extremamente importantes para cumprir com sucesso todo o processo de desenvolvimento de um sistema, conforme previsto.

É importante ressaltar que a Linguagem de Modelagem Unificada (UML - *Unified Modeling Language*) apoia o desenvolvimento incremental do software orientado a objetos a partir de modelos que podem evoluir com a inclusão de novos detalhes, enfatizando a descrição de um sistema nas perspectivas estrutural e comportamental, ou seja, a visão estática e dinâmica do sistema especificada em diferentes técnicas de modelagem. Assim, a adoção da UML não está vinculada exclusivamente a uma atividade ou a uma fase do processo de desenvolvimento de software, definindo quem deve fazer o que, quando e como.

Para modelagem da atividade ou fase de análise com a UML, é usual especificar, primeiramente, a modelagem dos casos de uso correspondentes aos requisitos funcionais do sistema, o modelo de classes e a modelagem de atividades que demonstram o fluxo de controle das funcionalidades do sistema, ressaltando os aspectos dinâmicos do sistema. Na sequência, recomenda-se documentar o comportamento dos objetos das classes, descrevendo como um sistema responde aos eventos internos e externos que influenciam na mudança dos estados de um objeto. Assim, na primeira seção desta unidade, será apresentado o Diagrama de Máquina de Estados, que tem como objetivo demonstrar o comportamento de um elemento através de um conjunto de transições de estados realizadas entre os estados dos objetos de uma classe, de um caso de uso ou mesmo de um sistema completo. Na segunda seção, continuaremos com a modelagem dinâmica do sistema sob a perspectiva de interação de como os objetos do sistema agem internamente para apoiarem a realização das funcionalidades representadas pelos casos de uso, trabalhando com o Diagrama de Sequência. Por fim, na terceira seção, exploraremos os conceitos, componentes e exemplos dos demais diagramas de interação da UML: Diagrama de Comunicação, Diagrama de Visão Geral de Interação e Diagrama de Tempo.

Caro aluno, seja bem-vindo à seção sobre a modelagem de estados dos objetos.

Avançando na modelagem de análise, é fundamental definir o comportamento dos objetos na perspectiva dos estados que eles assumem durante a execução das funcionalidades do sistema. Todo objeto do mundo real ou do mundo computacional assume diferentes estados durante a sua existência. Por exemplo, no mundo real, um avião está em repouso, decolando, voando ou pousando; uma pessoa está parada, caminhando, correndo, etc. Da mesma forma, cada objeto, no mundo computacional, se encontra em um estado particular, decorrente de algum acontecimento que ocasiona as mudanças de estado, a fim de atender a uma regra de negócio do contexto do sistema. Por isso, compreender a modelagem dos estados dos objetos e suas transições é essencial para documentar de forma consistente um software orientado a objetos.

Durante a execução de uma funcionalidade do sistema, um objeto muda de estado quando acontece algum evento interno ou externo ao sistema, provocando uma transição entre os estados do objeto e, com isso, ele realiza determinadas ações responsáveis pela consistência e integridade dos dados do sistema. Como exemplo, imagine você efetuando uma compra on-line em uma loja de produtos esportivos. Ao você escolher e confirmar os produtos que deseja comprar, o seu pedido de compra assume o estado de “Pedido recebido”; posteriormente, o sistema de venda da loja integrado ao sistema de faturamento confere se o pagamento foi aprovado e, uma vez que isso acontece, ele altera o estado do mesmo objeto pedido para “Pagamento aprovado”; na sequência do processo de compra do cliente, o setor de estoque e distribuição de produtos da loja separa os produtos do pedido e encaminha-os para o setor de logística, o qual providencia o transporte do pedido ao destinatário. A partir desse momento, o sistema atualiza a situação do pedido para “Transporte em andamento” e, assim que o cliente recebe o pedido, o sistema atualiza, por último, para “Entrega realizada”. No contexto desse exemplo, observa-se que o mesmo objeto – pedido de compra do cliente – transitou por diferentes estados durante um período de tempo de execução do sistema, portanto cada objeto pode transitar por um número finito de estados durante a sua existência.

Nesta seção, você compreenderá os conceitos e os componentes que integram a técnica de modelagem comportamental denominada Diagrama de Máquina de Estados, que representa o ciclo de vida dos objetos com a especificação dos estados e suas transições. Também, conhecerá a notação gráfica de cada elemento do diagrama, conforme a UML, para aplicá-la corretamente na elaboração do Diagrama de Máquina de Estados. Para reforçar o seu aprendizado, em sua atuação como analista de sistemas, você terá como desafio a abstração e identificação dos objetos que possuem estados relevantes a serem

documentados, a definição dos estados e suas transições e a modelagem dos Diagramas de Máquina de Estados para um sistema de hotelaria, referente ao módulo recepção.

Você, como analista de sistemas em uma empresa de desenvolvimento e soluções de software e na sua atuação de responsável pelo projeto de um sistema de hotelaria – módulo recepção, considerando que toda a modelagem da análise de requisitos e a modelagem estática da atividade de análise do sistema já foram especificadas, principalmente o Diagrama de Classes, terá que definir para quais objetos das classes do sistema deve ser modelado o Diagrama de Máquina de Estados, a partir das regras de negócio estabelecidas na modelagem dos requisitos funcionais do sistema.

Para reforçar a sua aprendizagem, considere que você faz parte de uma equipe de desenvolvedores de uma empresa de desenvolvimento de sistemas de software e que um dos projetos em andamento é referente ao módulo recepção de um sistema de hotelaria. A descrição a seguir relata o contexto das principais atividades da rotina da recepção do hotel, incluindo o controle de reservas, com suas regras de negócio.

|| DESCRIÇÃO DO MÓDULO RECEPÇÃO

O módulo recepção é responsável por controlar as reservas, os cadastros dos hóspedes e de empresas e o check-in (entrada do hóspede) e check-out (saída do hotel) de um hóspede.

É necessário manter um cadastro dos apartamentos do hotel, no qual deve constar o número, o tipo (padrão solteiro, padrão casal, padrão conjugado, luxo solteiro, etc.), a situação (disponível, ocupado, em arrumação, em manutenção ou desativado) e o valor da diária. Cada apartamento é classificado com um tipo apenas.

Um hóspede deve ser cadastrado com: CPF, RG, nome, endereço completo, data de nascimento, telefone residencial, celular, sexo, filiação, escolaridade, estado civil, nacionalidade, endereço eletrônico, número do passaporte (se for hóspede estrangeiro), situação (normal, preferencial, inadimplente, encerrado), placa do carro e nome da empresa que trabalha. Para um hóspede que se hospeda com sua família, é necessário fazer o cadastro de todos os membros da família hospedados. Um hóspede pode se hospedar em um hotel vinculado à empresa que trabalha ou de uma empresa provedora de gerenciamento global de viagens corporativas, sendo necessário ter o cadastro prévio dessas empresas. Uma empresa conveniada com o hotel deve ser cadastrada com: CNPJ, nome fantasia, razão social, inscrição estadual, ramo de atividade, endereço completo, telefone, endereço eletrônico, celular, e-mail e nome de uma pessoa para contato.

Um hóspede, para se hospedar no hotel, tem que ter um cadastro com seus dados pessoais efetuado previamente e uma reserva prévia realizada por ele (por telefone, por aplicativo ou pela web) ou pela empresa que tem vínculo. Sendo a empresa responsável pela estada do hóspede, ela é a encarregada de custear integralmente as despesas referentes às diárias, e quanto às refeições, ela pode custear ou não.

Uma reserva é mantida por: nome da(s) pessoa(s) que irá(ão) se

hospedar em um apartamento, data que efetuou a reserva, data de entrada, hora prevista de entrada, data de saída, quantidade de adultos, quantidade de crianças, idade das crianças e nome da empresa que trabalha (se a empresa for a responsável pela estada do hóspede). Um hóspede ou uma empresa pode realizar várias reservas.

Um hóspede, ao chegar ao hotel, informa seu nome para resgatar a reserva, confere seu cadastro pessoal e o atualiza, caso seja necessário. O check-in é registrado com: número do apartamento, dados do hóspede, data de entrada, hora de entrada, data prevista de saída, nome do acompanhante (também cadastrado como hóspede), se tiver, placa do carro e observação.

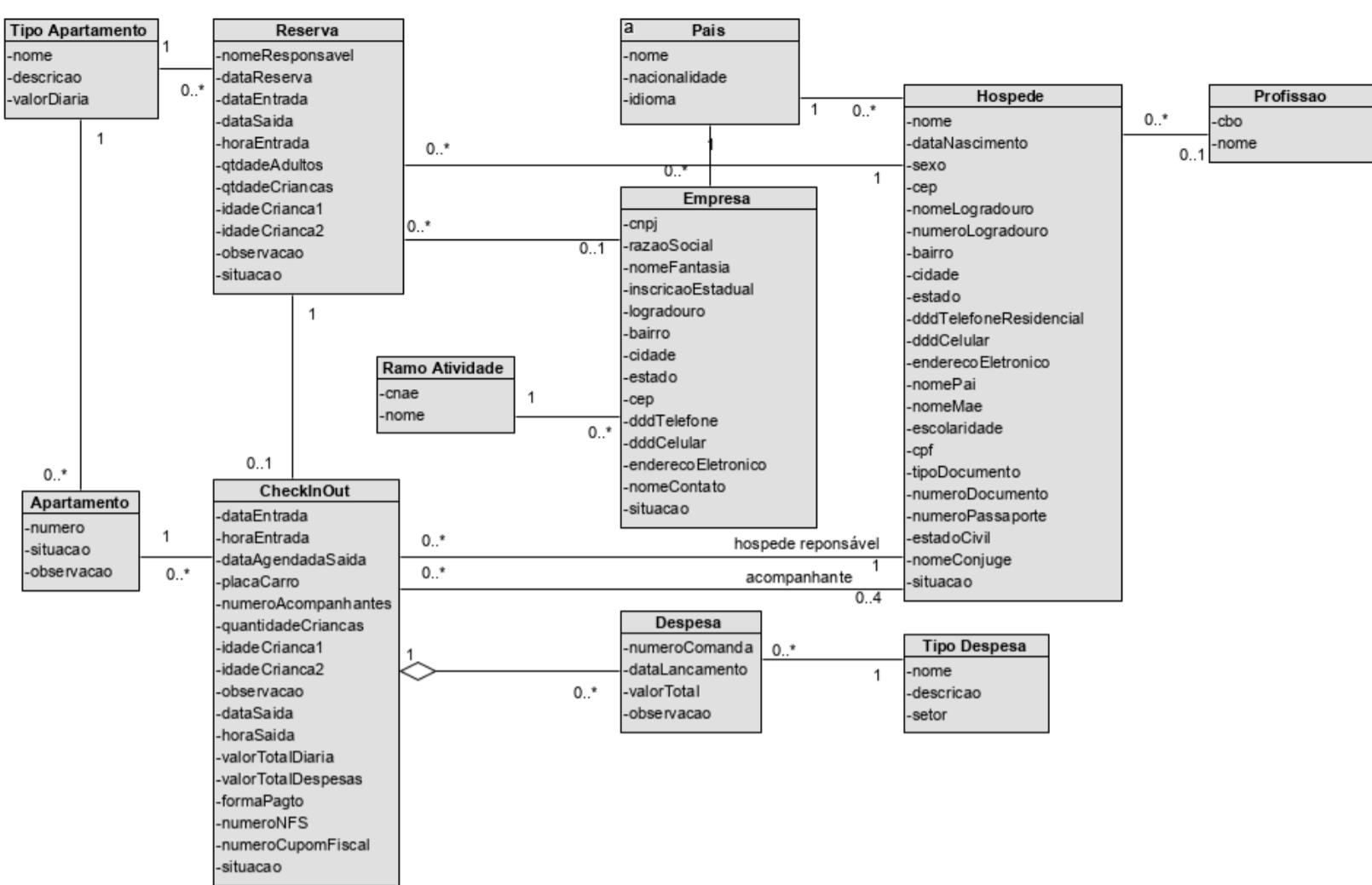
Durante a estada do hóspede no hotel, todos os alimentos consumidos ou serviços realizados para o hóspede são lançados como despesas, registrando o número do apartamento, a data, o tipo de despesa (serviço de lavanderia, refeição no restaurante, lanche, etc.), a descrição da despesa, a quantidade e o valor.

Na saída do hóspede é realizado o check-out, registrando: a data e a hora de saída, o valor total das despesas e das diárias, totalizando o valor do check-out, e a forma de pagamento (dinheiro ou cartão). Assim que o hóspede confere todas as despesas lançadas e confirma o seu check-out, são emitidos a nota fiscal de saída, referente às diárias, e o cupom fiscal, referente às despesas.

Considerando que a modelagem dos casos de uso foi especificada e analisando o Diagrama de Classes de análise (apenas com os atributos) a seguir, para sua melhor familiarização com o contexto do sistema, faça:

Elabore o Diagrama de Máquina de Estados correspondente à classe de objetos “Reserva” com os estados Realizada, Confirmada, Cancelada ou Efetivada. Defina as transições de estados com regras que considere pertinente ao contexto do sistema.

Figura 3.1 | Diagrama de Classes



Fonte: elaborada pela autora.

Bom trabalho e ótimos estudos!

CONCEITO-CHAVE

As técnicas de modelagem comportamentais da UML representam o comportamento e a interação entre os elementos do sistema orientado a objetos, colaborando para modelagem da visão dinâmica do sistema. Geralmente, inicia-se a modelagem comportamental do sistema a partir da especificação do Diagrama de Casos de Uso e do Diagrama de Atividades e, na sequência, enfatiza-se a elaboração do Diagrama de Máquina de Estados correspondente aos objetos das classes com estados relevantes.

Você já sabe que todas as técnicas de modelagem da UML abrangem um conjunto de elementos que se vinculam aos conceitos básicos e aos princípios do paradigma orientado a objetos. Dessa forma, é fundamental relembrarmos os conceitos de estado, transição de estado e evento que sustentam o Diagrama de Máquina de Estados.

CONCEITOS BÁSICOS E PRINCÍPIOS DO PARADIGMA ORIENTADA A OBJETOS

o

- **Eventos:** representam os acontecimentos que provocam a mudança de estado dos objetos, podendo ser uma ação interna ou externa do objeto. Um evento, ao ser disparado em um instante de tempo de execução do sistema, invoca uma operação específica dos objetos de uma classe, a partir do envio de uma mensagem ao objeto, provocando uma mudança de estado dele. Na definição de Rumbaugh *et al.* (1997), um evento é uma ocorrência única de um estímulo individual de um objeto para outro que acontece em certo momento, disparando uma transmissão ou informação unidirecional entre os objetos.
- **Estados:** representam a abstração de uma forma de apresentação dos objetos em um instante de tempo de execução do sistema com uma duração finita, o qual demonstra a reação de um objeto em resposta a um evento. Na definição de Booch, Jacobson e Rumbaugh (2006, p. 290), um estado é “uma condição ou situação na vida de um objeto durante a qual o objeto satisfaz alguma condição, realiza alguma atividade ou aguarda um evento. Um objeto permanece em um estado por uma quantidade finita de tempo”. Considere como exemplo um objeto do tipo “Automóvel,” o qual, num instante de tempo (t_1), se encontra no estado “desligado” e, a partir de um evento disparado, por exemplo, o acionamento da chave na ignição do veículo, uma ação humana, o objeto reage e muda o seu estado para “ligado”, diante da partida do motor, no instante de tempo seguinte (t_2).
- **Transição de estado:** representa a mudança de estado de um objeto em resposta a um evento disparado. Para Booch, Jacobson e Rumbaugh (2006), uma transição é um relacionamento entre dois estados, indicando que um objeto no primeiro estado realiza alguma ação e assume outro estado quando um evento é disparado e condições são satisfeitas.

Ver anotações

DIAGRAMA DE MÁQUINA DE ESTADOS

A técnica de modelagem comportamental – Diagrama de Máquina de Estados – demonstra o comportamento dinâmico de um elemento por meio de um conjunto de estados relevantes e das transições entre os estados finitos dos objetos de uma classe, de casos de uso ou até mesmo do sistema como um todo. É considerado um estado relevante ou importante ao contexto do sistema o estado de um objeto que implica em ações a serem consistidas durante a execução do sistema, por exemplo, um objeto “funcionário”, que se encontra no estado de “afastado por licença médica” em determinado período, não pode ser permitido trabalhar durante o período estipulado do afastamento. Uma forma de consistir isso via sistema é não permitir que o usuário funcionário tenha acesso aos sistemas da empresa; ou um objeto “produto”, que se encontra no estado de “esgotado”, não pode ter a sua venda permitida. Dessa forma, na prática, o Diagrama de Máquina de Estados é recomendado, principalmente, para modelar os estados dos objetos das classes.

Segundo Guedes (2018), o Diagrama de Máquina de Estados demonstra o comportamento de um elemento, geralmente uma instância de uma classe, a partir de um conjunto finito de transições de estado. No entanto, pode-se usá-lo para modelar também o comportamento de um caso de uso. Conforme Bezerra (2014), esse diagrama permite descrever o ciclo de vida de objetos de uma classe com a indicação dos eventos que causam as transições entre os estados, a partir da realização das operações dos objetos.

ELEMENTOS DO DIAGRAMA DE MÁQUINA DE ESTADOS

Um Diagrama de Máquina de Estados é representado, basicamente, pelos elementos: estado inicial, estados, transições de estados e estado final, sendo que este não é obrigatório e, também, pode conter vários estados finais no mesmo diagrama. A notação gráfica de todos os

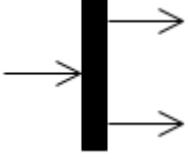
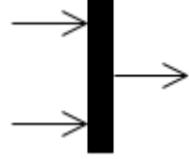
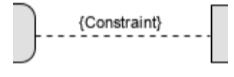
elementos que podem compor o Diagrama de Máquina de Estados e uma breve descrição de cada elemento são apresentadas no Quadro 3.1 a seguir.

0

Quadro 3.1 | Elementos do Diagrama de Máquina de Estados

Ver anotações

Notação	Elemento
●	Estado Inicial (<i>Initial State</i>) representa o estado de um objeto quando ele é criado, indicando o estado padrão que ele assumirá. Só pode haver um estado inicial na máquina de estados.
○	Estado Final (<i>Final State</i>) representa o fim do ciclo de vida de um objeto. Este estado é opcional e pode haver mais de um na máquina de estados.
	Estado (<i>State</i>) representa uma situação de existência dos objetos de uma classe, durante a qual ele satisfaz alguma condição ou realiza alguma atividade.
	Estado de Submáquina ou Estado Composto (<i>Submachine State</i>) indica que um estado contém internamente dois ou mais estados com suas transições, gerados independentes ou não. É uma forma de simplificar a representação da máquina de estados a partir do detalhamento de um estado principal.
	Transição (<i>Transition</i>) representa um relacionamento entre dois estados, indicando a mudança de estado a partir da ocorrência de um evento.
◆	Escolha (<i>Choice</i>) representa um ponto na transição de estados de um objeto em que deve ser tomada uma decisão. As transições de um estado de escolha devem ser indicadas com uma condição de guarda.

Notação	Elemento
	<p>Barra de Sincronização com Bifurcação (Fork) representa a ocorrência de estados paralelos, causados por transições concorrentes. Esta barra indica que dois ou mais processos paralelos estão sincronizados em um determinado momento do processo.</p>
	<p>Barra de Sincronização com Junção (Join) representa a ocorrência de estados paralelos, causados por transições concorrentes. Esta barra indica o momento em que dois ou mais subprocessos se uniram em um único processo.</p>
	<p>Condição (Constraint) indica uma restrição representada em linguagem natural no formato de texto, com o propósito de declarar parte da semântica de um elemento.</p>
	<p>Ponto de Entrada (Entry Point) indica a entrada de uma máquina de estado ou estado composto.</p>
	<p>Ponto de Saída (Exit Point) indica a saída de uma máquina de estado ou estado composto.</p>
	<p>Vértices de Junção (Junction) são usados para encadear várias transições, indicando a construção de caminhos de transição compostos entre estados.</p>
	<p>Histórico Superficial (Shallow History) representa uma transição para um estado histórico superficial de estados compostos, chamando o estado mais recente que estava ativo, ou seja, o histórico superficial guarda informações sobre o estado atual, mas não sobre os seus subestados.</p>

Notação	Elemento
⊕	Estado Profundo (<i>Deep History</i>) representa a configuração ativa mais recente do estado composto que contém diretamente esse pseudoestado, por exemplo, a configuração de estado que estava ativa quando o estado composto foi encerrado pela última vez, ou seja, o histórico profundo guarda informações sobre o estado atual e os seus subestados.
×	Terminador indica que a execução da máquina de estado por meio de seu objeto está terminada, ou seja, equivale à invocação do método de destruição o objeto.
	Nota ou comentário é utilizado para descrever observações aos elementos da máquina de estados, contudo não expressa força semântica específica aos elementos da máquina de estado, mas pode conter informações úteis para os desenvolvedores.

Fonte: adaptado de <https://bit.ly/2QOqeql>. Acesso em: 15 maio 2020.

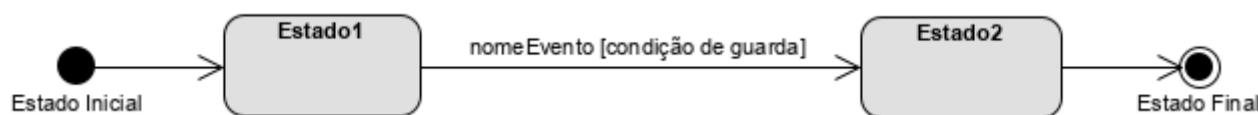
O Quadro 3.1 ilustra os elementos básicos do Diagrama de Máquina de Estados na sua forma mais simplificada, ou seja, a representação dos estados apenas com seu nome e a indicação das transições de estados e dos estados inicial e final. O nome de um estado deve ser único no diagrama, iniciando com letra maiúscula. Tipicamente, usa-se um verbo no gerúndio, por exemplo, “Ativando”, “Cancelando”, “Enviando”, etc.

ATENÇÃO

É importante lembrar que, os estados e as transições de estado de um objeto constituem o seu ciclo de vida, assim, cada diagrama tem apenas um único estado inicial e pode ter nenhum, um ou vários estados finais. Cada objeto passa por um número finito de estados durante a sua existência,

realizando determinadas ações durante a execução do sistema, sendo que, quando não é indicado um estado final no ciclo de vida de um objeto, isso indica que ele é contínuo.

Figura 3.2 | Elementos do Diagrama de Máquina de Estados



Fonte: elaborada pela autora.

ASSIMILE

O Diagrama de Máquina de Estados é adotado, principalmente, para modelar o comportamento dos objetos das classes, contudo nem todas as classes especificadas no Diagrama de Classes precisam de uma máquina de estados correspondente. O Diagrama de Máquina de Estados é elaborado apenas para as classes de objetos que possuem um comportamento dinâmico relevante e específico ao contexto do sistema modelado.

ELABORANDO O DIAGRAMA DE MÁQUINA DE ESTADOS

Na elaboração do Diagrama de Máquina de Estados, é fundamental identificar as regras de negócio aplicadas ao contexto dos objetos, para auxiliar na definição dos estados e das transições. Para identificar os possíveis estados de um objeto, deve-se analisar os valores de seus atributos, simulando a instanciação dos objetos, a partir da execução das funcionalidades do sistema, já especificadas pelos casos de uso, bem como compreender a troca de mensagens entre os objetos.

Para definir as transições entre os estados, deve-se identificar os eventos internos e externos aos objetos da classe e analisar se há algum fator que condicione a transição de estado, nesse caso, deve-se representar por meio da indicação de condições de guarda.

A elaboração do Diagrama de Máquina de Estados pode consistir na simples representação dos estados e nas transições entre eles, assim como em uma representação mais detalhada dos estados dos objetos com a indicação das atividades internas, também denominadas de ações de estado, e ainda apresentar as transições internas dos estados.

De acordo com Guedes (2018), uma ação está associada a uma transição, ou seja, quando o objeto assume um novo estado ou está mudando de estado, ocasionando uma simples atribuição de um valor a um atributo. Uma atividade interna está sempre associada ao estado que o objeto assumiu, ou seja, corresponde aos métodos executados pelo objeto, porém não causam alteração na situação do estado. As ações de estado são representadas pelas cláusulas predefinidas “*entry*, *exit* e *do*” no interior do retângulo do estado, sendo:

- ***Do*:** representa uma atividade realizada durante o tempo em que o objeto se encontra no estado.
- ***Entry*:** representa as ações realizadas no momento em que o objeto assume o novo estado.
- ***Exit*:** representa as ações executadas quando o objeto está mudando de estado.

Os seguintes passos são recomendados para construir um Diagrama de Máquina de Estados, a partir da identificação e definição das classes (BEZERRA, 2014):

1. Identificar os estados relevantes para os objetos da classe.
2. Identificar os eventos e as transições de estados que ele ocasiona.
3. Identificar para cada estado as transições possíveis quando um evento ocorre, respeitando as regras de negócio do contexto do sistema.
4. Identificar para cada estado as atividades internas relevantes.
5. Verificar se há fatores que influenciam nos eventos que ocasionam a transição entre os estados. Se for o caso, uma condição de guarda

deve ser definida para a transição, e verificar se alguma ação deve ser executada quando uma transição é disparada, indicando, assim, as ações internas do estado.

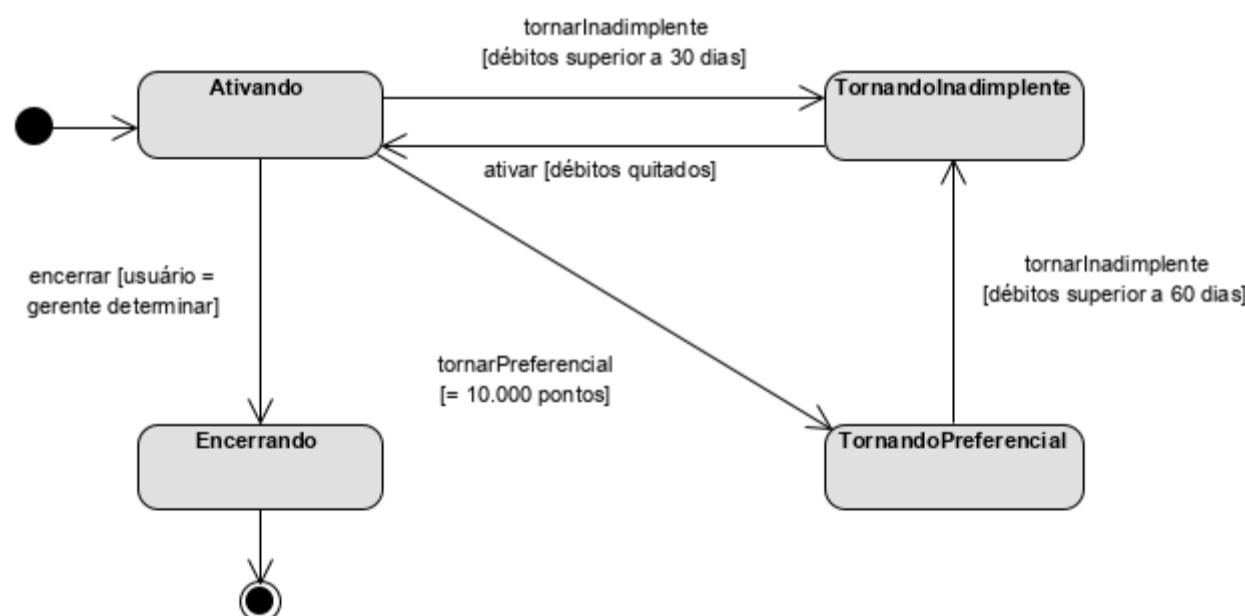
6. Identificar os atributos que estão envolvidos na indicação de cada condição de guarda das transições de estado. Pode ser que esses atributos não foram previamente identificados. Nesse caso, deve-se adicioná-los às classes correspondentes.
7. Definir o estado inicial e os eventuais estados finais.
8. Desenhar o Diagrama de Máquina de Estados, dispondo os estados de tal forma que o ciclo de vida do objeto possa ser visualizado de cima para baixo e da esquerda para a direita.

REFLITA

Sobre a principal indicação da modelagem do Diagrama de Máquina de Estados, ou seja, os objetos das classes que possuem estados relevantes, quem define quais estados dos objetos são relevantes ao contexto do domínio do sistema?

Vamos conferir a representação simplificada de um Diagrama de Máquina de Estados, ilustrado na Figura 3.3, correspondente aos objetos da classe “Cliente”, referentes a um sistema de uma loja virtual. Considerando as regras de negócio do contexto do sistema, foram definidos os estados Ativando, TornandoNadimplente, TornandoInadimplente e Encerrando. O diagrama representa estados simples apenas com o nome e suas transições de estados. Observe que, nas transições de estados, junto ao nome dos eventos, foi indicada uma condição de guarda específica às regras e particularidades do contexto do sistema, a qual indica que cada objeto só assumirá o novo estado se a condição de guarda for verdadeira.

Figura 3.3 | Diagrama de Máquina de Estados da classe Cliente

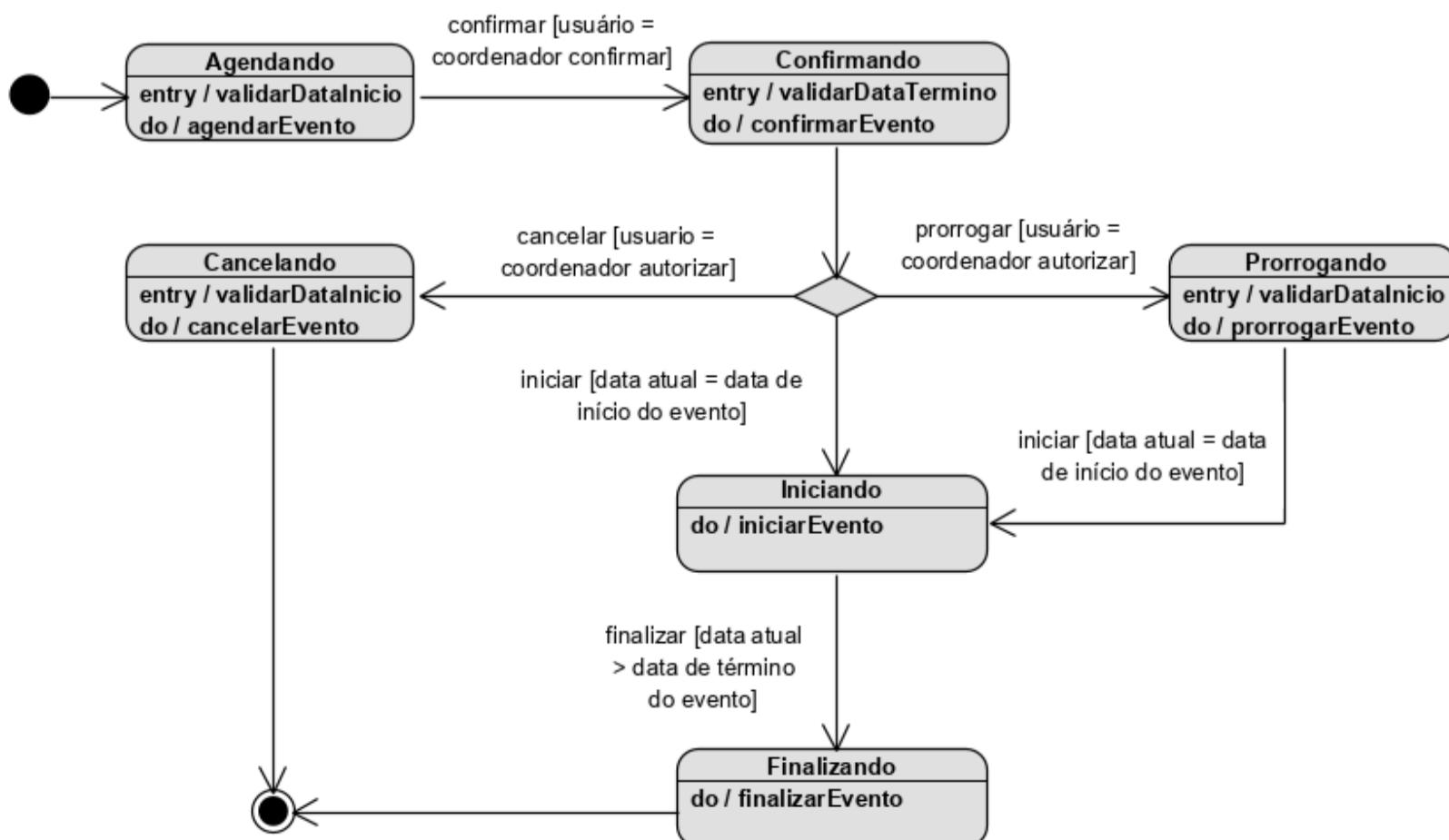


Fonte: elaborada pela autora.

Agora, confira, na Figura 3.4, o Diagrama de Máquina de Estados correspondente aos objetos da classe “Evento” com os estados “Agendando”, “Confirmando”, “Iniciando”, “Cancelando”, “Prorrogando” e “Finalizando”, referentes a um sistema de controle de eventos

científicos. Observe que todos os estados possuem a ação de estado indicada pela cláusula “*Do*”, que corresponde à operação nominada ao estado que o objeto assumirá durante a execução do sistema. Cada ação de estado do tipo “*Do*” deve ser listada como uma operação na classe especificada na versão final do Diagrama de Classes, geralmente, o da atividade de projeto. As condições de guarda indicadas nas transições de estados estão de acordo com regras do contexto do sistema, as quais devem ser implementadas posteriormente, a fim de assegurarem a consistência dos objetos do sistema. O estado inicial “Agendando” possui a ação de entrada (cláusula “*Entry*”) “validarDataInicio”, usada se no momento da instanciação de um novo evento a data inicial indicada é futura à data atual do sistema. A mesma consistência deve ser realizada quando o objeto assumir os estados “Confirmando”, “Cancelando” ou “Prorrogando”, a partir das transições de estados estabelecidas sob a condição de autorização do coordenador do evento, ou seja, as transições serão realizadas mediante um evento disparado por um ator (coordenador) que interage com o sistema. Por fim, uma vez que um objeto assumir os estados “Cancelando” ou “Finalizando”, encerra-se o ciclo da máquina de estados dos objetos da classe “Evento”.

Figura 3.4 | Diagrama de Máquina de Estados da classe Evento



Na modelagem de sistemas de informação para comércio eletrônico (*e-commerce*) de diferentes segmentos de lojas virtuais, bem como de sistemas de gerenciamento de logística para o monitoramento de entregas, é imprescindível a adoção do Diagrama de Máquina de Estados para especificar fielmente os estados e as transições de estados dos objetos e, posteriormente, implementar todas as operações correspondentes às ações de estados. Na perspectiva da visão dos usuários de um sistema em execução, a mudança de estado dos objetos reflete no gerenciamento da “situação” das transações realizadas, principalmente de forma on-line e remota. No exemplo citado de monitoramento de entregas, o próprio cliente consegue acompanhar o rastreamento de uma entrega pela sua situação, geralmente sendo: encaminhado para o centro de distribuição, em processo de coleta, recebido no centro de distribuição, transferência para a cidade destino, recebido no centro de distribuição da cidade destino, separado para o roteiro de entrega e processo de entrega e entrega realizada.

Esses são os estados de um objeto.

Considerando as técnicas de modelagem comportamentais da UML, o Diagrama de Máquina de Estados é fundamental para demonstrar o comportamento do conjunto de estados e transições de estados dos objetos das classes especificados no Diagrama de Classes, para assim compor a documentação da atividade de análise de um sistema orientado a objetos condizente com o que será implementado.

REFERÊNCIAS

BEZERRA, E. **Princípios de análise e projeto de sistemas com UML**. 3. ed. Rio de Janeiro: Elsevier, 2014.

BOOCH, G.; JACOBSON, I.; RUMBAUGH, J. **UML**: guia do usuário. 2. ed.

Rio de Janeiro: Campus, 2006.

GUEDES, G. T. A. **UML**: uma abordagem prática. 3. ed. São Paulo:

Novatec, 2018.

PRESSMAN, R.; MAXIM, B. **Engenharia de software**: uma abordagem

profissional. 8. ed. Porto Alegre: AMGH, 2018.

RUMBAUGH, J. *et al.* **Modelagem e projetos baseados em objetos**. Rio

de Janeiro: Campus, 1997.

FOCO NO MERCADO DE TRABALHO

MODELAGEM DE ESTADOS

Iolanda Cláudia Sanches Catarino

Ver anotações

ELABORAÇÃO DO DIAGRAMA DE MÁQUINA DE ESTADOS

Na elaboração do diagrama de máquina de estados, é fundamental identificar as regras de negócio aplicadas ao contexto dos objetos, a fim de auxiliar na definição dos seus estados e das suas transições.



Fonte: Shutterstock.

Deseja ouvir este material?

Áudio disponível no material digital.

Na descrição da situação-problema do módulo recepção de um sistema de hotelaria, você observou que a identificação dos requisitos funcionais, referentes aos cadastros e controles da reserva, check-in e check-out, está explícita na descrição e reflete diretamente na modelagem dos casos de uso do sistema. Assim, considerando que a UML foi adotada para modelagem do sistema, recomenda-se iniciar a modelagem comportamental da atividade de análise com o Modelo de Casos de Uso e a documentação de cada caso de uso. Na sequência, inicia-se a modelagem estrutural, elaborando o Diagrama de Classes, que é considerado a principal técnica de modelagem da UML, para especificar as classes e seus relacionamentos em diferentes perspectivas de visões e detalhes. A partir da identificação e definição das classes de objetos com os estados relevantes e diante da compreensão do contexto das regras do sistema, você deve evoluir com a modelagem comportamental de análise.

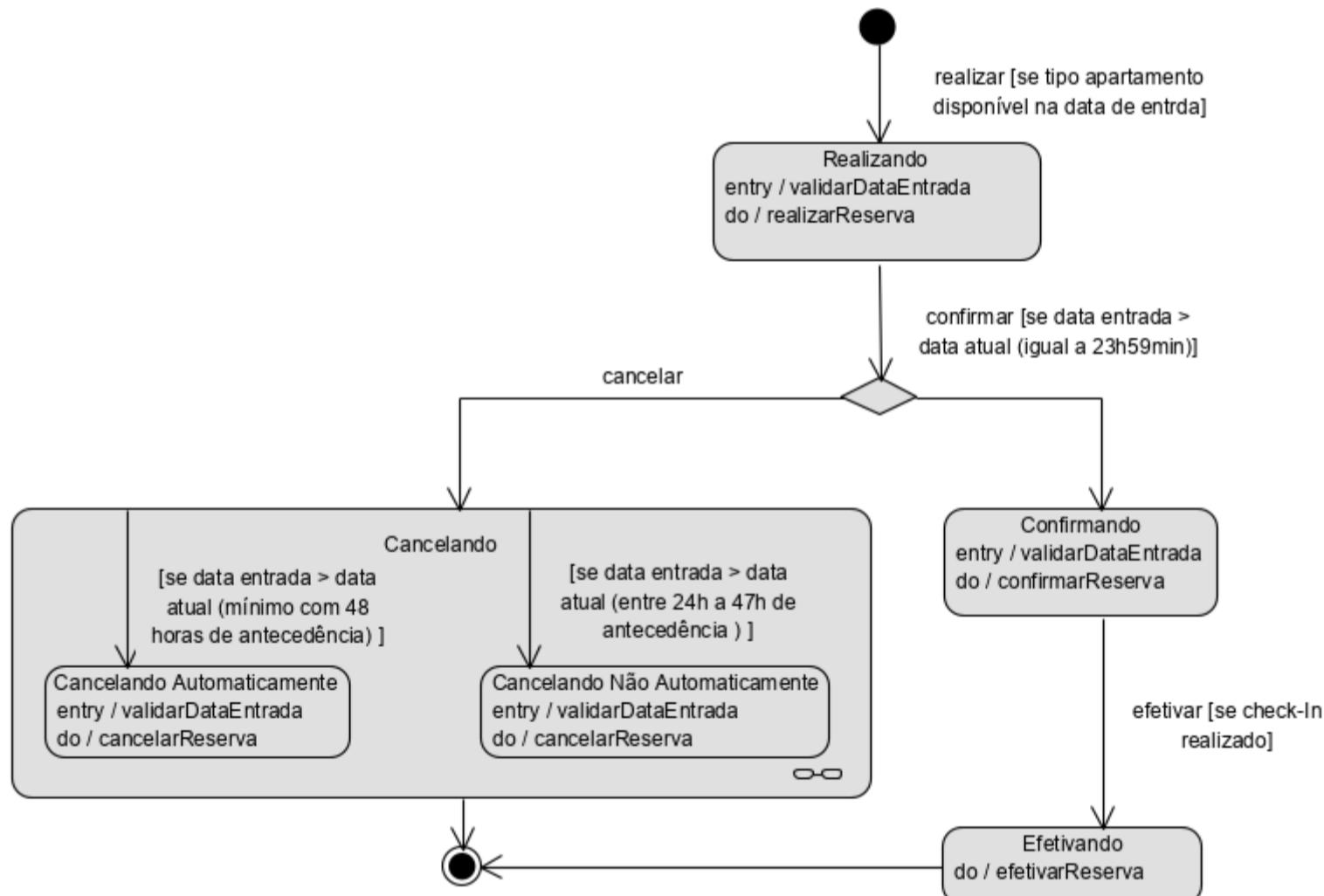
Na Figura 3.4, você conferiu a primeira versão do Diagrama de Classes com ênfase na listagem dos atributos, na qual se pode observar que as classes de objetos “Reserva”, “Hospede”, “Empresa” e CheckInOut” possuem estados relevantes, porque entre os atributos foi definido o atributo “situacao”, que consiste naquele que armazenará os valores correspondentes aos estados dos objetos dessas classes.

Considerando os estados (realizada, confirmada, cancelada ou efetivada) definidos para os objetos da classe “Reserva”, a Figura 3.5 apresenta o Diagrama de Máquina de Estados correspondente à classe “Reserva”. Para a solução proposta, foram definidas as seguintes regras de transição entre os estados:

- Uma reserva, ao ser instanciada, deve ser automaticamente definida com o estado de “Realizada” se tiver o tipo de apartamento pretendido disponível no período indicado para a reserva.

- Uma reserva “Realizada” pode ser cancelada pelo próprio hóspede, via web ou aplicativo, até 48h de antecedência, sem custos. Com menos de 48h de antecedência até 24h, se o hóspede desejar o cancelamento da reserva, deve ligar para o hotel e falar diretamente com um responsável pelo setor e este proceder com o pedido. Uma vez que a reserva for cancelada, não pode ser ativada novamente.
- Uma reserva “Realizada” assume o estado de “Confirmada” a partir de 23h59min antes da data de entrada indicada na reserva.
- Uma reserva “Confirmada” assume o estado de “Efetivada” assim que o check-in do hóspede for registrado no sistema.

Figura 3.5 | Diagrama de Máquina de Estados – Classe Reserva



Fonte: elaborada pela autora.

Observe, na figura, que foi representado um estado composto para o estado reserva “Cancelada”, considerando a regra que uma reserva pode ser cancelada automaticamente pelo hóspede ou não, mediante o prazo estipulado.

Posteriormente, na implementação do sistema, os estados relevantes definidos para os objetos das classes, respeitando as condições de guarda especificadas nos eventos que ocasionam as transições de

estados, bem como a definição das ações de estados indicadas pelas cláusulas “*Do*” e “*Entry*”, devem ser fielmente implementados para garantir a consistência e integridade das validações do sistema com sua modelagem.

NÃO PODE FALTAR

MODELAGEM DE INTERAÇÕES – DIAGRAMA DE SEQUÊNCIA

Iolanda Cláudia Sanches Catarino

Ver anotações

DIAGRAMA DE SEQUÊNCIA

O diagrama de sequência representa a ordem temporal em que as mensagens são trocadas entre os objetos envolvidos na execução de um processo.



Fonte: Shutterstock.

Deseja ouvir este material?

PRATICAR PARA APRENDER

Caro aluno, seja bem-vindo à seção sobre modelagem de interações com o diagrama de sequência!

Nesta seção vamos continuar a modelagem dinâmica do sistema para detalhar a comunicação entre os objetos do sistema a partir do diagrama de sequência, um dos diagramas de interação da UML. É preciso lembrar que os diagramas de interação representam um subgrupo dos diagramas comportamentais e visam mostrar uma interação, formada por um conjunto de objetos e seus relacionamentos, detalhando as mensagens que são enviadas entre eles. Em geral, a modelagem de um sistema tem vários diagramas de interação, e esse conjunto, com todos os diagramas de interação especificados, constitui o seu modelo de interações.

O diagrama de sequência tem o objetivo de representar a ordem temporal em que as mensagens são trocadas entre os objetos envolvidos na execução de um processo, que foi especificado como um caso de uso.

Assim, você vai conhecer e compreender os conceitos, os componentes, a notação gráfica de cada elemento do diagrama e as orientações para proceder com a construção do diagrama de sequência.

Na especificação dos requisitos funcionais do sistema, cada um deles foi representado no diagrama de casos de uso como um caso de uso e, para documentá-lo com a descrição de seu funcionamento interno, recomenda-se elaborar a descrição do cenário do caso de uso no formato de roteiro detalhado, com o relato do seu cenário principal e eventuais cenários alternativos, ou no formato simplificado. Quem define o melhor formato de documentar um caso de uso é o analista de sistemas, com sua equipe de desenvolvedores, mediante a metodologia definida pela empresa de desenvolvimento.

E será que só a documentação de cada caso de uso, no formato de roteiro, é suficiente para o entendimento de seu funcionamento interno?

Não! Não é suficiente. Além do relato de funcionamento de cada caso de uso, independentemente do formato adotado, é importante especificar de que forma os objetos do sistema interagem no funcionamento do caso de uso e quais informações devem ser enviadas em uma mensagem de um objeto para o outro e, ainda, em que ordem!

Então, a partir desta seção, você compreenderá como os diagramas de interação consolidam o entendimento sobre os aspectos dinâmicos do sistema e avançaremos na modelagem comportamental dele para a agência de estágios e intercâmbios, elaborando o diagrama de sequência para o caso de uso que representa a funcionalidade de realização de entrevista de estágio.

Vamos avançar com a modelagem comportamental do sistema de hotelaria, módulo Recepção, descrita na Seção 3.1. A partir da descrição do contexto do módulo Recepção que concentra as rotinas da recepção do hotel, incluindo o controle de reservas, vamos priorizar a modelagem

da interação que corresponde ao processo de reserva realizada por um hóspede. Assim, o relato de como funciona essa atividade de reserva, de responsabilidade da recepção do hotel, é descrito a seguir.

o

| DESCRIÇÃO DO PROCESSO DE REALIZAÇÃO DE RESERVAS

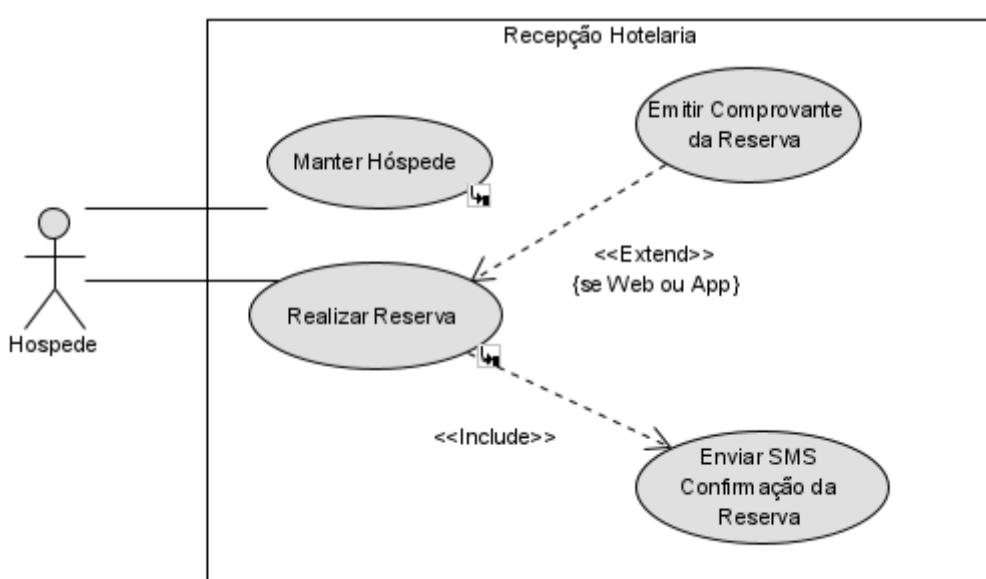
Um hóspede, para se hospedar no hotel, tem que ter um cadastro com seus dados pessoais efetuado previamente e tem que ter uma reserva prévia realizada por ele (por telefone, por aplicativo ou pela Web) ou pela empresa que o hóspede tem vínculo. Sendo a empresa responsável pela estada do hóspede, ela também é a responsável em custear integralmente as despesas referentes as diárias e as despesas referentes as refeições (ela pode custear ou não).

Uma reserva é mantida por: nome da(s) pessoa(s) que irá(ão) se hospedar em um apartamento, data que efetuou a reserva, data de entrada, hora prevista de entrada, data de saída, quantidade de adultos, quantidade de crianças, idade das crianças e nome da empresa que trabalha (se for a empresa responsável pela estada do hóspede). Um hóspede ou uma empresa pode realizar várias reservas.

A Figura 3.7 ilustra um recorte do Diagrama de Casos de Uso especificado para a atividade de Análise, ilustrando parte dos casos de uso identificados para atenderem as funcionalidades descritas no módulo de Recepção do sistema de hotelaria, correspondente ao caso de uso “Realizar Reserva”.

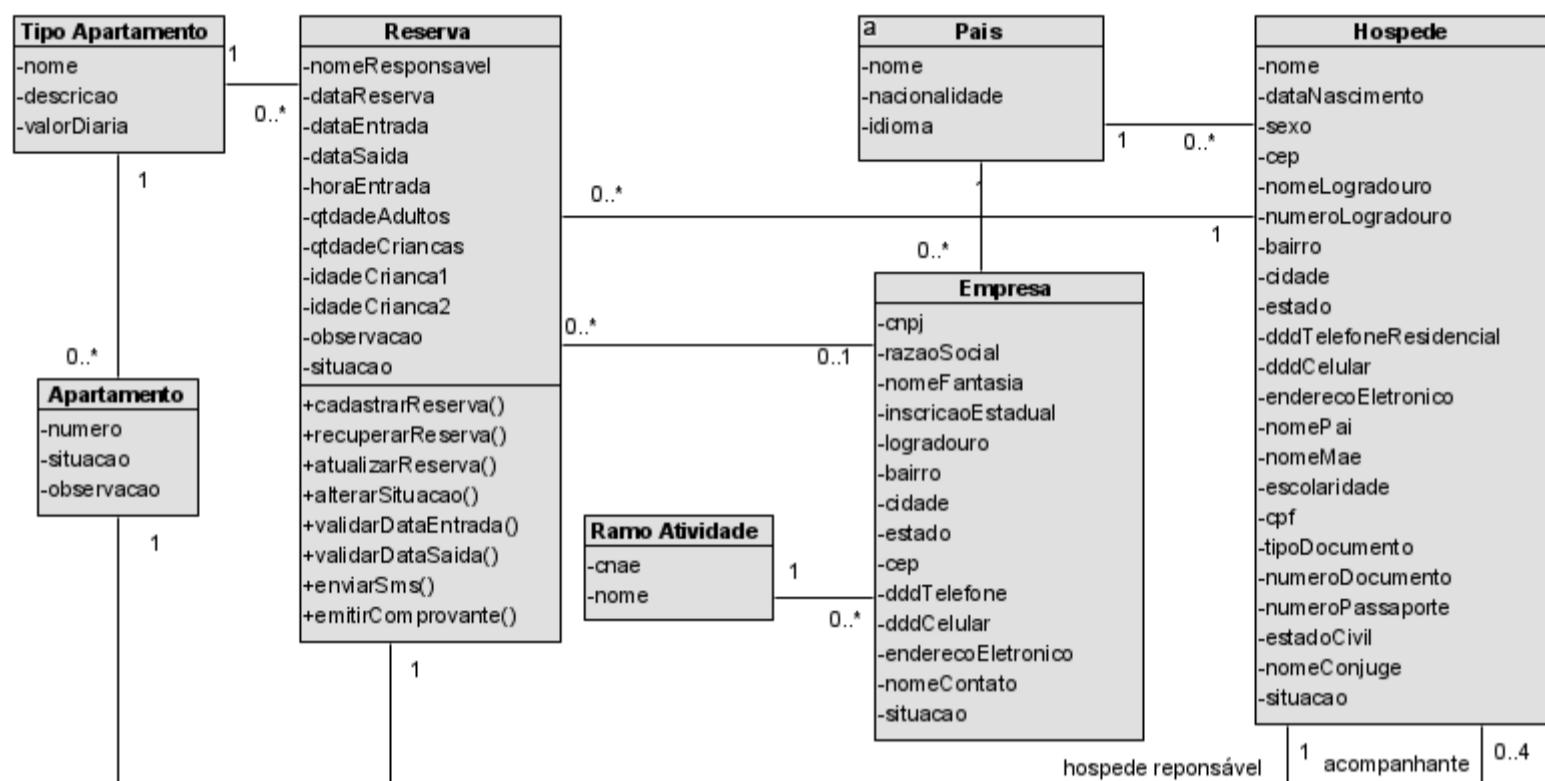
Ver anotações

Figura 3.7 | Recorte do diagrama de casos de uso



A próxima Figura apresenta um recorte do Diagrama de Classes, demonstrando as classes que foram definidas para a manipulação dos objetos “Reserva”.

Figura 3.8 | Recorte do diagrama de classes



Fonte: elaborada pela autora.

Elabore o diagrama de sequência correspondente ao caso de uso “Realizar Reserva”, representando o cenário principal e considerando as classes de objetos definidas na Figura 3.8.

Bom trabalho!

Boa leitura e bons estudos!

CONCEITO-CHAVE

Caro aluno, nesta seção avançaremos os estudos sobre a modelagem dinâmica do sistema!

Usualmente, a modelagem comportamental do sistema inicia-se com o modelo de casos de uso, o qual descreve os requisitos funcionais do sistema e os atores que interagem com o sistema, respondendo, assim, o que o sistema deve fazer e para quem. No entanto, o modelo de casos

de uso não expõe o comportamento interno das funcionalidades especificadas. Para isso, a UML contempla os diagramas de interação, que representam um subgrupo dos diagramas comportamentais.

| DIAGRAMA DE INTERAÇÃO

Os diagramas de interação mostram como os objetos do sistema agem internamente para apoiarem a realização das funcionalidades representadas pelos casos de uso, consolidando, assim, o entendimento dos aspectos dinâmicos do sistema. Os diagramas de interação da UML são: diagrama de sequência, diagrama de comunicação, diagrama de visão geral de interação e o diagrama de tempo.

Pense em como você poderá visualizar um sistema em execução e compreender a relação entre todos os objetos que participam da realização de cada funcionalidade. Uma melhor forma de representar esse funcionamento é construindo roteiros de cenários, descrevendo passo a passo a interação entre os objetos que participam da execução dos casos de uso e as mensagens que são trocadas entre eles. Outra opção é desenhar o protótipo correspondente à interface dos casos de uso para completar a compreensão dos cenários. Além dessas orientações, para se familiarizar com o funcionamento do sistema, é necessário completar e validar a modelagem de cada caso de uso com os diagramas de interação.

Segundo Guedes (2018), a interação entre objetos para dar suporte à funcionalidade de um caso de uso denomina-se realização de um caso de uso, o qual descreve o comportamento de um ponto de vista interno ao sistema, sendo que a realização de um caso de uso é representada por diagramas de interação.

Uma das formas mais utilizadas de se especificar a interação entre os objetos é a ênfase à ordenação temporal das mensagens, representando a sequência lógica da troca de mensagens formada por um conjunto de objetos e seus relacionamentos a partir da adoção do diagrama de sequência, o qual é uma forma de representar a ordem temporal em que as mensagens são trocadas entre os objetos envolvidos na execução de um processo.

■ DIAGRAMA DE SEQUÊNCIA

Conforme Guedes (2018), o diagrama de sequência descreve a ordem temporal em que as mensagens são trocadas entre os objetos envolvidos na execução de um processo que representa um caso de uso, bem como no ator responsável pela interação com os objetos.

De acordo com Booch, Jacobson e Rumbaugh (2006, p. 243), o diagrama de sequência é “um diagrama de interação que dá ênfase à ordenação temporal de mensagens. Graficamente, um diagrama de sequência é uma tabela que mostra objetos distribuídos no eixo X e mensagens, em ordem crescente no tempo, no eixo Y”.

O diagrama de sequência baseia-se no diagrama de casos de uso, em que é elaborado um diagrama de sequência para cada caso de uso, que se apoia no diagrama de classes para determinar os objetos das classes que realizam o caso de uso, com a indicação das mensagens trocadas entre os objetos, que são, na maioria das vezes, as operações das classes.

REFLITA

Os diagramas de interação representam um subgrupo dos diagramas comportamentais, enfatizando a modelagem dinâmica do sistema para demonstrarem o funcionamento interno dos casos de uso. Dessa forma, você conseguiu compreender como os diagramas comportamentais se complementam e por que devem ser consistentes com o diagrama estrutural – diagrama de classes?

Agora vamos conhecer e compreender os elementos que constituem o diagrama de sequência.

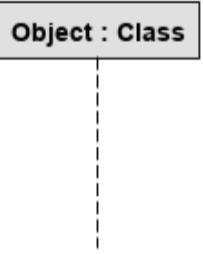
■ ELEMENTOS DO DIAGRAMA DE SEQUÊNCIA

Um diagrama de sequência é construído e representado pelos seguintes elementos: ator, mensagens, objetos, linha de vida e foco de controle. A notação gráfica dos principais elementos que podem compor o diagrama de sequência e uma breve descrição de cada elemento é apresentada no Quadro 3.3 a seguir.

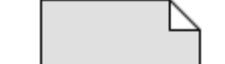
o

Ver anotações

Quadro 3.3 | Elementos do diagrama de sequência

Notação	Elemento
	Linha de vida: representa a existência de um elemento (objeto ou ator) participante da realização do caso de uso em um período de tempo. É representada por uma linha vertical tracejada abaixo do elemento, chamada de cauda.
 Actor	Autor: representa os mesmos atores já criados no diagrama de casos de uso; são apoiados por uma linha de vida e enviam mensagens para os objetos como uma forma de interação para solicitarem a execução de uma operação ou simplesmente o envio de informações. No diagrama sempre representa o ator primário responsável por enviar a mensagem inicial, que começa a interação entre os objetos.
	Objeto: representa os objetos que participam da realização do caso de uso; são também apoiados por uma linha de vida, que juntamente com os atores, formam um cabeçalho para o diagrama. Um objeto pode existir desde o início da interação ou ser criado ao longo dela. Um objeto é representado por um retângulo com um nome único, conforme o padrão da notação de objeto.

Notação	Elemento
	<p>Foco de controle: representa o período de tempo durante o qual um elemento executa uma ação, diretamente ou não. É representado por um retângulo estreito na vertical sobre a linha de vida, podendo aparecer diversas vezes ao longo dela.</p>
	<p>Mensagem síncrona: a mensagem é síncrona quando o emissor aguarda o retorno para continuar com a interação. São as mensagens comumente utilizadas no diagrama de sequência. É representada por uma linha horizontal com uma seta sólida na extremidade.</p>
	<p>Mensagem assíncrona: a mensagem é assíncrona quando o emissor continua enviando mensagens sem aguardar o retorno, com isso o elemento receptor da mensagem assíncrona não precisa atendê-la imediatamente. É representada por uma linha horizontal com uma seta aberta.</p>
	<p>Mensagem de retorno: é uma mensagem que um objeto envia ao outro em resposta à mensagem recebida após a execução de uma ação. As mensagens de retorno são representadas por uma linha tracejada com uma seta na extremidade, apontando para o elemento que recebe a resposta.</p>
	<p>Mensagem reflexiva ou automensagem: é uma mensagem indicativa de que o objeto remetente da mensagem é também o receptor. A mensagem reflexiva é representada por uma seta que sai do objeto e retorna para ele mesmo.</p>

Notação	Elemento
	<p>Mensagem construtora: indica o momento em que o objeto passa a existir no sistema, ou seja, o objeto é instanciado ao longo do processo por uma mensagem enviada. A mensagem construtora é representada por uma linha tracejada com seta na extremidade, apontando para o centro do objeto criado. O retângulo que representa o objeto é posicionado mais abaixo no diagrama.</p>
	<p>Mensagem destrutora: indica a destruição do objeto no decorrer da interação, o qual não se mostra mais necessário no processo. É representado pelo símbolo X na parte inferior da linha de vida do objeto que está sendo destruído.</p>
	<p>Quadro de interação: representa uma interação independente, possibilitando isolar um diagrama de sequência com um contexto específico, formando uma fronteira para que possa ser integrado aos demais diagramas de sequência. O quadro é representado graficamente por um retângulo com um rótulo no canto superior esquerdo, que identifica o tipo do quadro.</p>
	<p>Nota ou comentário: serve para escrever observações aos elementos que compõe o diagrama de sequência, contendo informações úteis para os desenvolvedores, porém não expressa força semântica específica aos elementos do diagrama.</p>

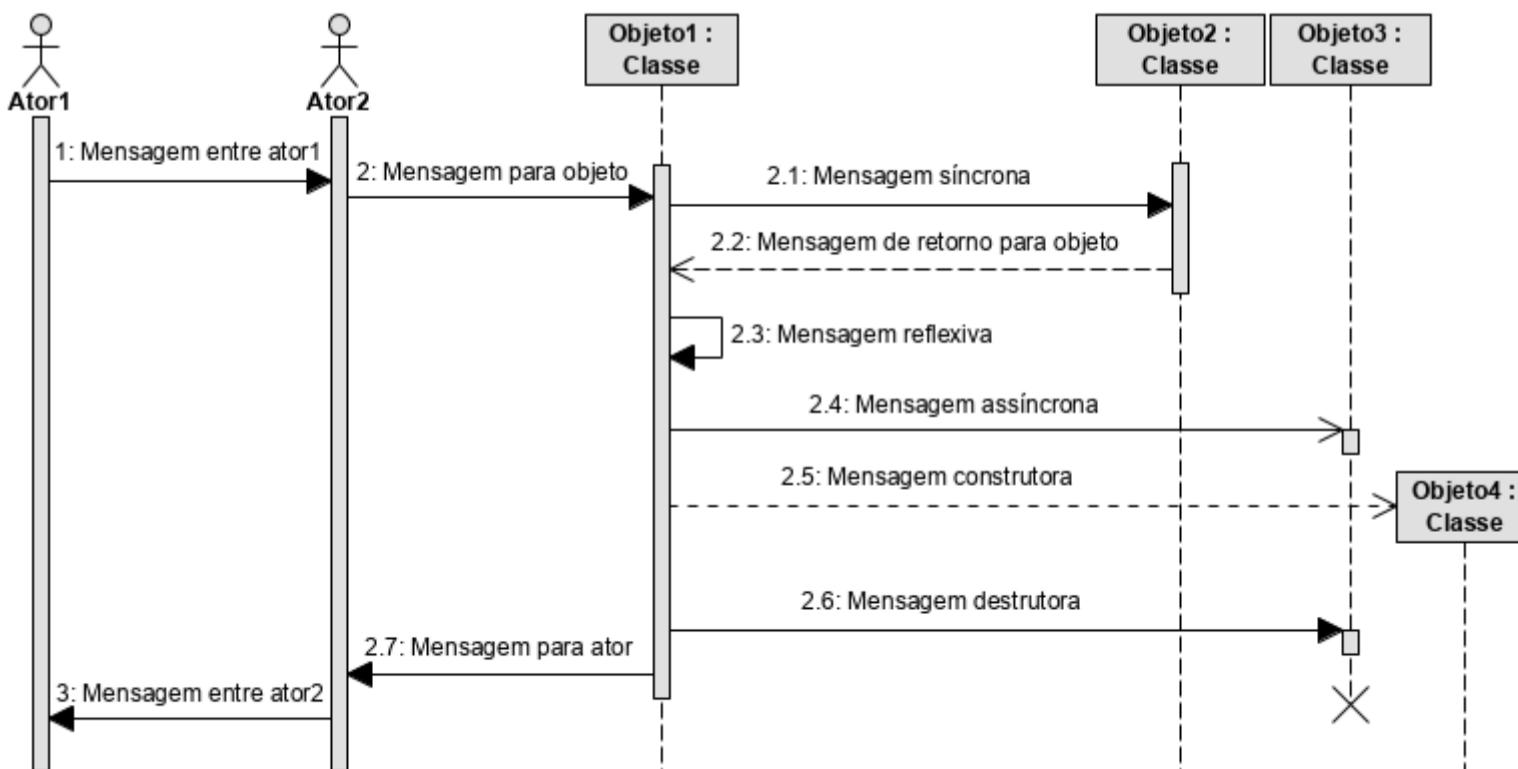
Fonte: elaborado pela autora.

Nos diagramas de interação, como no diagrama de sequência, as mensagens representam a solicitação que um elemento envia para o outro com o objetivo de executar uma ação, demonstrando a ocorrência de eventos. O objeto que envia a mensagem denomina-se objeto emissor ou remetente, e o objeto que recebe a mensagem denomina-se objeto receptor. As possíveis trocas de mensagens que podem acontecer entre os elementos do diagrama são:

- **Autor e autor:** indica uma comunicação entre os atores.
- **Autor e objeto:** geralmente o autor provoca um evento, enviando uma mensagem que dispara uma operação, contudo o autor pode simplesmente transmitir uma informação sem disparar uma operação.
- **Objeto e objeto:** indica o envio de uma mensagem, disparando uma operação. Quando um objeto envia uma mensagem para si mesmo, essa operação é denominada mensagem reflexiva.
- **Objeto e autor:** indica o retorno de uma mensagem com o seu resultado.

A Figura 3.9 ilustra a interação entre os elementos do diagrama de sequência com os tipos de mensagens possíveis.

Figura 3.9 | Elementos e tipos de mensagens do diagrama de sequência



Fonte: elaborada pela autora.

Nos diagramas de interação, as mensagens representam a solicitação que um elemento envia para o outro com o objetivo de executar uma ação, demonstrando a ocorrência de eventos. As possíveis trocas de mensagens que podem acontecer entre os elementos do diagrama de sequência são: ator e ator; ator e objeto; objeto e objeto; objeto e ator. Contudo, a troca de mensagens entre os elementos ator e objeto ou entre objeto e ator sempre deve referenciar um objeto que representa a interface do caso de uso.

Além da representação da troca de mensagens entre os elementos do diagrama de sequência, é importante indicar o conteúdo da mensagem enviada pelo remetente, o qual especifica informações a serem passadas para o elemento receptor denominado rótulo da mensagem.

Os rótulos podem indicar:

- Informações que descrevem uma solicitação ou resposta entre os elementos ator e objeto, que representa uma interface correspondente à funcionalidade do caso de uso do diagrama, como “Solicitar cadastro do candidato”; “Solicitar inscrição para vaga” ou “Consultar vaga”.
- Uma mensagem, que é uma operação de uma classe, é representada com o nome da operação mais um par de parênteses no final “nomeOperacao()”, por exemplo, inserirCandidato() ou obterVaga(); tem exatamente a mesma nomenclatura da operação listada no diagrama de classes.
- Uma mensagem, que é uma operação de uma classe, assim como a do item anterior, e que precisa representar a necessidade de um retorno ou não, mostra-se como “nomeOperacao():void”, nesse caso, o “void” é a palavra reservada para representar que não haverá retorno; ou como “nomeOperacao():Boolean”, nesse caso, o tipo de dados “Boolean” representa que, ao executar essa operação, terá como retorno o valor “Boolean”.
- Uma mensagem, que é também uma operação de uma classe, assim como a dos itens anteriores, e que precisa representar uma mensagem com passagem de parâmetro, utiliza-se da nomenclatura “nomeOperacao(atributo: tipo de dado)”, por exemplo recuperarInscricaoVaga cpf:Long).

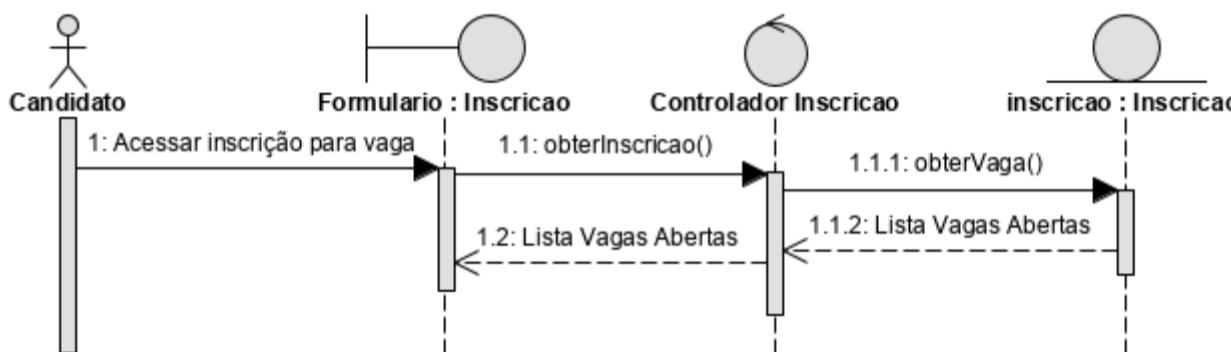
Juntamente com o rótulo da mensagem, também pode-se indicar uma condição de guarda que estabelece uma regra ou condições para que uma mensagem possa ser disparada. As condições de guarda são descritas entre colchetes na mensagem.

Na representação dos objetos do diagrama de sequência, pode-se adotar os estereótipos do diagrama de classes para melhor compreensão das interações entre os objetos, tais como: <<boundary>>, <<control>> e <<entity>>; sendo:

- **Estereótipo do tipo <<boundary>>**: denominado de classe de fronteira, é aquele que representa a interface do sistema, indicando a comunicação entre o ator primário e os demais objetos das classes que participam da interação.
- **Estereótipo do tipo <<control>>**: denominado de classe de controle, o qual serve de intermediário entre as classes definidas como <<boundary>> e <<entity>> para tratar as regras de negócio e o fluxo da aplicação.
- **Estereótipo do tipo <<entity>>**: denominado de classe de entidade, é aquele que mostra que as classes do sistema também são entidades, seja persistente ou transiente durante a execução do sistema.

A Figura 3.10 ilustra uma parte de um diagrama de sequência para exemplificar a notação gráfica dos estereótipos das classes. O elemento “Formulario:Inscricao” representa uma classe de fronteira que faz a interação com o ator “Candidato” e com o elemento “ControladorInscricao”, representando uma classe de controle responsável pela mediação do tratamento das regras de negócio com o elemento “inscrição:Inscricao”, que representa uma classe de entidade persistente na interação.

Figura 3.10 | Diagrama de sequência com estereótipos das classes



Fonte: elaborada pela autora.

FRAGMENTOS

Na representação do diagrama de sequência, também é possível utilizar **fragmentos de interação** e **fragmentos combinados** que possibilitam o alinhamento de interações, sendo que cada fragmento representa uma interação independente, formando uma fronteira entre os elementos do diagrama. Nesse sentido, é recomendável representar os fragmentos apenas se de fato contribuírem para a compreensão da interação.

Um **fragmento de interação**, também denominado de ocorrência de interação, representa a ocorrência de um outro diagrama de interação da UML, sendo que, na representação do diagrama de sequência, indica outro diagrama de sequência.

Na notação gráfica do quadro de interação, identifica-se o rótulo com a expressão “**ref**” e, no centro do quadro, descreve-se o nome do diagrama de sequência referenciado. Nos fragmentos de interação, pode-se utilizar do elemento chamado “portões (*gates*)” para relacionar

uma mensagem fora de um fragmento de interação com uma mensagem dentro do fragmento de interação, estabelecendo uma interface entre fragmentos, ou seja, um ponto de conexão.

Um **fragmento combinado** é utilizado para definir o fluxo de controle da interação, correspondendo a uma sequência de mensagens encapsuladas em um fragmento, compondo um procedimento que pode ser reutilizado em demais diagramas de sequência. Os fragmentos combinados são representados pelo elemento quadro de interação com uma identificação no rótulo, que descreve o tipo de operador de interação, que pode ser (BEZERRA, 2014):

- ***alt*: abreviatura de Alternative (Alternativa)**: modela a construção procedural do tipo se-então-senão, ou seja, uma escolha entre duas ou mais ações, sendo que o quadro de interação é dividido em partes por uma linha tracejada, representando cada operador com uma condição de guarda (texto entre colchetes que estabelece uma condição ou uma regra). Apenas um operador do quadro é executado.
- ***opt*: abreviatura de Option (Opção)**: modela a construção procedural do tipo se...então, sendo que o operador *opt* representa uma escolha de comportamento que será ou não executado a partir de uma condição de guarda.
- ***loop*: abreviatura de Looping (Laço)**: representa que uma interação deve ser realizada zero ou mais vezes conforme indicação da expressão com os limites mínimo e máximo, definindo a quantidade de repetições.

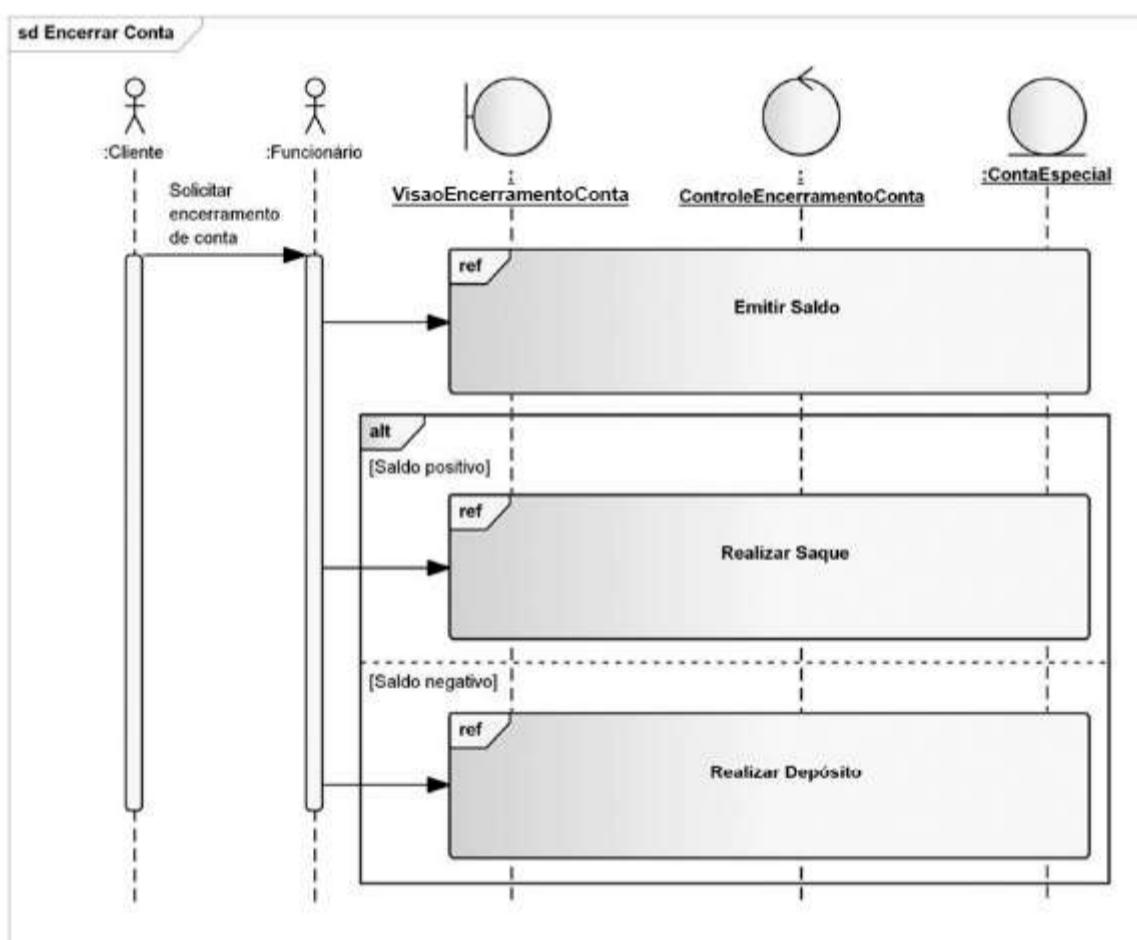
ASSIMILE

Os fragmentos do diagrama de sequência podem ser de dois tipos: fragmentos de interação e fragmentos combinados. Um fragmento de interação representa a ocorrência de um outro diagrama de sequência e o fragmento combinado é

utilizado para definir o fluxo de controle da interação, correspondendo a uma sequência de mensagens agrupadas em um fragmento, que compõe um procedimento.

A Figura 3.11 ilustra um exemplo de diagrama de sequência correspondente ao caso de uso “Encerrar Conta”, que contém o fragmento de interação “Emitir Saldo” e o fragmento combinado do tipo alternativa, contendo dois fragmentos de interação – “Realizar Saque” e “Realizar Depósito”.

Figura 3.11 | Exemplo de diagrama de sequência com fragmentos



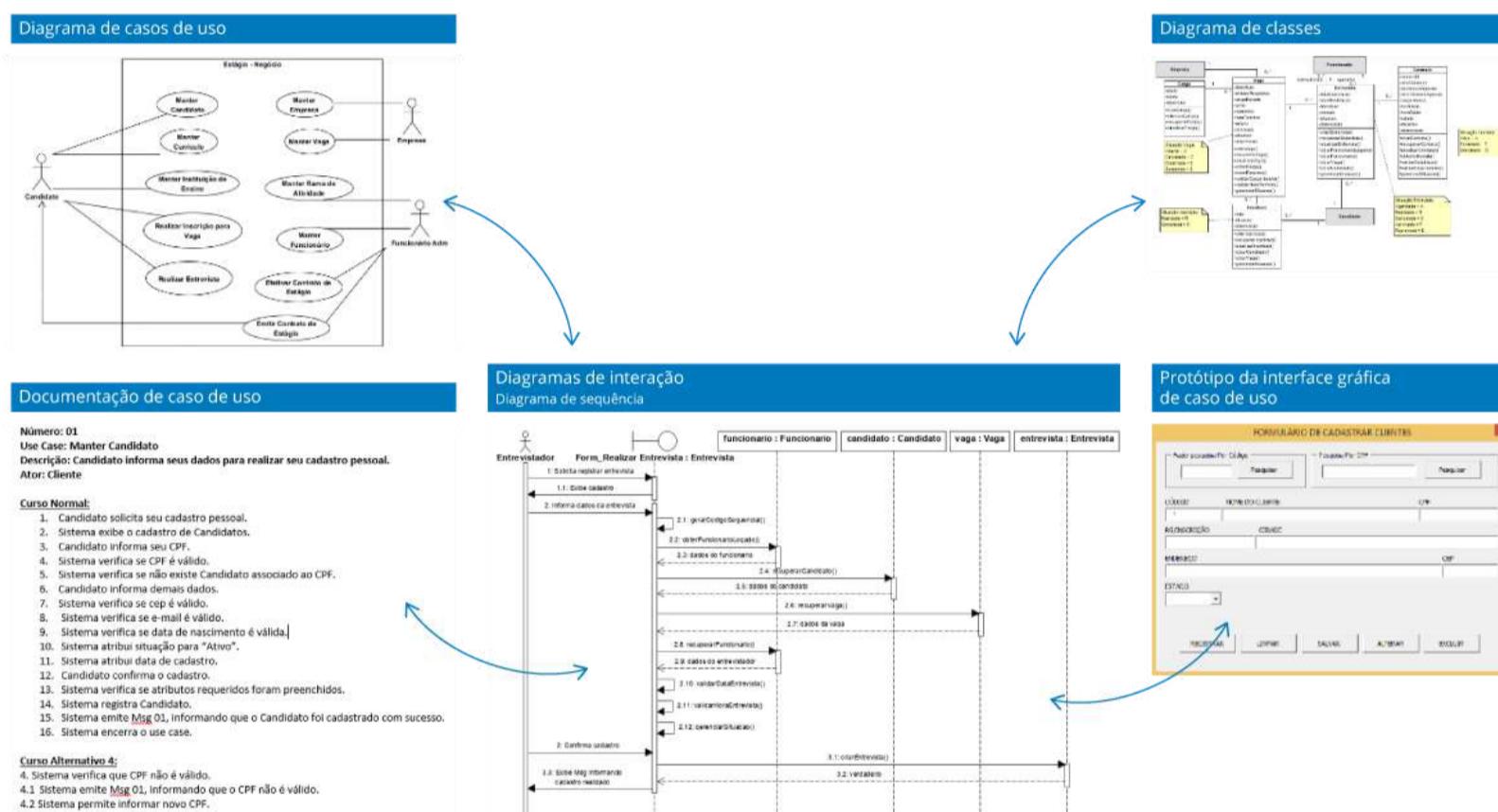
Fonte: Guedes (2018, p. 208).

ELABORANDO O DIAGRAMA DE SEQUÊNCIA

Para elaborar um diagrama de interação, é necessário um diagrama de casos de uso com a descrição do roteiro dos cenários dos casos de uso e também do diagrama de classes. Em geral, durante a construção de um diagrama de interação, é comum identificar novas classes, atributos e principalmente a definição das operações, bem como uma descrição refinada dos cenários dos casos de uso. Dessa forma, os diagramas de interação da UML reforçam o apoio ao desenvolvimento incremental de sistemas orientados a objetos a partir de modelos que podem evoluir com a inclusão de novos detalhes.

A Figura 3.12 representa a relação entre o diagrama de casos, com sua documentação, e um projeto do protótipo da interface gráfica correspondente ao caso de uso descrito, além do diagrama de classes, cuja finalidade é identificar a classe ou classes de objetos que participam da realização de cada caso de uso para, assim, construir o diagrama de interação pretendido, especificando a troca de mensagens entre os objetos.

Figura 3.12 | Relação entre os modelos comportamental e estrutural



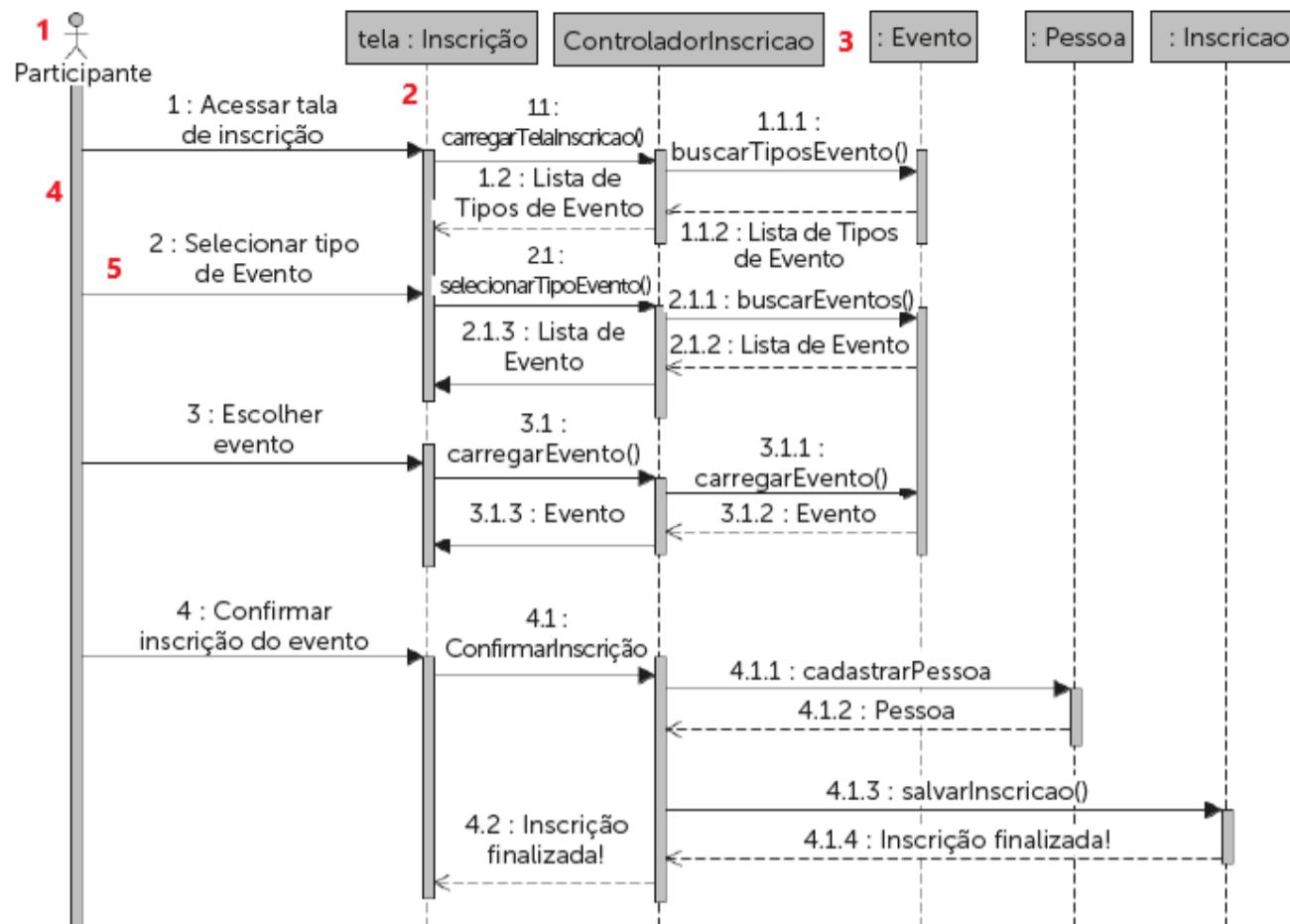
Fonte: elaborada pela autora.

Os seguintes passos são recomendados para construir o diagrama de sequência correspondente a cada caso de uso:

1. Utilizar os roteiros dos cenários da documentação do caso de uso para identificar os objetos que participam da realização do caso de uso.
2. Analisar o diagrama de classes para definir as classes correspondentes aos objetos identificados no item 1.
3. Rever os roteiros dos cenários da documentação do caso de uso para identificar as mensagens que são trocadas entre os objetos das classes definidas.
4. Analisar as operações das classes definidas e ser consistentes com as mensagens identificadas no item 3.
5. Elaborar o diagrama de sequência, compondo o cabeçalho do diagrama com o ator primário, que interage com o caso de uso e com os objetos das classes definidas no item 2, e, se necessário, particionar o diagrama em fragmentos de interação e/ou fragmentos combinados.
6. Verificar a consistência do diagrama de sequência com o diagrama de casos de uso e, a partir do conhecimento adquirido com a construção do diagrama de interação, se necessário, aperfeiçoar a descrição dos cenários.
7. Verificar a consistência do diagrama de sequência com o diagrama de classes e, a partir do conhecimento adquirido com a construção do diagrama de interação, se necessário, aperfeiçoar as classes de objetos que integram o diagrama com novos atributos, operações e associações. Geralmente a ferramenta CASE, com a qual o diagrama foi modelado, atualiza automaticamente o diagrama de classes, adicionando as operações às classes dos objetos que foram complementadas no diagrama.

A Figura 3.13 ilustra um exemplo de diagrama de sequência correspondente ao cenário principal do caso de uso Realizar Inscrição do Evento com a indicação de seus elementos. O elemento 1 representa o ator “Participante”, já indicado no diagrama de casos de uso. O elemento 2 representa o elemento linha de vida, que acompanha cada objeto ou ator do diagrama. O elemento 3 representa o objeto “Evento” e, na sequência, os demais objetos “Pessoa” e “Inscrição”. O elemento 4 representa o foco de controle sobre a linha de vida do ator “Participante”. O elemento 5 representa uma mensagem enviada pelo ator participante e recebida pelo objeto “tela : Inscrição”, que dispara mensagens simples e mensagens que invocam operações, sendo a mensagem identificada pela numeração 3.1 “carregarEvento()”, enviada pelo objeto “tela : Inscrição” para o objeto “ControladorInscrição”, é uma mensagem síncrona que dispara a operação “carregarEvento()”.

Figura 3.13 | Exemplo de diagrama de sequência

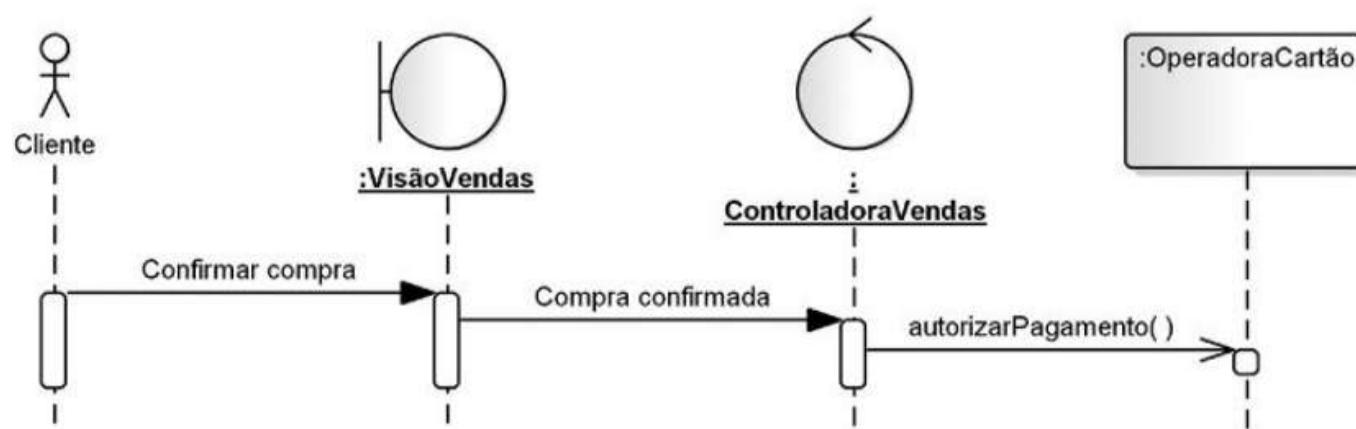


Fonte: elaborada pela autora.

EXEMPLIFICANDO

Nos diagramas de interação, a mensagem assíncrona ocorre quando o emissor continua enviando mensagens sem aguardar o retorno. Com isso, o elemento receptor da mensagem assíncrona não precisa atendê-la imediatamente. A Figura 3.14 ilustra um exemplo de mensagem assíncrona representada em uma parte de um diagrama de sequência, cujo objeto ControladoraVendas envia a mensagem assíncrona autorizarPagamento() para a classe OperadoraCartão e não aguarda um retorno dessa mensagem para continuar com o processo.

Figura 3.14 | Exemplo de mensagem assíncrona



Fonte: Guedes (2018).

Bezerra (2014) afirma que a modelagem dinâmica de um sistema não é algo simples e que, em sistemas complexos, há vários caminhos que podem ser adotados para definir a execução do sistema. Nesse contexto, a melhor alternativa é trabalhar iterativamente, ou seja, elaborar os modelos de casos de uso e de classes inicialmente e, posteriormente, desenvolver os modelos de interações e de estados para só então retornar aos modelos iniciais e verificar a consistência entre os modelos.

o

Ver anotações

REFERÊNCIAS

BEZERRA, E. **Princípios de análise e projeto de sistemas com UML**. 3. ed. Rio de Janeiro: Elsevier, 2014.

BOOCH, G.; JACOBSON, I.; RUMBAUGH, J. **UML**: guia do usuário. 2. ed.
Rio de Janeiro: Campus, 2006.

DRLJAČA, D.; LATINOVIĆ, B.; STARČEVIĆ, D. Modelling the Process of Is Auditing in the Public Administration Using Uml Diagrams. **Journal of Information Technology & Applications**, [S. l.], v. 7, n. 1, p. 32-41, 2017.

Disponível em: <https://bit.ly/357w9Pz>. Acesso em: 21 maio 2020.

FOWLER, M. **UML essencial**: um breve guia para a linguagem-padrão de modelagem de objetos. 3. ed. Porto Alegre: Bookman, 2005.

GUEDES, G. T. A. **UML**: uma abordagem prática. 3. ed. São Paulo: Novatec, 2018.

FOCO NO MERCADO DE TRABALHO

MODELAGEM DE INTERAÇÕES – DIAGRAMA DE SEQUÊNCIA

Maurício Acconcia Dias

Ver anotações

CONSTRUÇÃO DO DIAGRAMA DE SEQUÊNCIA

O diagrama de sequência baseia-se no diagrama de casos de uso, elaborando normalmente um diagrama de sequência para cada caso de uso e apoiando-se no diagrama de classes para determinar os objetos das classes envolvidas no processo.



Fonte: Shutterstock.

Áudio disponível no material digital.

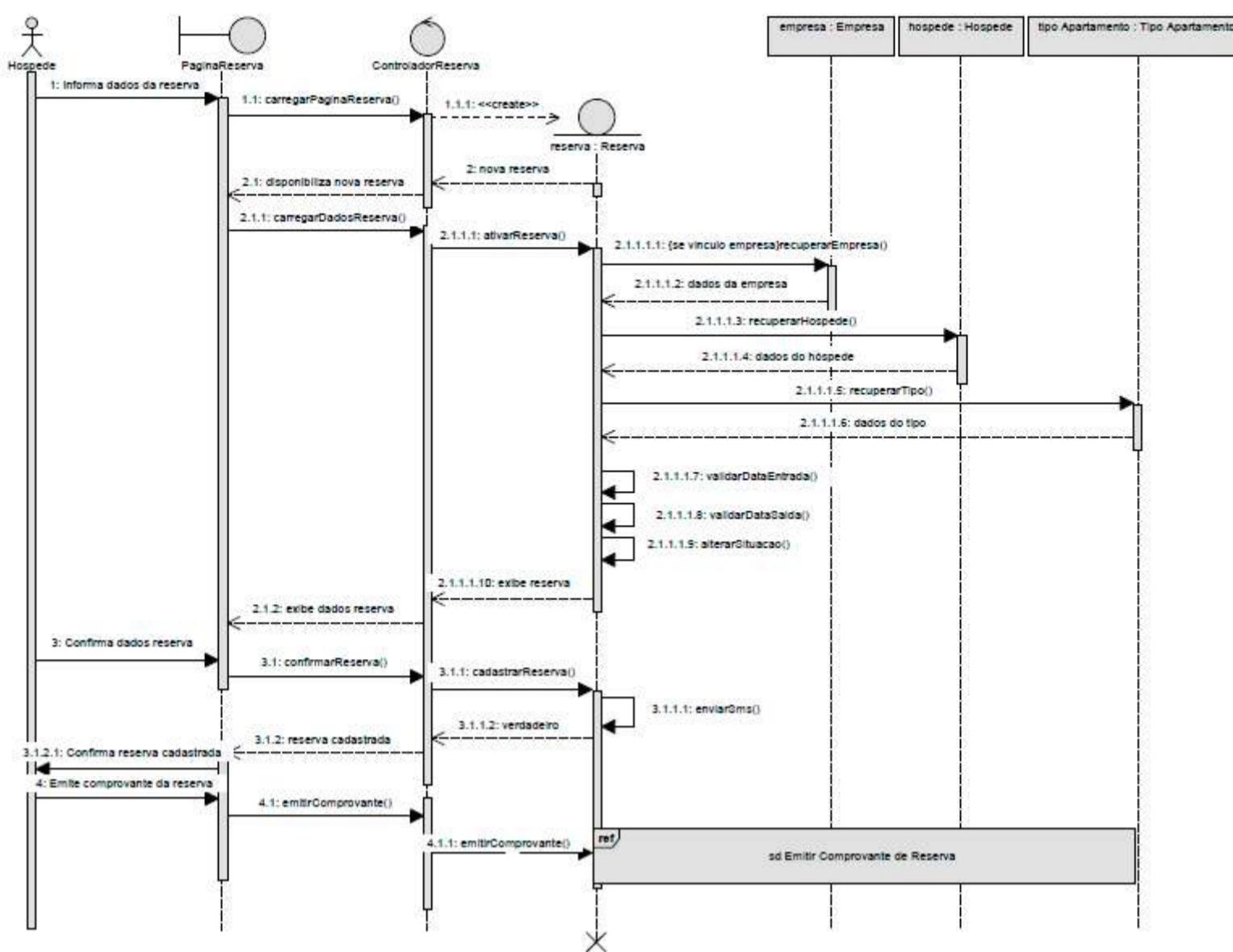
SEM MEDO DE ERRAR

Na descrição da situação-problema, referente à funcionalidade para realização de reservas do módulo Recepção de um sistema de hotelaria, você observou, no recorte do diagrama de casos de uso, a representação do caso de uso “Realizar Reserva”, que interage com o ator primário “Hóspede” e está relacionado com o caso de uso estendido “Emitir Comprovante da Reserva”, o qual subentende que a partir das reservas efetivadas, pode-se emitir o comprovante da reserva. Se o hóspede quiser emitir-lo, sendo uma opção, contudo, para toda reserva registrada, o sistema envia um SMS de confirmação da reserva, indicado pelo relacionamento do tipo inclusão “Enviar SMS Confirmação da Reserva”, obrigatoriamente.

Para elaborar o diagrama de sequência, também foi necessário analisar as classes definidas no diagrama de classes para, assim, identificar os objetos que participam da realização do caso de uso. Como recomendação da UML, para facilitar a construção do diagrama de sequência, é possível elaborar a descrição do cenário de cada caso de uso no formato de roteiro principal e de roteiros alternativos. Outra forma é também desenhar o protótipo da interface do caso de uso.

A Figura 3.15 apresenta o diagrama de sequência referente ao cenário principal do caso de uso “Realizar Reserva”.

Figura 3.15 | Diagrama de sequência (cenário principal)



Fonte: elaborada pela autora.

O diagrama integra os elementos ator primário “Hospede”, que é o responsável por fornecer os dados da reserva a partir da interação com um ator secundário, funcionário da recepção do hotel (o qual não foi demonstrado no diagrama, pois atores secundários são opcionais na representação); o objeto “PaginaReserva” que representa o formulário da interface gráfica correspondente ao caso de uso, o objeto “ControladorReserva”, responsável pela interação intermediária entre o objeto “PaginaReserva” do tipo fronteira (*boundary*) com os demais objetos do tipo entidade (*entity*), para tratar as regras de negócio e o fluxo da aplicação; os objetos “empresa:Empresa”, “hospede:Hospede” e “tipoApartamento:TipoApartamento” que participam da realização do caso de uso, e o objeto “reserva:Reserva” que é criado ao receber uma mensagem construtora “create”. No eixo Y do diagrama, foram demonstradas as trocas de mensagens entre os objetos da interação e, no final, foi inserido um fragmento de interação, que representa a

opção de estender a execução do caso de uso “Emitir Comprovante da Reserva”, conforme foi estabelecido um relacionamento de extensão para o caso de uso “Realizar Reserva”.

NÃO PODE FALTAR

MODELAGEM DOS DEMAIS DIAGRAMAS DE INTERAÇÃO

Iolanda Cláudia Sanches Catarino

Ver anotações

DIAGRAMAS DE INTERAÇÃO

Os diagramas de interação (sequência, comunicação, visão geral de interação e tempo) mostram a interação de como os objetos do sistema agem internamente para apoiarem a realização das funcionalidades representadas pelos casos de uso.



Fonte: Shutterstock.

Deseja ouvir este material?

Áudio disponível no material digital.

Caro aluno, seja bem-vindo à seção sobre a modelagem dos demais diagramas de interações da *Unified Modeling Language* (UML).

Nesta seção, serão apresentados outros diagramas da UML 2.5 para a modelagem comportamental do sistema, mostrando a modelagem com outros diagramas de interação. Assim, serão apresentados o objetivo e a notação gráfica dos elementos que compõem o diagrama de comunicação, o diagrama de visão geral de interação e o diagrama de tempo.

O **diagrama de comunicação** representa o relacionamento entre os objetos que participam da execução de um processo, o qual caracteriza a realização de um caso de uso a partir da troca de mensagens. Até a versão 1.5 da UML, o diagrama de comunicação era conhecido como diagrama de colaboração. Se você modelou as colaborações entre os objetos a partir do diagrama de estrutura composta, que é um diagrama estrutural da UML, é importante que você comece a compreender o comportamento de um ou mais casos de uso, pois isso facilitará a descrição da interação entre os objetos que realizam um caso de uso.

Vamos conferir um exemplo. Considere:

- O caso de uso “Realizar Reserva”, que corresponde a: hóspede informa os dados para realização da reserva de apartamento em um período específico.
- A precondição: para um hóspede realizar uma reserva, seja por telefone ou diretamente no site ou aplicativo do hotel, ele já deve ter seu cadastro efetivado com seus dados pessoais.
- No diagrama de casos de uso foi representado um caso de uso “Realizar Reserva” com a interação do ator primário “Hóspede”. Então, analisando o contexto do domínio do sistema, você deve identificar primeiramente qual classe ou quais classes deve-se definir para tratar a manipulação do objeto reserva no momento que se executa o caso de uso “Realizar Reserva”. Ainda, o que seria

essa manipulação do objeto? A manipulação do objeto é sempre considerada o principal objetivo do caso de uso, assim, nesse exemplo, se o caso de uso refere-se à funcionalidade de reservar um apartamento, então é exatamente esse o objetivo principal desse caso de uso: efetivar/cadastrar/criar uma reserva de apartamento, o qual caracteriza a operação de inclusão; um objetivo secundário desse caso de uso seria, a partir da reserva confirmada, em um outro momento, a possibilidade de acessá-la e de alterar a data de entrada ou saída, o que caracterizaria a operação de alteração.

- No diagrama de classes, considera-se que as classes já foram identificadas e representadas com seus atributos, operações básicas e relacionamentos. Então, este é o momento em que você deve analisar as classes e definir quais objetos da classe ou classes colaboram entre si durante a execução do caso de uso exemplificado como “Realizar Reserva”.
- Com essa análise da colaboração dos objetos das classes, define-se que a classe que manterá as instâncias reservas é a classe “Reserva”; além disso, toda reserva corresponde a um hóspede, então, cada objeto “Reserva” faz referência a um objeto da classe “Hóspede” e toda reserva corresponde a um tipo de apartamento reservado, assim faz-se referência também ao objeto da classe “TipoApartamento”.
- Com isso, define-se que há comunicação entre os objetos das classes “Reserva”, “Hospede” e “TipoApartamento”, sendo que essa comunicação deve ser especificada no diagrama de comunicação a partir de um relacionamento chamado “Vínculo”, estabelecido entre os objetos dessas classes, além da indicação da troca de mensagens entre esses objetos para representar a realização (execução) do caso de uso “Realizar Reserva”.

O **diagrama de visão geral de interação** é um novo diagrama da UML 2.0. Ele é uma variação do diagrama de atividades, o qual integra diferentes tipos de diagramas de interação com o objetivo de dar uma visão geral dos diversos cenários que descrevem a realização de um caso de uso. Também pode ser utilizado para demonstrar uma visão geral do sistema ou de processos que interagem entre si a partir de um fluxo, sendo que o principal diagrama de interação que integra os quadros que compõem o diagrama de visão geral de interação é o diagrama de sequência. Assim, aproveitando o mesmo exemplo do caso de uso “Realizar Reserva”, pode-se adotar o diagrama de visão geral de interação e, em um único diagrama, representar todo o processo que envolve a realização de uma reserva, ou seja, desde o hóspede efetuando o seu cadastro e realizando a sua reserva até ele finalizando com a emissão de um comprovante da efetivação da reserva.

E, para concluir o estudo dos diagramas de interação, vamos fazer uma contextualização sobre o **diagrama de tempo**, que também é um novo diagrama da UML 2.0. Ele é indicado principalmente para modelar a mudança de estados dos objetos com a indicação exata da unidade de tempo (segundos, minutos, horas e dias) e da duração de cada transição de estado. Por isso, esse diagrama é recomendado para modelar sistemas de automação, sistemas multimídia, sistemas de tempo real, aplicações Web, processos de rede cuja sincronização de eventos é necessário especificar etc. Assim, para modelar os estados de objetos e suas transições de sistemas de informação, recomenda-se utilizar o diagrama de máquina de estados, considerando que a precisão do tempo não é tão relevante para ser especificada.

Para reforçar a sua aprendizagem, vamos avançar com a modelagem comportamental do Sistema de Hotelaria - Módulo Recepção já descrito na Seção 3.1.

Vamos enfatizar a interação referente ao processo de realização de uma reserva, considerando o caso de uso “**Realizar Reserva**” e a descrição do relato de como funciona o procedimento de uma reserva no hotel e

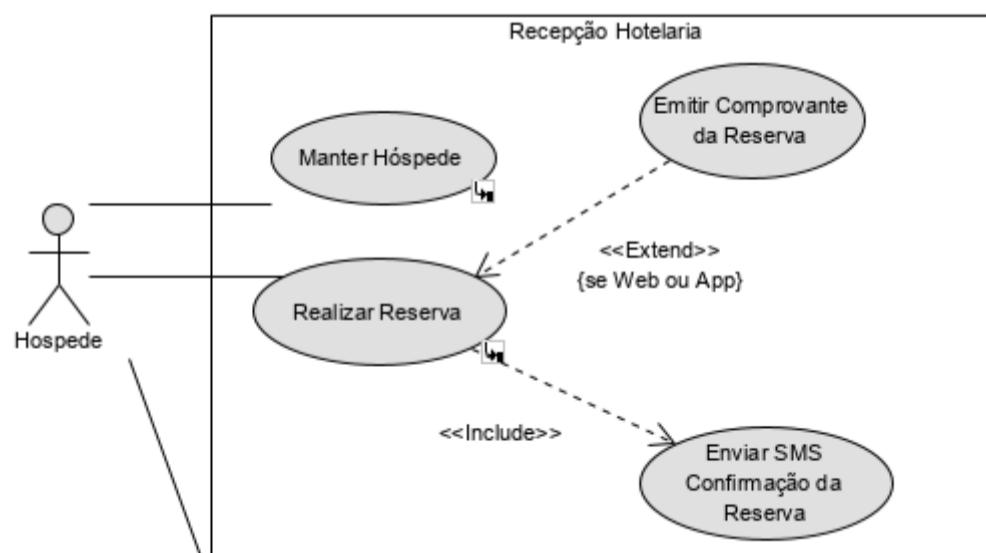
suas regras de negócio.

|| DESCRIÇÃO PROCEDIMENTO DE UMA RESERVA NO HOTEL

Um hóspede, para se hospedar no hotel, tem de ter um cadastro com seus dados pessoais (dados essenciais) efetuado previamente e uma reserva prévia realizada por ele (por telefone, por aplicativo, pela Web ou pela empresa com a qual o hóspede tem vínculo). Sendo a empresa responsável pela estada do hóspede, ela é a responsável por custear integralmente as despesas referentes às diárias e as referentes às refeições, que ela pode custear ou não. Uma reserva é mantida por: nome da(s) pessoa(s) que irá(ão) se hospedar em um apartamento, data em que efetuou a reserva, data de entrada, hora prevista de entrada, data de saída, quantidade de adultos, quantidade de crianças, idade das crianças e nome da empresa em que trabalha (caso a empresa seja responsável pela estada do hóspede). Um hóspede ou uma empresa podem realizar várias reservas.

Agora elabore o **diagrama de comunicação** referente ao caso de uso **“Realizar Reserva”**, específico para criação/cadastro da reserva. As figuras a seguir correspondem a um recorte do diagrama de casos de uso e do diagrama de classes para sustentar a elaboração do diagrama de comunicação.

Figura 3.16 | Recorte do diagrama de casos de uso

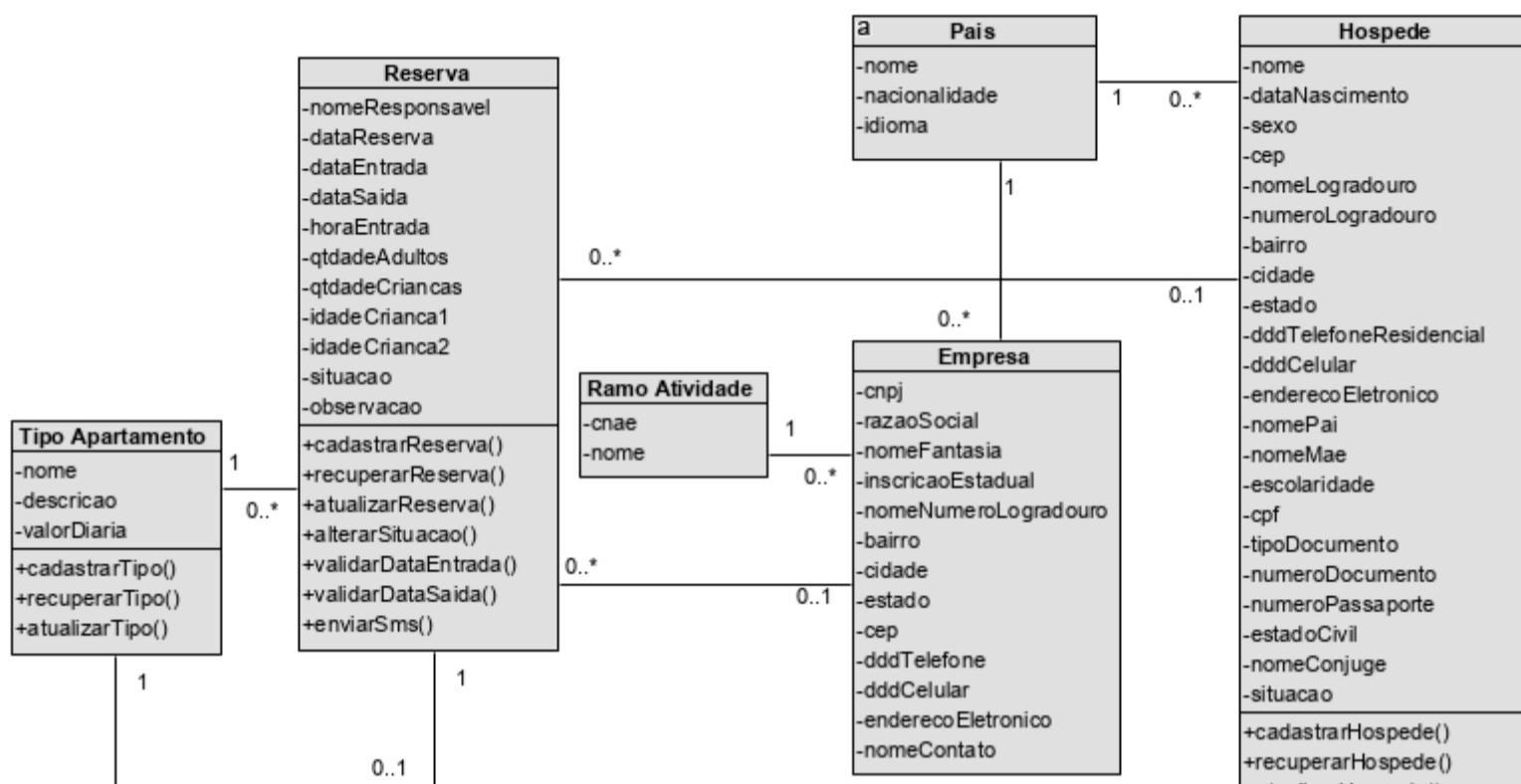


Fonte: elaborada pela autora.

A próxima figura apresenta um recorte do diagrama de classes, demonstrando as classes que colaboram para realização do caso de uso “Realizar Reserva”.

0

Figura 3.17 | Recorte do diagrama de classes



Fonte: elaborada pela autora.

Agora é com você!

Vamos avançar com a modelagem dinâmica do sistema, conhecendo os demais diagramas de interação da UML.

Além do diagrama de sequência, o diagrama de comunicação, o diagrama de visão geral de interação e o diagrama de tempo são classificados como os diagramas de interação que enfatizam a interação entre os objetos para a realização de um processo, geralmente um caso de uso.

A seguir são apresentados o objetivo, a notação gráfica dos elementos que compõem os diagramas e os exemplos para ilustrar a aplicabilidade de cada um deles. Você perceberá que não existe uma regra na UML indicando uma obrigatoriedade ou qual é o melhor diagrama de interação a ser utilizado para modelagem das interações entre os objetos; você, como analista de sistemas ou como desenvolvedor, é que deve decidir qual é a melhor opção a ser adotada em consonância com o domínio do sistema ou da aplicação.

DIAGRAMA DE COMUNICAÇÃO

A partir da UML 2.0, o diagrama de colaboração passou a ser denominado diagrama de comunicação. Esse diagrama está totalmente vinculado ao diagrama de sequência, apresentado na Seção 3.2 desta unidade, sendo considerado uma demonstração do diagrama de sequência em uma outra perspectiva.

Assim, o diagrama de comunicação representa o relacionamento entre os objetos envolvidos na realização de um caso de uso, enfatizando o sentido da troca de mensagens entre os objetos que participam de uma interação.

Diferentemente do diagrama de sequência, que demonstra a ordem temporal das mensagens trocadas entre os objetos, representando o diagrama em dois eixos e dispendendo a troca das mensagens sequencialmente de cima para baixo, o diagrama de comunicação não demonstra a temporalidade da realização de um processo. A leitura do diagrama de comunicação deve ser conduzida pela ordem de envio de mensagens entre os objetos, acompanhando o rótulo das mensagens, que descreve uma expressão de sequência obrigatória, incluindo a numeração da mensagem, as informações enviadas e também um elemento de controle, como uma condição de guarda, se necessário. O sentido da mensagem é indicado por uma seta posicionada próxima ao rótulo da mensagem, apontando para o objeto receptor da mensagem.

De acordo com Booch, Rumbaugh e Jacobson (2006), o diagrama de comunicação enfatiza a organização estrutural dos objetos que participam de uma interação a partir da indicação das mensagens enviadas e recebidas com uma identificação numeral que define a ordem temporal das mensagens. Segundo Guedes (2018), o diagrama de comunicação complementa o diagrama de sequência, concentrando-se na representação de como os elementos do diagrama estão vinculados e na ocorrência das mensagens que esses elementos trocam

entre si durante a execução de um processo, normalmente baseado em um caso de uso, não se preocupando com a temporalidade do processo. Para Bezerra (2014), o diagrama de comunicação representa os objetos relevantes, suas ligações e as mensagens trocadas entre os objetos para a realização de um caso de uso.

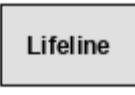
o

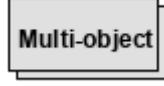
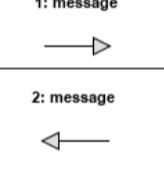
Ver anotações

ELEMENTOS DO DIAGRAMA DE COMUNICAÇÃO

O diagrama de comunicação apresenta elementos em comum com o diagrama de sequência, contudo não suporta os fragmentos de interação e os fragmentos combinados. A notação gráfica dos principais elementos que podem compor o diagrama de comunicação e uma breve descrição de cada um é apresentada no Quadro 3.4 a seguir.

Quadro 3.4 | Elementos do diagrama de comunicação

Notação	Elemento
 Actor	Autor: representa os mesmos atores já criados no diagrama de casos de uso e também reutilizados no diagrama de sequência; no entanto, não são apoiados por uma linha de vida. Um ator envia mensagens para os objetos como uma forma de interação para solicitarem a execução de uma operação ou para simplesmente enviarem informações. O diagrama sempre representa o ator primário responsável por enviar a mensagem inicial que começa a interação entre os objetos.
 Lifeline	Linha de vida (Lifeline): representa a existência de um elemento participante (objeto) da interação, geralmente uma instância de uma classe. No diagrama de comunicação, o elemento <i>lifeline</i> não tem uma linha vertical tracejada abaixo do elemento, nem foco de controle, como no diagrama de sequência. É representada graficamente por um retângulo com um nome que descreve o objeto.

Notação	Elemento
	<p>Multiobjeto: representa uma coleção de objetos de uma mesma classe, participando da interação. O elemento multiobjeto pode ser utilizado para representar o lado “muitos” de uma associação do tipo “um para muitos” ou para representar uma lista de objetos formada no momento da interação. É representado graficamente por dois retângulos sobrepostos, sendo que o nome do multiobjeto é apresentado no retângulo que fica por cima.</p>
	<p>Vínculo: ou também denominado ligação, representa uma ligação entre duas <i>lifelines</i>; corresponde a uma instância de relacionamento (associações ou dependência) estabelecido no diagrama de classes. O elemento vínculo também é estabelecido entre os elementos ator e <i>lifelines</i>. É representado graficamente uma linha entre dois objetos ou entre o ator e os objetos.</p>
	<p>Mensagem: representa a solicitação que um elemento envia para o outro com o objetivo de executar uma ação entre os elementos do diagrama, em geral, demonstrando a chamada de operações dos objetos. Para indicar as mensagens, deve-se primeiramente estabelecer o vínculo entre as <i>lifelines</i>, sendo que em um único vínculo pode haver várias mensagens. Apresenta o mesmo sentido das mensagens no diagrama de sequência. É representada por uma pequena linha horizontal com uma seta fechada na extremidade da direção do elemento receptor da mensagem. Para indicar uma mensagem de retorno, representa-se uma seta aberta, porém não tracejada.</p>

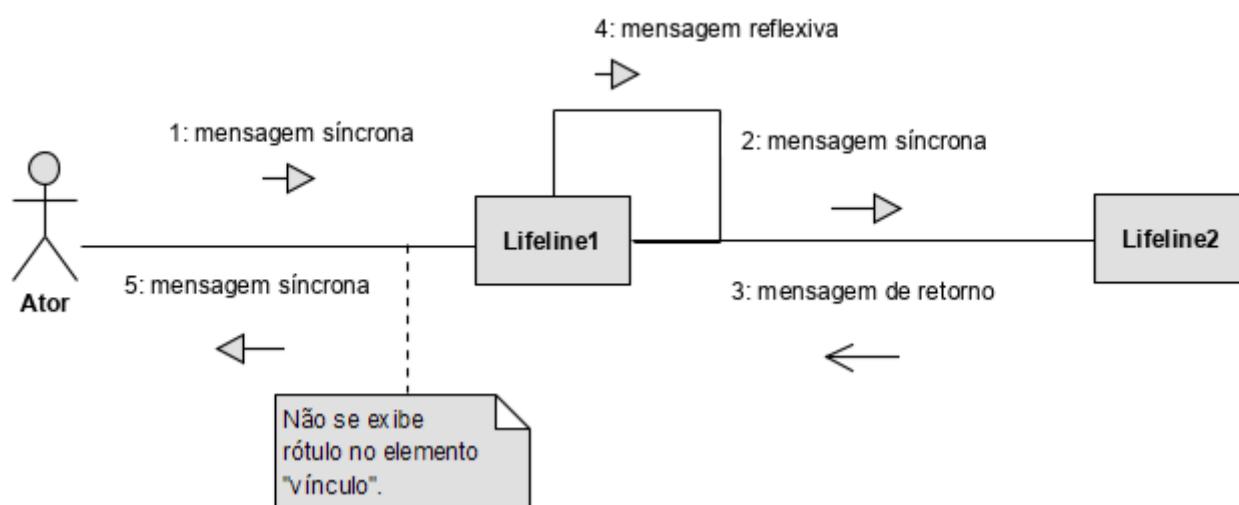
Notação	Elemento
	<p>Mensagem reflexiva ou automensagem: também pode ser denominada autochamada. É uma mensagem que indica que a <i>lifeline</i> dispara uma mensagem para si própria, de forma que ela é o remetente e o receptor da mensagem. A mensagem reflexiva é representada por uma linha que sai da <i>lifeline</i> e retorna para ela mesma.</p>
	<p>Nota ou comentário: é utilizado para descrever observações aos elementos que compõe o diagrama de sequência, contendo informações úteis para os desenvolvedores, porém não expressa força semântica específica aos elementos do diagrama.</p>

Fonte: elaborado pela autora.

A ênfase do diagrama de comunicação está em demonstrar exatamente a ligação entre os objetos representados pela *lifeline* que participam da realização de um caso de uso. Na especificação da modelagem de um software em uma ferramenta CASE compatível com a UML, é importante já ter elaborado o diagrama de casos de uso e o diagrama de classes para facilitar a elaboração do diagrama de comunicação a partir da reutilização dos elementos comuns. Lembrando também que, uma vez que se fez a opção em ter especificadas as interações, primeiramente com o diagrama de sequência, a maioria das ferramentas CASE gera automaticamente o diagrama de comunicação.

A Figura 3.18 ilustra a ligação entre os elementos básicos do diagrama de comunicação. Observe que a mensagem identificada com o número 3 refere-se a uma mensagem de retorno representada graficamente por uma seta aberta, no entanto não é tracejada como é no diagrama de sequência.

Figura 3.18 | Elementos do diagrama de comunicação

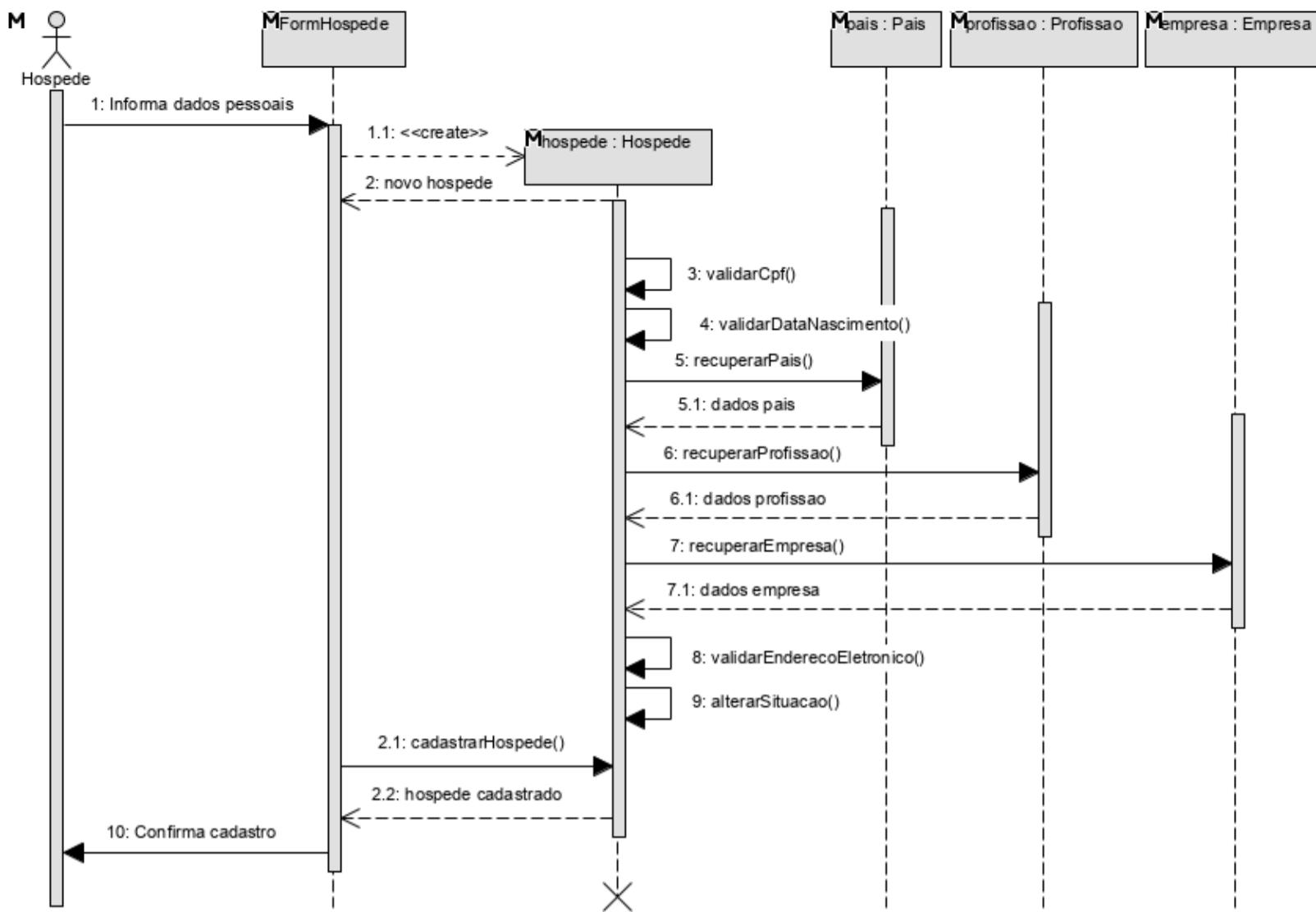


Fonte: elaborada pela autora.

A Figura 3.19 ilustra um diagrama de sequência correspondente ao cenário principal do caso de uso “Manter Hóspede”, que contempla o cadastro de um novo hóspede a partir da participação dos seguintes elementos, que realizam o caso de uso: ator “Hóspede”, ator primário responsável por fornecer os dados para que ocorra o cadastramento de um novo hóspede e que inicia a interação do caso de uso; *lifeline* “FormHospede”, representa a interface gráfica do caso de uso, sendo o elemento principal de interação entre o ator e os objetos das classes para que ocorra a troca de mensagens até concretizar o cadastramento de um novo hóspede; e os objetos das classes “pais:Pais”, “profissao:Profissao”; “empresa:Empresa”; e “hospede:Hospede”.

Agora confira a Figura 3.20, que apresenta o diagrama de comunicação correspondente ao diagrama de sequência da Figura 3.19 com os mesmos elementos que participam da realização do caso de uso. O diagrama de comunicação foi gerado automaticamente pela ferramenta CASE *Visual Paradigm Community Edition*. Na ferramenta, a partir do diagrama de sequência aberto na área de trabalho, seleciona-se a opção “*Synchronize to Communication Diagram*”, com o botão direito do mouse entre os elementos do diagrama de sequência, e, assim, o diagrama de comunicação é gerado com o nome idêntico ao do diagrama de sequência.

Figura 3.19 | Exemplo de diagrama de sequência



Fonte: elaborada pela autora.

Na Figura 3.20, observe que a numeração definida no rótulo das mensagens indica a ordem de envio das mensagens envolvidas na interação do caso de uso, sendo que, em algumas delas, a numeração está no formato simples, representado por números inteiros, por exemplo, 1, 2, 3 etc. e, em outras, está no formato composta, representada por valores em casas decimais, por exemplo, 2.1, 2.2, 5.1 etc., exatamente conforme consta no diagrama de sequência. Caso o diagrama de comunicação não seja elaborado automaticamente pela ferramenta CASE, a partir do diagrama de sequência, geralmente adota-se a numeração no formato simples entre todas as mensagens.

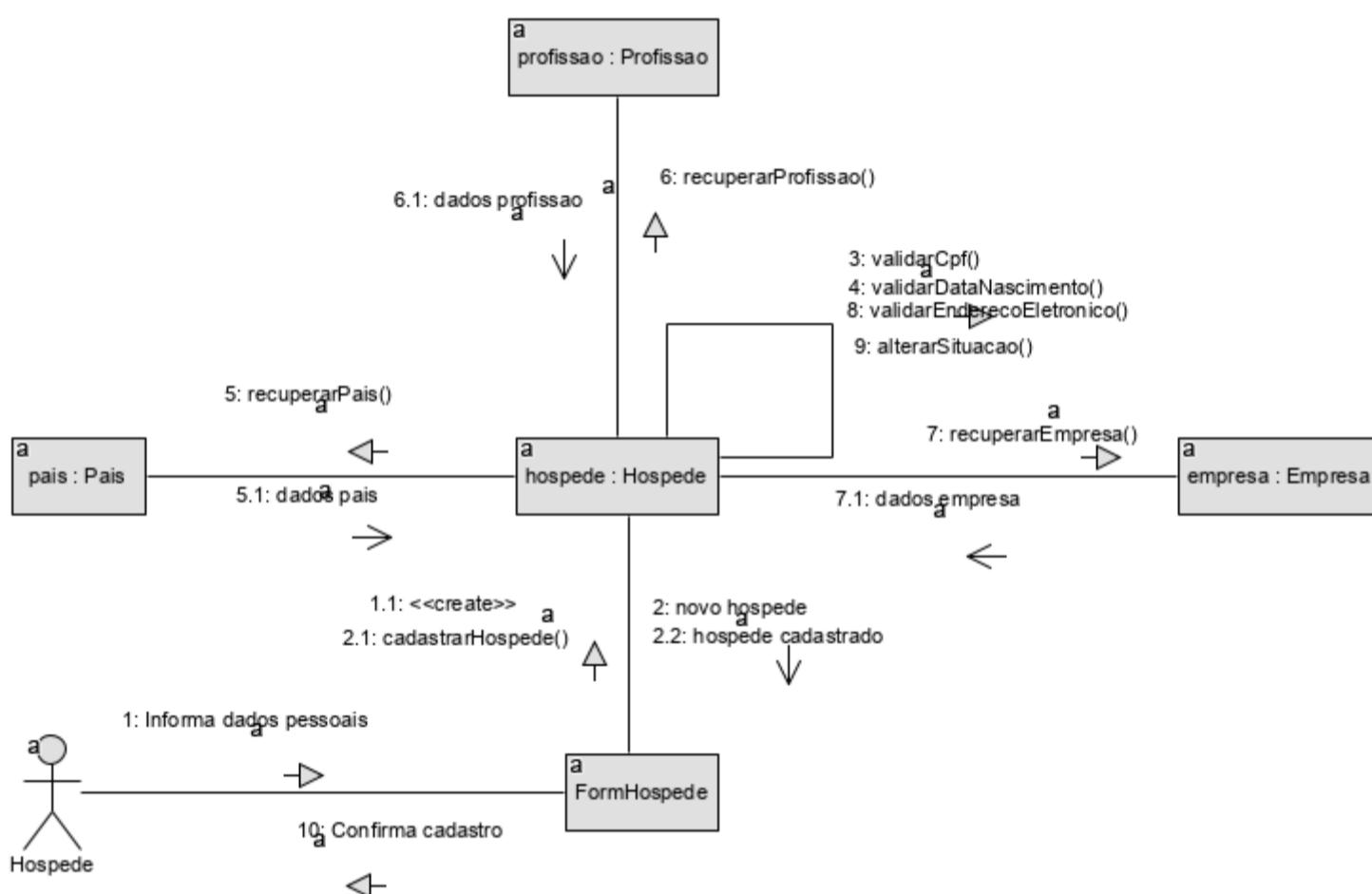
ASSIMILE

A ênfase do diagrama de comunicação está em demonstrar exatamente a ligação entre os objetos representados pela *lifeline*, que participam da realização de um caso de uso. No diagrama de comunicação, o relacionamento entre o ator e as *lifelines* é estabelecido pelo elemento denominado vínculo, sendo que, para indicar as mensagens, deve-se primeiramente estabelecer o vínculo entre as *lifelines*. Em um

único vínculo pode-se representar várias mensagens; cada uma delas é identificada por uma numeração que indica a ordem em que foram disparadas.

Observe novamente a troca de mensagens “2 e 2.2” entre a *lifeline* “FormHospede” e a *lifeline* do objeto “hospede:Hospede”; e a troca de mensagens “5.1, 6.1 e 7.1” entre a *lifeline* do objeto “hospede:Hospede” e as demais *lifelines* de objetos, que são todas indicativas de mensagens de retorno, representadas pela seta aberta.

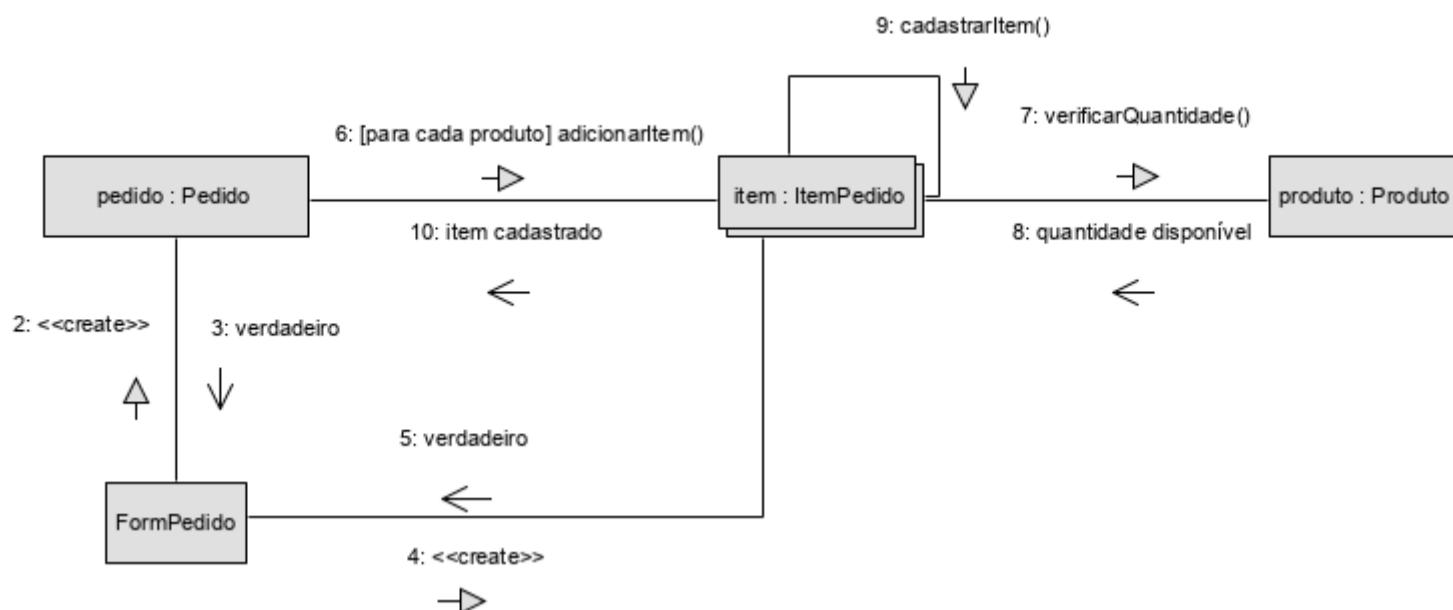
Figura 3.20 | Exemplo de diagrama de comunicação



Fonte: elaborada pela autora.

Dos elementos da notação gráfica do diagrama de comunicação, o **multiobjeto** é um elemento exclusivo dos diagramas de interação, sendo utilizado no diagrama de comunicação com mais frequência. A Figura 3.21 ilustra uma parte de um diagrama de comunicação que exemplifica o uso do elemento multiobjeto “item:item Pedido”, considerando que um pedido é composto por, no mínimo, um item obrigatoriamente.

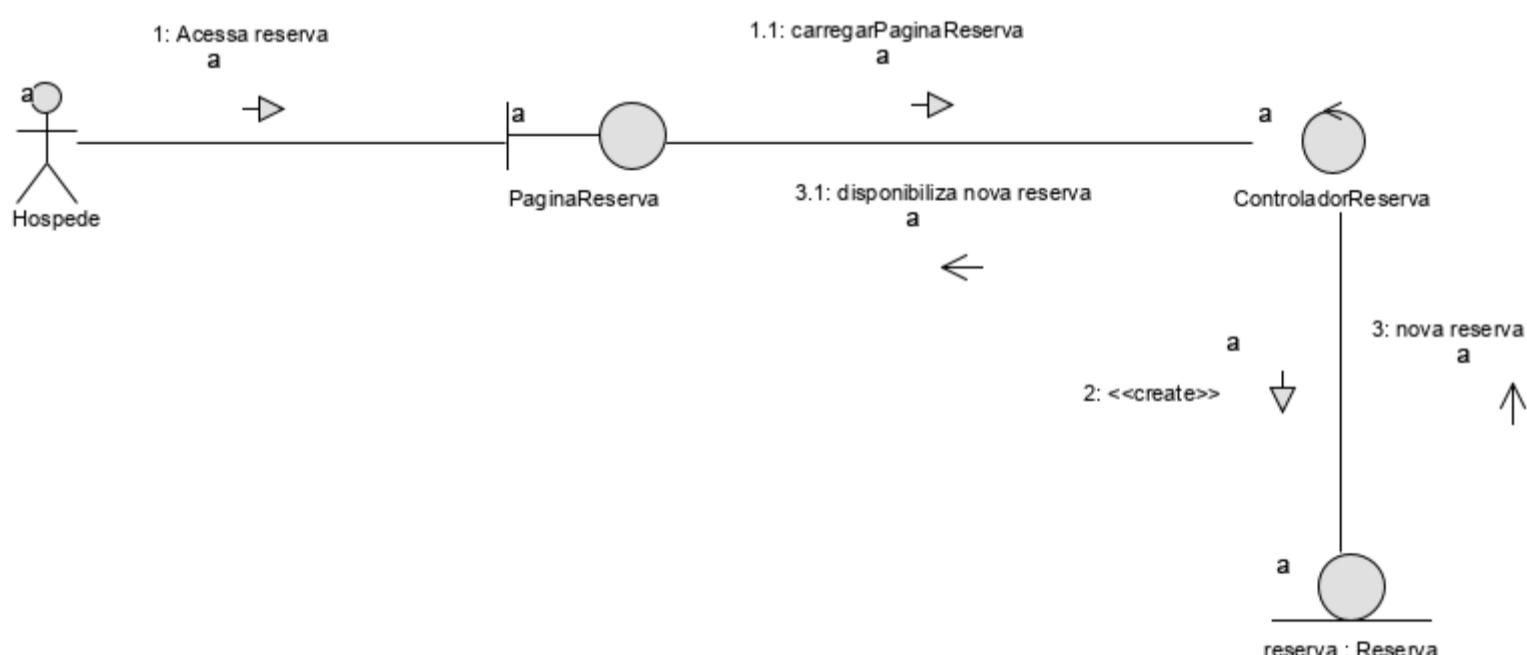
Figura 3.21 | Exemplo de diagrama de comunicação com multiobjeto



Fonte: elaborado pela autora.

Além disso, os mesmos estereótipos <<boundary>>, <<control>> e <<entity>> aplicados no diagrama de sequência podem ser igualmente utilizados nas *lifelines* do diagrama de comunicação, representando da mesma forma as mensagens trocadas entre os objetos vinculados à realização do caso de uso. A Figura 3.22 demonstra uma parte de um diagrama de comunicação com a indicação dos estereótipos.

Figura 3.22 | Exemplo de diagrama de comunicação com estereótipos



Fonte: elaborada pela autora.

No exemplo, o ator “Hóspede” inicia a realização do caso de uso “Realizar Reserva”, interagindo com a página correspondente à funcionalidade de realização de uma reserva. O elemento “PaginaReserva” indica uma *lifeline* do tipo fronteira (<<boundary>>); o

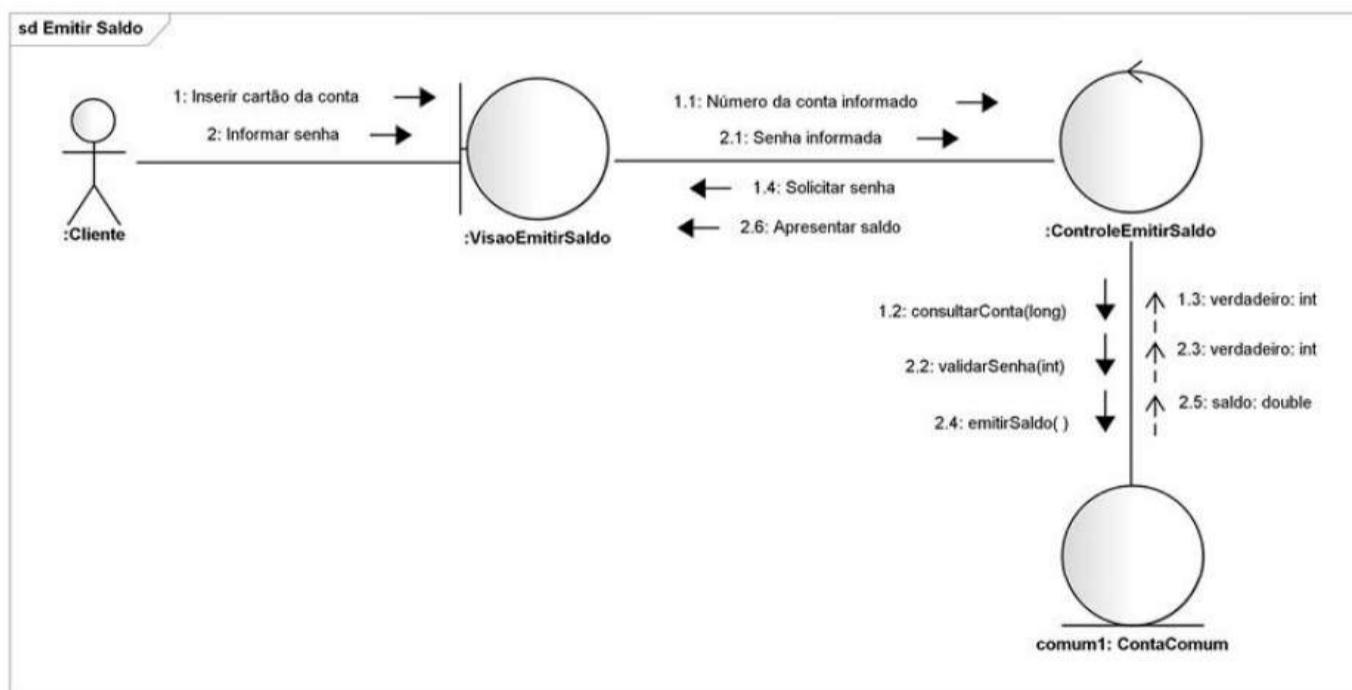
elemento “ControladorReserva” indica uma *lifeline* do tipo controladora (<<control>>) e o elemento “reserva:Reserva” indica um objeto *lifeline* do tipo entidade (<<entity>>).

É importante atentar-se para o fato de que, durante a modelagem das interações, a indicação de cada mensagem trocada entre os objetos envolvidos na realização de um processo, geralmente os casos de uso, implicará a existência de uma operação no objeto receptor, o qual deve ser consistentes com as operações listadas nas classes do diagrama de classes.

EXEMPLIFICANDO

A figura a seguir ilustra um exemplo do diagrama de comunicação com as *lifelines* representadas pelos estereótipos de classes <>boundary>>, <>control>> e <>entity>> correspondentes ao caso de uso “Emitir Saldo”, referente ao exemplo clássico do domínio de movimentação bancária em caixas eletrônicos denominado *Automatic Teller Machine* (ATM). No exemplo, o ator “Cliente” interage com a *lifeline* “VisaoEmitirSaldo” e esta com a *lifeline* que representa a classe controladora da interação “ControleEmitirSaldo”, que por sua vez se comunica com a *lifeline* da classe “ContaComum” a partir da troca de mensagens síncronas e de retorno. Observe que o diagrama foi modelado em uma ferramenta CASE, a qual indica as mensagens de retorno com a seta aberta, conforme o padrão da notação, e com a linha tracejada idêntica à representação do diagrama de sequência.

Figura 3.23 | Exemplo de diagrama de comunicação



Fonte: Guedes (2018, p. 35).

A seguir vamos conhecer o objetivo e a aplicabilidade de outro diagrama de interação, o diagrama de visão geral de interação.

DIAGRAMA DE VISÃO GERAL DE INTERAÇÃO

O diagrama de visão geral de interação é um novo diagrama da UML 2.0. É uma variação do diagrama de atividades, que integra os diagramas de interação, principalmente o diagrama de sequência, demonstrando um processo geral.

De acordo com Guedes (2018), o diagrama de visão geral de interação demonstra uma visão geral de um sistema ou processo, envolvendo vários subprocessos que interagem entre si a partir de um fluxo similar ao do diagrama de atividades, o qual utiliza quadros no lugar dos nós de ação. Segundo Bezerra (2014), o diagrama de visão geral de interação consiste em uma abordagem para modularizar a construção de diagramas de interação com o objetivo de dar uma visão geral dos diversos cenários de um caso de uso.

o

Ver anotações

Com o diagrama de visão geral de interação é possível ter uma visão de alto nível das interações de vários processos ou de um único processo correspondente à realização de um caso de uso.

Assim, quem orienta a melhor forma de adotar o diagrama não é a definição da UML e sim a metodologia de uma equipe e/ou empresa de desenvolvimento de sistemas. Por exemplo, considere o domínio de um sistema de hotelaria referente ao módulo reserva, o qual mantém as funcionalidades do cadastro de hóspede, realização de reserva e demais funcionalidades. Ao adotar o diagrama de visão geral de interação, em um único diagrama, seria possível unir a representação das interações do cadastro do hóspede, especificando as operações de inclusão, alteração ou consulta, e, na sequência, integrar a representação da interação correspondente à realização da reserva; ou representar dois diagramas separados. Essa decisão, depende da complexidade do domínio e funcionalidades do sistema.

| NOTAÇÃO GRÁFICA

A notação gráfica do diagrama de visão geral de interação consiste na representação de dois tipos de quadros, conforme ilustra a Figura 3.24, sendo:

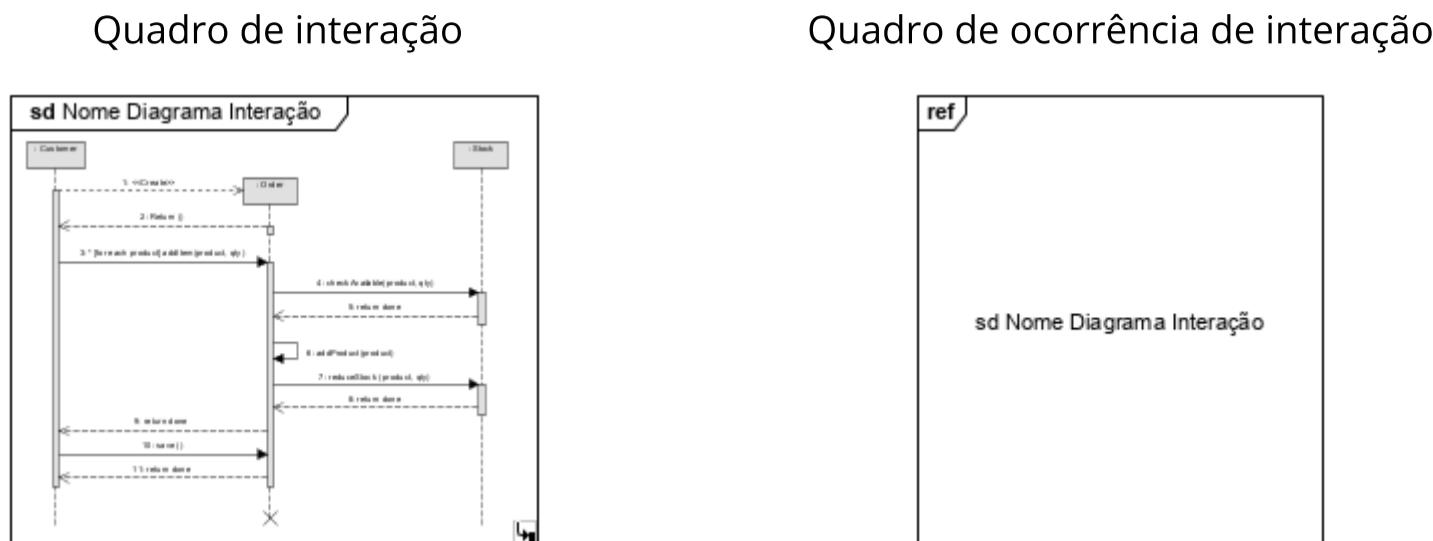
- **Quadros de interação:** os que contêm a representação completa dos diagramas de interação do tipo diagrama de sequência ou diagrama de comunicação. O quadro de interação é representado graficamente por um retângulo que contém uma subdivisão no canto superior esquerdo identificado por um rótulo com a expressão “*TipoDiagrama NomeDiagrama*”; no interior do quadro, é especificado o diagrama de interação de acordo com o tipo descrito no rótulo.
- **Quadros de ocorrência de interação:** fazem referência a um diagrama de interação especificado separadamente por um

diagrama de sequência, contudo não apresentam detalhamento, ou seja, pode-se utilizar ocorrências de interação para fatorar interações comuns a vários diagramas de visão geral de interação e reutilizar esse quadro diversas vezes. O quadro de ocorrência de interação também é representado graficamente por um retângulo, contendo uma subdivisão no canto superior esquerdo, identificado por um rótulo com a palavra “**ref**”; no interior do quadro é descrito apenas o nome do diagrama de interação reutilizado.

ASSIMILE

O diagrama de visão geral de interação é uma variação do diagrama de atividades, no entanto utiliza quadros no lugar dos nós de ação, demonstrando uma visão de alto nível das interações de um sistema ou processo a partir da integração de vários diagramas de interação, principalmente os do diagrama de sequência.

Figura 3.24 | Notação gráfica de quadros de interação e de quadros de ocorrência de interação



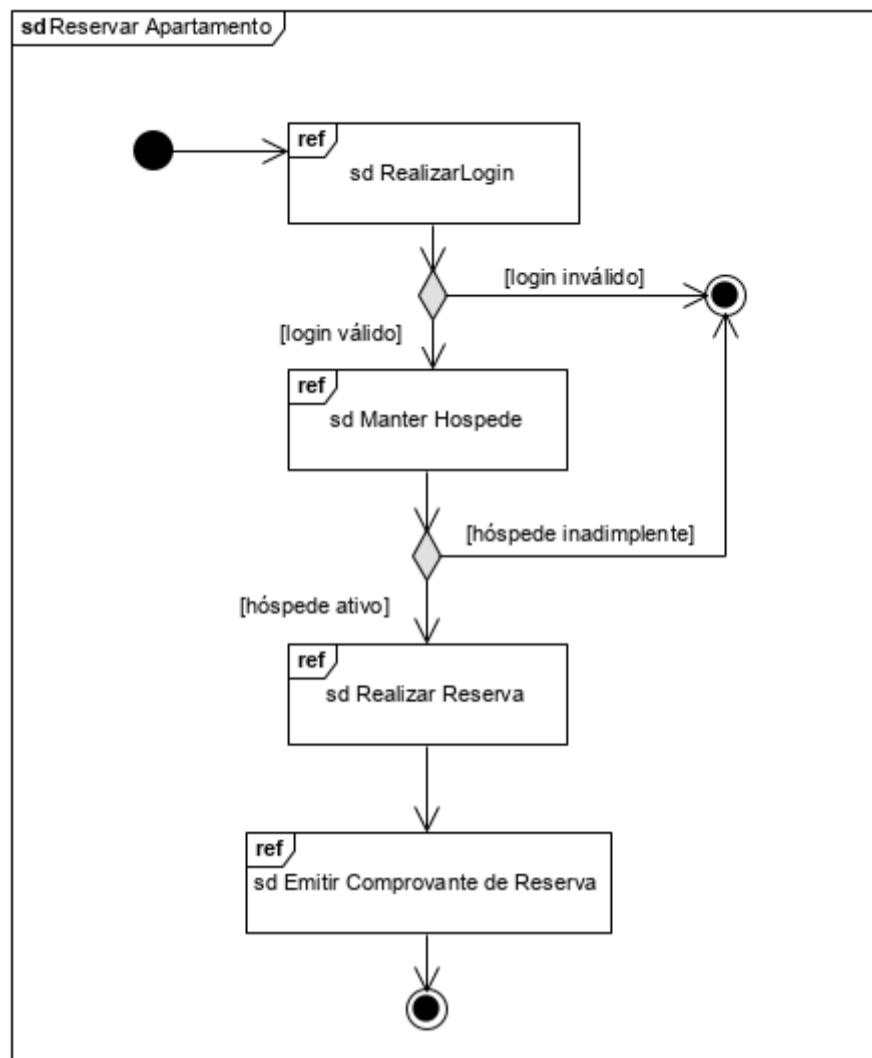
Fonte: elaborada pela autora.

Na elaboração do diagrama de visão geral de interação, além da representação dos quadros de interação e/ou quadros de ocorrência de interação, há a união dos quadros com os mesmos elementos do diagrama de atividades (nó inicial, nó final, nó de decisão, nó de bifurcação e nó de união), que você conheceu na Seção 2.3 deste livro.

A Figura 3.25 ilustra um diagrama de visão geral de interação integrando quadros de ocorrência de interação dos seguintes diagramas de sequência: *Realizar Login*, *Manter Hóspede*, *Realizar Reserva* e *Emitir Comprovante de Reserva*.

Na Figura 3.25 é possível observar o controle do fluxo de várias atividades para concretizar o processo geral de reservar um apartamento, o qual utilizou-se de quadros de ocorrência de interação no lugar dos nós de ação de um diagrama de atividades.

Figura 3.25 | Exemplo de diagrama de visão geral de interação



Fonte: elaborado pela autora.

E, para finalizar os diagramas de interação da UML, vamos conhecer o diagrama de tempo.

| DIAGRAMA DE TEMPO

O diagrama de tempo foi introduzido a partir da UML 2.0. Ele representa, de forma concisa e simples, a mudança pontual nos estados de um objeto, relevante para o contexto da execução de um processo que envolve várias atividades ou especificamente de um caso de uso em resposta aos eventos disparados durante uma interação. Pode-se indicar diferentes unidades de medida de tempo no mesmo diagrama ou apenas uma.

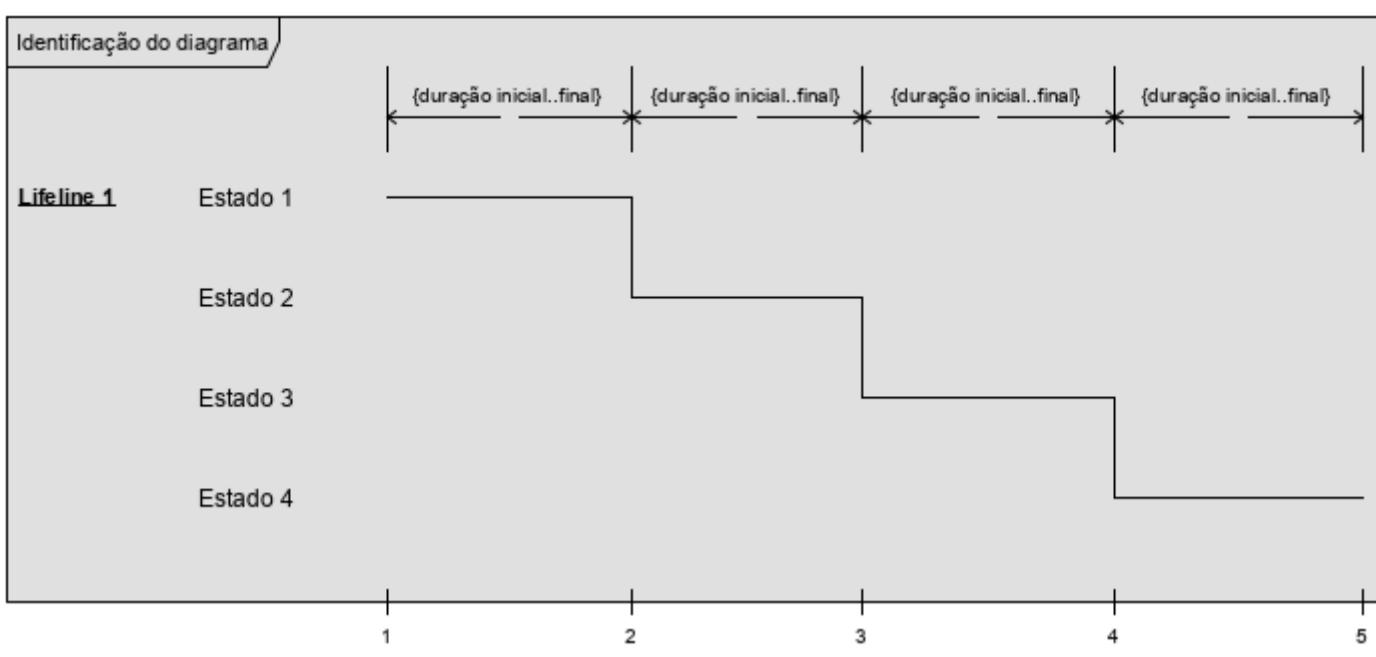
O diagrama de tempo é útil para documentar a modelagem principalmente de sistemas de tempo real, bem como de aplicações que utilizam recursos de multimídia e sincronização de eventos.

De acordo com Guedes (2018), o diagrama de tempo, ou de temporização, demonstra a mudança no estado de um objeto, uma condição de uma instância de uma classe ou o papel que ela assume em um período exato e importante de tempo, em resposta aos eventos externos.

O diagrama de tempo concentra-se nas condições que mudam dentro e entre linhas de vida ao longo de um eixo de tempo linear. O diagrama descreve o comportamento dos classificadores individuais e as interações dos classificadores, concentrando a atenção no tempo dos eventos, que causam mudanças nas condições modeladas das linhas de vida.

A notação gráfica do diagrama de tempo consiste em um único retângulo (quadro) com a indicação do nome em um rótulo, indicado no canto superior à esquerda do quadro, contendo os seguintes elementos, conforme ilustra a Figura 3.26:

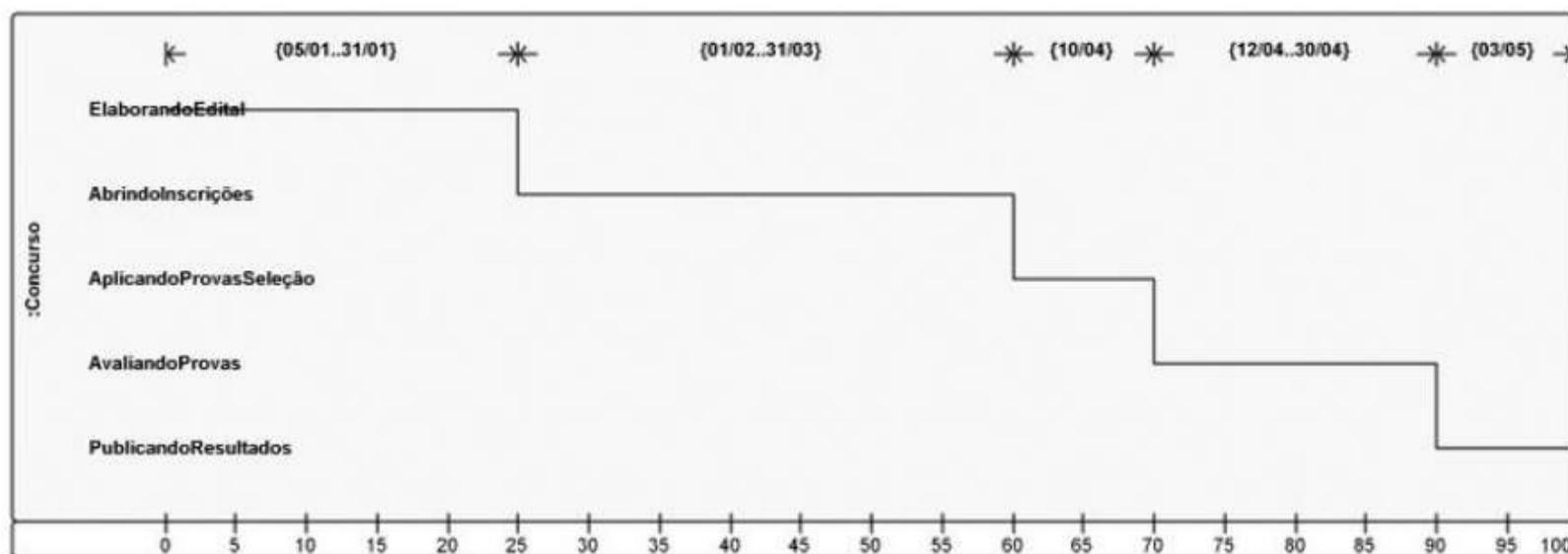
- **Lifeline:** linha de vida é um elemento nomeado que caracteriza um participante individual na interação, representando apenas uma entidade que interage. A linha de vida é aparece com o nome do classificador ou com a instância que ele representa. É colocada dentro do quadro do diagrama ou em uma "raia".
- **Estado ou condição:** são os estados do classificador ou atributo participante ou de algumas condições testáveis, como um valor discreto ou enumerável de um atributo. O nome dos estados ou condições são vinculados a cada *lifeline*, ilustrados de forma concisa na vertical ou no formato robusto que concentra os estados distribuídos na linha de vida.
- **Tempo da restrição:** o tempo da restrição refere-se a um intervalo de tempo, representado por uma unidade de tempo, indicada no eixo X do quadro, podendo ser segundos, minutos, horas, dias etc. O intervalo de tempo é uma expressão usada para determinar se a restrição é satisfeita. A restrição de tempo é mostrada como associação gráfica entre um intervalo de tempo e a construção que ela restringe. Normalmente, essa associação gráfica é uma pequena linha, por exemplo, entre uma especificação de ocorrência e um intervalo de tempo.
- **Duração da restrição:** refere-se a um intervalo de duração, que indica o tempo mínimo e máximo que dura a transição de um estado. A duração da restrição é mostrada como uma associação gráfica entre um intervalo de duração e as construções que ela restringe.



Fonte: elaborada pela autora.

A Figura 3.27 ilustra um exemplo de diagrama de tempo correspondente à classe de objetos “Concurso”, que apresenta as etapas que identificam os estados “*ElaborandoEdital*, *AbrindoInscrições*, *AplicandoProvaSeleção*, *AnalizandoProvas* e *PublicandoResultados*”, definindo o período das datas em função da unidade de tempo dia.

Figura 3.27 | Exemplo de diagrama de tempo



Fonte: Guedes (2018, p. 353).

Dependendo do domínio e da complexidade de um projeto de software, é importante fornecer múltiplas visões da modelagem do sistema sob diferentes aspectos de análise e detalhamento. Como vimos, a UML privilegia a descrição de um sistema em três perspectivas principais de visões, sendo que a perspectiva da visão estrutural enfatiza a visão estática do sistema; a funcional prioriza a modelagem das funcionalidades do sistema correspondente aos requisitos funcionais; e a perspectiva dinâmica representa a visão do comportamento dos objetos em tempo de execução.

Agora você já conhece todas as técnicas de modelagem da UML, sabe o objetivo e a aplicabilidade de cada modelo, sendo que a UML abrange as técnicas de modelagem classificadas em estruturais e comportamentais. Contudo, a UML não define quais de suas técnicas de modelagem devem ser adotadas para documentar exatamente uma fase ou atividade de um processo de desenvolvimento.

REFLITA

Considerando que você tem que definir uma metodologia de desenvolvimento de software orientado a objetos, integrando um modelo de processo de engenharia com a *Unified Modeling Language* (UML) para documentar os artefatos de software, quais técnicas de modelagem comportamental da

UML você acredita que são essenciais e recomendadas para especificar as interações entre os objetos que realizam os casos de uso?

A MODELAGEM

Como modelagem essencial da atividade de análise de um processo de desenvolvimento de software, recomenda-se iniciar a modelagem a partir da delimitação do escopo do projeto de software e, com isso, dimensionar as partes que integram o sistema e, se necessário, adotar o diagrama de pacotes para agrupar e organizar, em pacotes lógicos ou físicos, os componentes ou módulos que integram um sistema e suas dependências. Diante dessa definição, inicia-se a modelagem comportamental do sistema com o diagrama de casos de uso (*Use Cases*), representando um refinamento da especificação dos requisitos funcionais do sistema com o objetivo de definir os serviços, tarefas ou funcionalidades do software. Na sequência, enfatiza-se a modelagem estrutural das classes de objetos a partir da especificação do diagrama de classes, sendo que, à medida que o sistema é desenvolvido, o modelo de classes é incrementado com novos detalhes de notação, sendo refinado em vários níveis de abstração ou versões. Por fim, enfatiza-se ainda a modelagem comportamental dos estados dos objetos das classes, elaborando um diagrama de máquina de estados para cada classe que possui estados relevantes para o contexto do domínio do sistema; e geralmente se faz a opção em demonstrar a realização de cada caso de uso com o diagrama de atividades ou com os diagramas de interação.

Assim, a consistência entre os modelos que integram a documentação da modelagem de análise e de projeto de um sistema de software deve ser assegurada a partir do desenvolvimento incremental e iterativo dos artefatos de software. Na própria atividade de análise, os modelos podem ser refinados com mais detalhes, apresentando várias perspectivas de visões de um mesmo diagrama e, posteriormente, a modelagem da atividade de projeto é considerada uma extensão

refinada dos diagramas de análise com um olhar para a implementação de acordo com as tecnologias que serão adotadas para o desenvolvimento do sistema.

Nesta seção finalizamos o estudo das técnicas de modelagem comportamentais da UML, que visam demonstrar o comportamento e a interação entre os elementos do sistema, colaborando para a perspectiva da visão dinâmica do sistema. Os diagramas de interação representam um subgrupo dos diagramas comportamentais, que mostram a interação de como os objetos do sistema agem internamente para apoiarem a realização dos casos de uso definidos no diagrama de casos de uso.

REFERÊNCIAS

BEZERRA, E. **Princípios de análise e projeto de sistemas com UML**. 3. ed. Rio de Janeiro: Elsevier, 2014.

BOOCH, G.; JACOBSON, I.; RUMBAUGH, J. **UML**: guia do usuário. 2. ed. Rio de Janeiro: Campus, 2006.

GUEDES, G. T. A. **UML**: uma abordagem prática. 3. ed. São Paulo: Novatec, 2018.

SOMMERVILLE, I. **Engenharia de software**. 10. ed. São Paulo: Pearson Education do Brasil, 2018.

FOCO NO MERCADO DE TRABALHO

MODELAGEM DOS DEMAIS DIAGRAMAS DE INTERAÇÃO

Iolanda Cláudia Sanches Catarino

Ver anotações 0

ELABORAÇÃO DO DIAGRAMA DE COMUNICAÇÃO

O diagrama de comunicação pode ser gerado a partir do diagrama de sequência.



Fonte: Shutterstock.

Deseja ouvir este material?

Áudio disponível no material digital.

Agora confira o diagrama de comunicação que correspondente ao caso de uso “Realizar Reserva”!

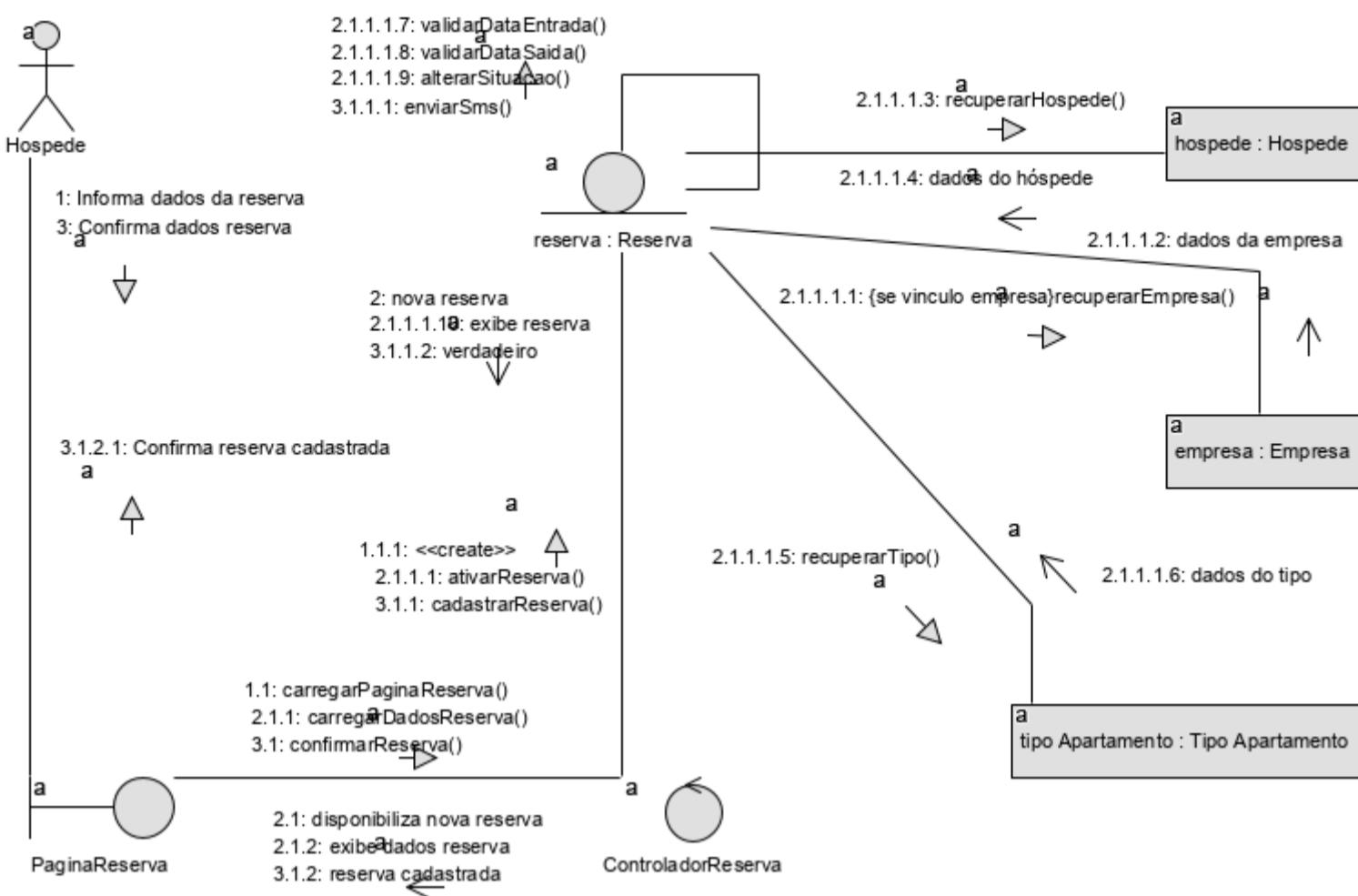
Na descrição da situação-problema, referente à funcionalidade para realização de reservas do Sistema de Hotelaria – Módulo Recepção, você observou, no recorte do diagrama de casos de uso, a representação do caso de uso “Realizar Reserva”, que interage com o ator primário “Hóspede” e está relacionada com o caso de uso estendido “Emitir Comprovante da Reserva”, com a condição de que o comprovante pode ser emitido se a reserva for realizada via Web ou por aplicativo; e está relacionada ainda com o caso de inclusão “Enviar SMS Confirmação da Reserva” obrigatoriamente, independentemente do modo como foi realizada a reserva.

Para elaborar o diagrama de comunicação, também foi necessário analisar as classes definidas no diagrama de classes, para, assim, identificar os objetos que participam da realização do caso de uso, sendo as classes “Reserva”, “TipoApartamento”, “Empresa” e “Hospede” as que estão associadas. Um objeto reserva, para ser criado, faz referência aos objetos das classes empresa, hóspede e tipo de apartamento, considerando a multiplicidade indica nas associações.

Na elaboração do diagrama de comunicação, você pode ter elaborado primeiramente o diagrama de sequência para depois gerá-lo automaticamente na ferramenta CASE, ou não. Para facilitar a construção do diagrama, é recomendado elaborar a descrição do cenário do caso de uso no formato de roteiro principal e alternativos ou, no mínimo, desenhar o protótipo da interface do caso de uso para assim ter a melhor compreensão de uma sequência lógica ideal para a execução da funcionalidade.

A Figura 3.28 apresenta o diagrama de comunicação referente ao cenário principal do caso de uso “Realizar Reserva”.

Figura 3.28 | Diagrama de sequência (cenário principal)



Fonte: elaborada pela autora.

O diagrama foi gerado automaticamente a partir do diagrama de sequência elaborado na ferramenta CASE *Visual Paradigm Community Edition*. Assim, a numeração que identifica as mensagens trocadas entre o ator e as *lifelines* são exatamente as geradas no diagrama de sequência. Contudo, se você elaborasse o diagrama de comunicação não vinculado ao diagrama de sequência, você poderia ter atribuído a numeração sequencial no formato simples.

NÃO PODE FALTAR

MODELAGEM INICIAL DA ATIVIDADE DE ANÁLISE

Iolanda Cláudia Sanches Catarino

Ver anotações 0

MODELAGEM DO ESTUDO DE CASO

Especificação dos casos de uso e das classes e, da documentação dos principais casos de uso.



Fonte: Shutterstock.

Deseja ouvir este material?

Áudio disponível no material digital.

CONVITE AO ESTUDO

Olá! Seja bem-vindo à última unidade do livro de Análise Orientada a Objetos.

Ao longo da história da engenharia de software, os modelos de processo de desenvolvimento de software evoluíram para atender satisfatoriamente aos diferentes domínios de sistemas computacionais, em consonância com as características e aplicabilidade das tecnologias de desenvolvimento de softwares.

As características do modelo de processo de software denominado de Processo Unificado (PU – *Unified Process*) foram apresentadas na primeira unidade deste livro, e nesta seção o tema será retomado, aplicado em um estudo de caso. PU surgiu para apoiar a Linguagem de Modelagem Unificada (UML – *Unified Modeling Language*), e nas demais unidades você compreendeu o objetivo e representação das principais técnicas de modelagem estrutural e comportamental da UML, que enfatizam as características de desenvolvimento dirigido a casos de uso, de forma iterativa e incremental, conduzindo, assim, de forma sistemática e evolutiva, a modelagem de sistemas orientados a objetos com a UML. Lembre-se: a UML não faz grande distinção entre a modelagem das atividades de análise e projeto de um modelo de processo de desenvolvimento de software.

Agora, nesta unidade, para consolidar o seu aprendizado acerca da análise orientada a objetos com a UML e reforçar a aplicação e integração das principais técnicas da UML na modelagem de um sistema de informação, demonstramos a modelagem das atividades de análise e projeto correspondente à fase de elaboração do PU, referente ao estudo de caso Locadora de Veículos. Considerou-se que na fase de concepção foram definidos a ideia inicial do negócio, o domínio do problema, a delimitação do escopo e planejamento do projeto e a identificação dos atores do sistema, estabelecendo a natureza dessa interação em uma perspectiva de alto nível, e foram elencados os principais casos de uso do sistema, compreendendo assim, o entendimento dos requisitos funcionais do sistema. Para especificação das técnicas de modelagem foi adotada a ferramenta *CASE Visual Paradigm Community Edition*.

Na primeira seção desta unidade, iniciamos a modelagem do estudo de caso, apresentando a descrição e regras de negócio do estudo de caso, a especificação dos casos de uso e das classes e, ainda, a documentação dos principais casos de uso, no formato de descrição numerada, a partir da descrição dos cenários principal e alternativos.

Na segunda seção, continuamos com a modelagem do estudo de caso a partir do Diagrama de Estrutura Composta das principais colaborações do sistema e, posteriormente enfatizamos a modelagem comportamental e de interação do sistema com a especificação dos principais Diagramas de Atividades, Diagramas de Máquina de Estados e Diagramas de Sequência.

Na terceira seção da unidade, apresentamos o refinamento dos aspectos estáticos e estruturais do Modelo de Classes, complementando as classes com os tipos de dados dos atributos com as demais operações que foram identificadas na especificação dos diagramas comportamentais e de interação, além da revisão dos relacionamentos das classes, apresentando, assim, uma nova versão do Modelo de Classes. Na sequência, abordamos a persistência dos objetos,

demostrando o mapeamento das classes em tabelas de banco de dados relacional, conforme os tipos de relacionamentos estabelecidos entre as classes de objetos. Por fim, apresentamos o refinamento dos principais aspectos comportamentais da modelagem, que reflete na consistência e interação dos diagramas da UML. Assim, tratamos a modelagem da atividade de Projeto como um refinamento da modelagem de Análise, ou seja, uma extensão detalhada do Modelo de Casos de Uso e do Modelo de Classes.

Ver anotações

PRATICAR PARA APRENDER

Caro aluno, seja bem-vindo à seção sobre a modelagem inicial da atividade de Análise de um estudo de caso!

Nesta seção vamos adotar a UML para modelagem da atividade de Análise e Projeto do estudo de caso Locadora de Veículos, e diante do conjunto de diagramas classificados em estruturais e comportamentais da UML que enfatizam a modelagem da perspectiva estática e dinâmica do sistema, iniciamos a documentação do sistema com a especificação do Modelo de Casos de Uso e do Modelo de Classes.

Você se recorda do motivo pelo qual chamamos de Modelo de Casos de Uso e não Diagrama de Casos de Uso? No contexto de modelagem de um sistema, referente a uma atividade ou fase de um processo de desenvolvimento de software, o termo modelo refere-se exatamente a um conjunto de diagramas, independentemente da técnica de modelagem, ou seja, casos de uso ou classes, por exemplo. Isso quer dizer que, dependendo do tamanho do sistema, não é possível representar todos os casos de uso ou classes em um único diagrama. Assim, o recomendado é categorizar os casos de uso ou classes, agrupando-os por assunto e elaborar mais de um Diagrama de Casos de Uso ou Diagrama de Classes. Dessa forma, por exemplo, para a modelagem dos casos de uso de um sistema ou módulo de um sistema é necessário ter mais de dois Diagramas de Casos de Uso, ou seja, os vários Diagramas de Casos de Uso compõem o Modelo de Casos de Uso.

De acordo com Booch, Rumbaugh e Jacobson (2006), um modelo é uma simplificação da realidade e pode ser estrutural, com ênfase à organização do sistema, ou comportamental, com ênfase à dinâmica do sistema, consistindo nos objetivos de ajudar a visualizar o sistema como ele é ou como desejamos que seja, proporcionar um guia para a construção do sistema, permitir especificar a estrutura ou o comportamento de um sistema, além de formalizar as decisões tomadas para o sistema. Assim, não confunda o conceito de modelo, que remete à modelagem de um sistema, com o conceito de modelo de processo.

Segundo Pressman e Maxim (2016, p. 40) “um modelo de processo fornece um guia específico para o trabalho de engenharia de software. Ele define o fluxo de todas as atividades, ações e tarefas, o grau de iteração, os artefatos e a organização do trabalho a ser feito”.

Diante da quantidade de casos de usos que foram identificados para o sistema da locadora de veículos, decidiu-se representar os casos de uso em dois diagramas, um concentrando os casos de uso correspondentes aos cadastros e movimentações de negócio, e outro diagrama mantendo os casos de uso referentes às consultas e relatórios.

Avançando a modelagem do sistema da locadora de veículos a partir do entendimento da funcionalidade de cada caso de uso, as classes de objetos do sistema foram abstraídas e elaboramos a primeira versão do Diagrama de Classes, priorizando a estrutura dos atributos e a listagem das operações básicas dos objetos de cada classe.

É importante reforçar que a especificação do Modelo de Casos de Uso e do Modelo de Classes contempla as regras de negócio definidas na descrição do estudo de caso e demais definições que foram consideradas relevantes para complementar as funcionalidades, seguindo o ponto de vista de um analista de sistemas. Assim, fique à vontade para enriquecer a solução proposta com demais funcionalidades ou adaptações que considere pertinentes ao domínio do sistema; contudo, não esqueça de respeitar as boas práticas da engenharia de software.

Para reforçar a sua aprendizagem, considere que você faz parte da equipe de desenvolvedores da empresa que está responsável pelo desenvolvimento do sistema da locadora de veículos, e você é o analista de sistema responsável pela modelagem do módulo “Pagamento” que está integrado com o módulo “Locação de Veículos”.

Considere a seguinte complementação da descrição do estudo de caso, referente ao módulo “Locação de Veículos”.

O pagamento de um aluguel é efetuado no ato da devolução do carro. A forma de pagamento de um aluguel, pode ser à vista ou a prazo, dependendo do valor total do aluguel. Para pagamento à vista em dinheiro, o valor total da devolução deve ser lançado automaticamente no caixa da empresa, assim que uma devolução é concluída. Para pagamento a prazo em cartão de crédito, deve ser gerado um controle de contas a receber (crédito a receber), correspondente ao número de parcelas, sendo necessário posteriormente tratar a baixa de uma conta a receber. Mesmo para pagamento com cartão de crédito do tipo débito é lançado como contas a receber, devido o prazo mínimo de 24 horas para ser creditado na conta da locadora

Considerando o complemento da descrição do estudo de caso e o Diagrama de Classes apresentado nesta seção para o módulo de “Locação de Veículo”, faça: elabore o Diagrama de Classes, referente ao módulo “Pagamento”, representando a integração com as classes do módulo “Locação de Veículo”.

Bom trabalho e ótimo projeto!

CONCEITO-CHAVE

A importância em documentar todo o processo de desenvolvimento de software, evidenciando principalmente as etapas iniciais do processo, foi destacada a partir da “Crise do Software”, no final da década de 1960. Independentemente da natureza de um projeto, a modelagem é essencial para delimitarmos o problema em estudo e desenvolvimento, restringindo o foco a um único aspecto por vez.

A modelagem não faz parte apenas da indústria da construção civil. Para as diferentes áreas da engenharia, construímos modelos para demonstrar a definição e detalhes do projeto, auxiliando os usuários envolvidos a visualizarem qual será o produto final. Assim, na engenharia de sistemas de software a modelagem não se restringe apenas a grandes sistemas, no entanto, quanto maior e mais complexo for o sistema, maior será a importância da modelagem para diminuir a probabilidade de ocorrência de erros ou da construção dos requisitos errados.

Para compreender a arquitetura de um sistema de software, é necessário integrar várias visões de modelos que se inter-relacionam ou se complementam. Em conjunto, essas visões representam a base do projeto do software. Segundo Booch, Rumbaugh e Jacobson (2006), dependendo da natureza do sistema, alguns modelos podem ser mais importantes e adequados do que outros e podem ser especificados em diferentes níveis de precisão. Por exemplo, em sistemas que utilizam muitos dados predominarão modelos voltados para a visão estática do projeto. Em sistemas centrados no uso de *Graphical User Interface*

(GUI), são importantes as visões estáticas e dinâmicas dos casos de uso, e em sistemas de execução crítica em tempo real, a visão dinâmica do processo tende a ser mais importante.

A Unidade 1 contextualizou os fundamentos e princípios da *Unified Modeling Language* (UML) e você estudou que a UML privilegia a descrição da modelagem de sistemas de software em três perspectivas principais de visões, que são: a **estrutural**, que enfatiza a visão estática do sistema, ou seja, os dados; a **funcional**, que prioriza as funcionalidades do sistema, enfatizando os requisitos funcionais; e a **temporal**, que prioriza a especificação dos eventos, representando o comportamento dos objetos em tempo de execução.

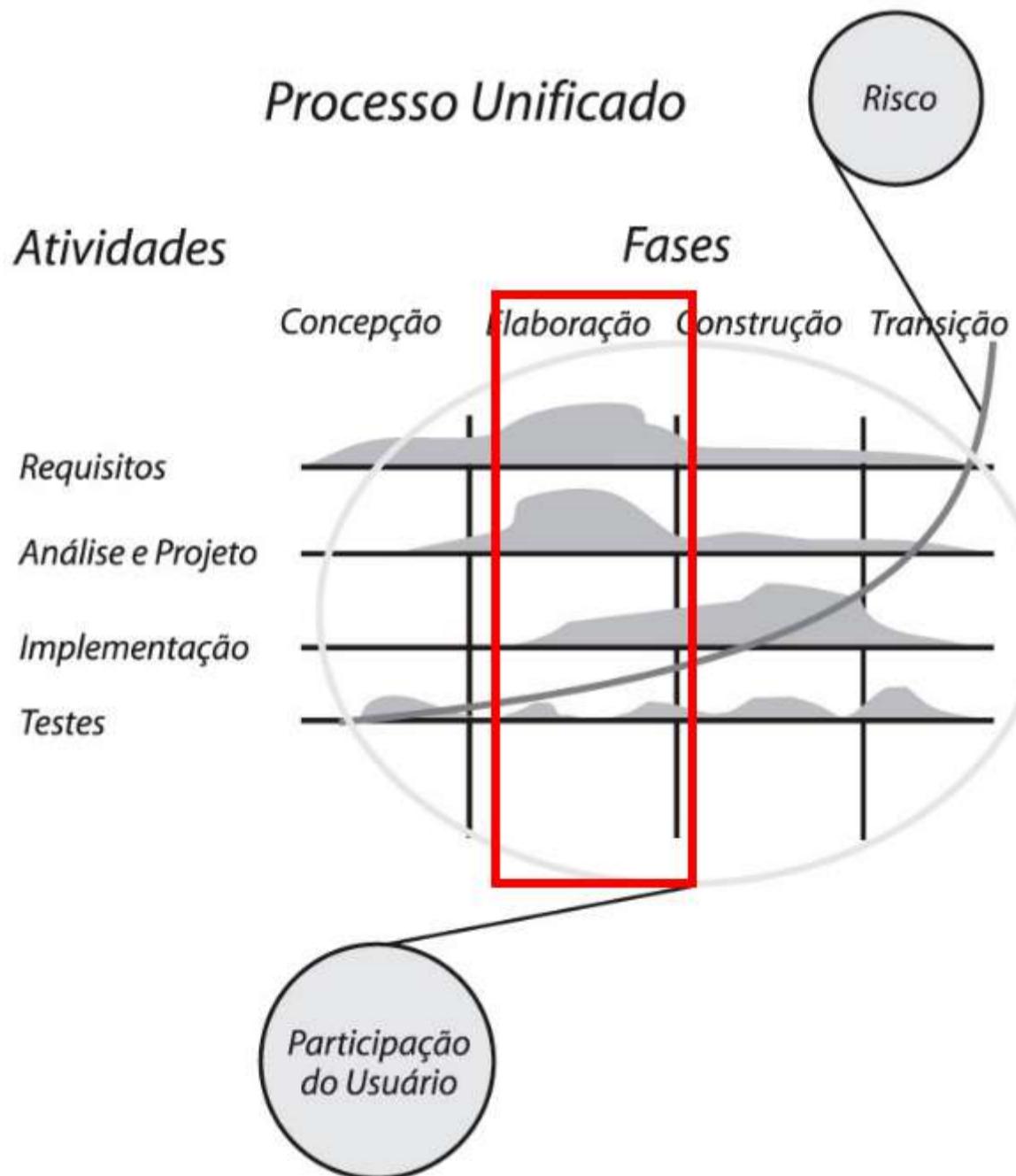
■ MODELAGEM DE SISTEMA COM PROCESSO UNIFICADO

Esta unidade consiste na demonstração da modelagem de um estudo de caso para o domínio de locadora de veículos, a partir da especificação das principais técnicas de modelagem da UML, para documentar as atividades de requisitos e análise e projeto, correspondentes à fase de elaboração do processo unificado.

Entre os vários modelos de processo da engenharia de software, o modelo de processo escolhido foi o Processo Unificado, considerando a sua relevante característica de manter um ciclo de vida incremental e iterativo, ou seja, o Processo Unificado apoia o desenvolvimento incremental, a partir de modelos que podem evoluir com a inclusão de novos detalhes durante a execução das atividades (*Workflow*) e, a cada ciclo do Processo Unificado, uma nova versão do produto é entregue aos usuários, conhecida como uma iteração.

A Figura 4.1 representa o ciclo de vida do Processo Unificado, indicando no eixo X a dimensão temporal distribuída nas fases de Concepção, Elaboração, Construção e Transição. Cada fase integra um conjunto de atividades – Requisitos, Análise e Projeto, Implementação e Testes, indicadas no eixo Y da figura, que são executadas de forma incremental e evolutiva, representando a entrega dos artefatos de software. Observe na figura a fase de Elaboração destacada, na qual prevalecem as atividades de Requisitos e Análise e Projeto. Assim, a documentação a seguir refere-se à especificação das principais técnicas de modelagem estruturais e comportamentais da UML, que foram adotadas na modelagem do estudo de caso proposto.

Figura 4.1 | Processo Unificado



Fonte: Medeiros (2004, p. 16).

| FASE DE CONCEPÇÃO

Iniciando a modelagem do sistema denominado “Locação de Veículos”, na primeira fase do Processo Unificado – Concepção, considerou-se:

1. O planejamento do projeto em consonância com a metodologia definida para o desenvolvimento do sistema.
2. A definição da ideia inicial do negócio, a partir da compreensão do domínio do problema ou domínio do sistema, ou seja, a compreensão do contexto para o qual foi provida a solução.
3. A delimitação do escopo do sistema, no qual identificam-se os processos de negócio aplicados aos requisitos do sistema.
4. O entendimento do contexto do sistema a partir da identificação dos atores do sistema e a natureza da interação entre os atores e o sistema, em uma perspectiva de alto nível.
5. A definição dos principais casos de uso do sistema, consistindo no entendimento dos requisitos funcionais do sistema.

ASSIMILE

Um processo de negócio é um conjunto de atividades ou tarefas estruturadas relacionadas que produzem um serviço ou produto específico para seus clientes. No processo de negócio, os insumos (materiais, conhecimento e outros) são transformados em resultados (produtos e serviços). Segundo Kirchoff (2015), um processo de negócio é o conjunto de atividades ou tarefas que são estruturadas e giram em torno da produção de um resultado de valor para o cliente, por meio da entrega de um serviço ou produto. Ele mostra o que deve ser realizado, como deve ser realizado e quem é o responsável.

| FASE DE ELABORAÇÃO

É na fase de Elaboração que se define como o sistema será construído, a partir da especificação dos requisitos funcionais do sistema, estabelecendo a arquitetura e mecanismos para o desenvolvimento do sistema. Nessa fase prevalece a execução das atividades de Requisitos e Análise e Projeto, conforme foi destacado na Figura 4.1.

No item a seguir, apresentamos a descrição do estudo de caso e uma modelagem mínima da atividade de Análise de Requisitos, utilizando algumas técnicas de modelagem da UML.

■ DESCRIÇÃO DO ESTUDO DE CASO – LOCAÇÃO DE VEÍCULOS

Para complementar o seu aprendizado, utilizaremos o estudo de caso Locação de Veículos para orientar na exemplificação da modelagem dos principais diagramas da UML.

LOCAÇÃO DE VEÍCULOS

Deseja-se desenvolver um novo sistema para uma locadora de veículos que expandiu e está atuando em várias cidades do país. A locadora de veículos aluga carros aos clientes previamente cadastrados. Um cliente será descrito por: nome, data de nascimento, sexo, nacionalidade, CPF, RG, endereço completo, telefone residencial, celular, endereço eletrônico, dados da CNH (número, registro, data validade, UF), profissão e situação (ativo, preferencial, inadimplente ou encerrado). Sendo um cliente estrangeiro, além dos dados já descritos, deve-se manter também um número de documento estrangeiro e o número do passaporte. Um cliente vinculado a uma empresa conveniada (empresa com cadastro aprovado e com desconto concedido aos seus colaboradores) da locadora deve indicar o nome da empresa no seu cadastro pessoal.

Os carros devem ser cadastrados, mantendo: placa, ano, marca, modelo, características, quilometragem, valor da diária, situação (disponível, alugado, manutenção interna, manutenção externa ou inativo) e observações. Os carros devem ser classificados por categoria ou grupo, conforme o seu modelo e características, por exemplo: econômico manual hatch, econômico manual sedan, econômico automático hatch, intermediário manual sedan, executivo, SUV etc.). O valor da diária e demais valores (cobertura do veículo, cobertura de terceiro, taxa de aluguel e quilometragem excedente) variam de acordo com a categoria do carro.

Um cliente poderá realizar o seu próprio cadastro pela web ou aplicativo a partir da criação de uma conta de usuário com login (endereço eletrônico ou CPF) e senha. Um cliente ativo ou preferencial poderá alugar vários carros em datas e horários distintos, com reserva prévia.

Um cliente previamente cadastrado poderá realizar reservas por telefone, web ou aplicativo. Uma reserva deve manter os dados do cliente, categoria do carro a ser reservada, opcionais do carro (bebê conforto, cadeira de bebê e/ou assento de elevação), período da reserva, loja e hora de retirada e de devolução do carro. Cada reserva corresponde a um carro. Toda reserva realizada por um cliente deverá enviar um comprovante da reserva para o cliente via e-mail e/ou SMS.

Um cliente poderá alugar um carro e retirá-lo em uma loja e posteriormente devolvê-lo em outra loja. Para cada aluguel deverá ser registrado um código do aluguel, dados do cliente, dados da categoria do carro alugado (categoria, valores, opcionais, quilometragem atual do carro), a data e hora do aluguel e a data e hora prevista de devolução. Cada aluguel corresponde a um carro. Após a efetivação do aluguel do veículo, deverá ser emitido o contrato de aluguel e o termo de vistoria do carro, sendo que uma cópia será entregue ao cliente e uma cópia o cliente assinará e ficará de posse da locadora.

Na devolução do carro, o cliente informará a forma de pagamento, será conferida a quilometragem percorrida durante o período do aluguel, registrará a quilometragem atual do carro, além da data e hora de devolução, será calculado o valor total do aluguel e a nota fiscal de serviço será emitida e/ou enviada para o e-mail do cliente. Com a confirmação da devolução do carro, a quilometragem e situação do carro serão atualizadas, sendo a situação alterada para “manutenção interna”. Posteriormente, realiza-se a conferência e lavagem do carro e sua disponibilização para um novo aluguel.

A forma de pagamento poderá ser à vista ou a prazo, dependendo do valor total do aluguel. A transação referente ao pagamento de um aluguel será realizada pelo sistema de pagamento que será integrado ao sistema de locação.

O sistema deve disponibilizar consultas e relatórios aos funcionários e gerentes, referentes aos clientes, carros, reservas, aluguéis e devoluções.

0

Na descrição do estudo de caso é possível identificar vários requisitos funcionais que estão explícitos na descrição, ou seja, as funcionalidades que o sistema deve prover para os diferentes atores que interagem com o sistema. Alguns requisitos funcionais já descrevem as suas regras de negócio (políticas, condições ou restrições que devem ser consideradas na execução dos processos), como para o requisito funcional “Cadastro de Cliente” estão evidentes algumas regras, sendo “a locadora de veículos aluga carros aos clientes previamente cadastrados”; “um cliente ativo ou preferencial poderá alugar vários carros em datas e horários distintos, com reserva prévia ou não”.

Ver anotações

■ ATIVIDADE REQUISITOS

No modelo de processo de engenharia de software – Processo Unificado, a atividade de Requisitos é a primeira atividade do ciclo de cada fase do processo. Abstrair, entender e definir os requisitos do domínio do problema é uma das tarefas mais difíceis da engenharia de software, pois é a etapa que fundamenta e sustenta todo o processo de desenvolvimento do software.

Segundo Pressman e Maxim (2016, p. 32)

“

[...] a engenharia de requisitos é uma ação de engenharia de software importante que se inicia durante a atividade de comunicação e continua na de modelagem. Ela deve ser adaptada às necessidades do processo, do projeto, do produto e das pessoas que estão realizando o trabalho.

Para Sommerville (2018), a engenharia de requisitos preocupa-se com o que deve ser feito, ou seja, a compreensão do problema, e não em como fazer, considerando o domínio do sistema. A Engenharia de Requisitos é o processo de descobrir, analisar, documentar e verificar os serviços e restrições.

Uma primeira classificação dos requisitos de um sistema é tratada como requisitos de usuário e *requisitos de sistema*, conforme exemplifica a Figura 4.2 (SOMMERVILLE, 2018):

- **Requisitos de usuário:** são declarações em uma linguagem natural, com diagramas ou não, de quais serviços o sistema deverá fornecer a seus usuários e as restrições com as quais este deve operar.
- **Requisitos de sistema:** são descrições mais detalhadas das funções, serviços e restrições operacionais do sistema de software. O documento de requisito de sistema, denominado de especificação funcional, deve definir exatamente o que a funcionalidade deve implementar.

Figura 4.2 | Exemplo de Requisito de Usuário e Requisito de Sistema

Requisito de Usuário	Requisito de Sistema
<pre> graph LR Cliente((Cliente)) --- ManterCliente([Manter Cliente]) </pre>	<ol style="list-style-type: none"> 1. Cliente solicita seu cadastro. 2. Sistema exibe o cadastro de clientes. 3. Cliente informa seus dados pessoais. 4. Sistema verifica que não existe cliente cadastrado. 5. Cliente informa demais dados pessoais. 6. Cliente confirma o seu cadastro. 7. Sistema registra o cliente.

Fonte: elaborada pela autora.

Por sua vez, os *requisitos de sistema* são classificados em dois tipos:

- **Requisitos Funcionais (RF):** são declarações de serviços que o sistema deve fornecer, de como o sistema deve reagir a entradas

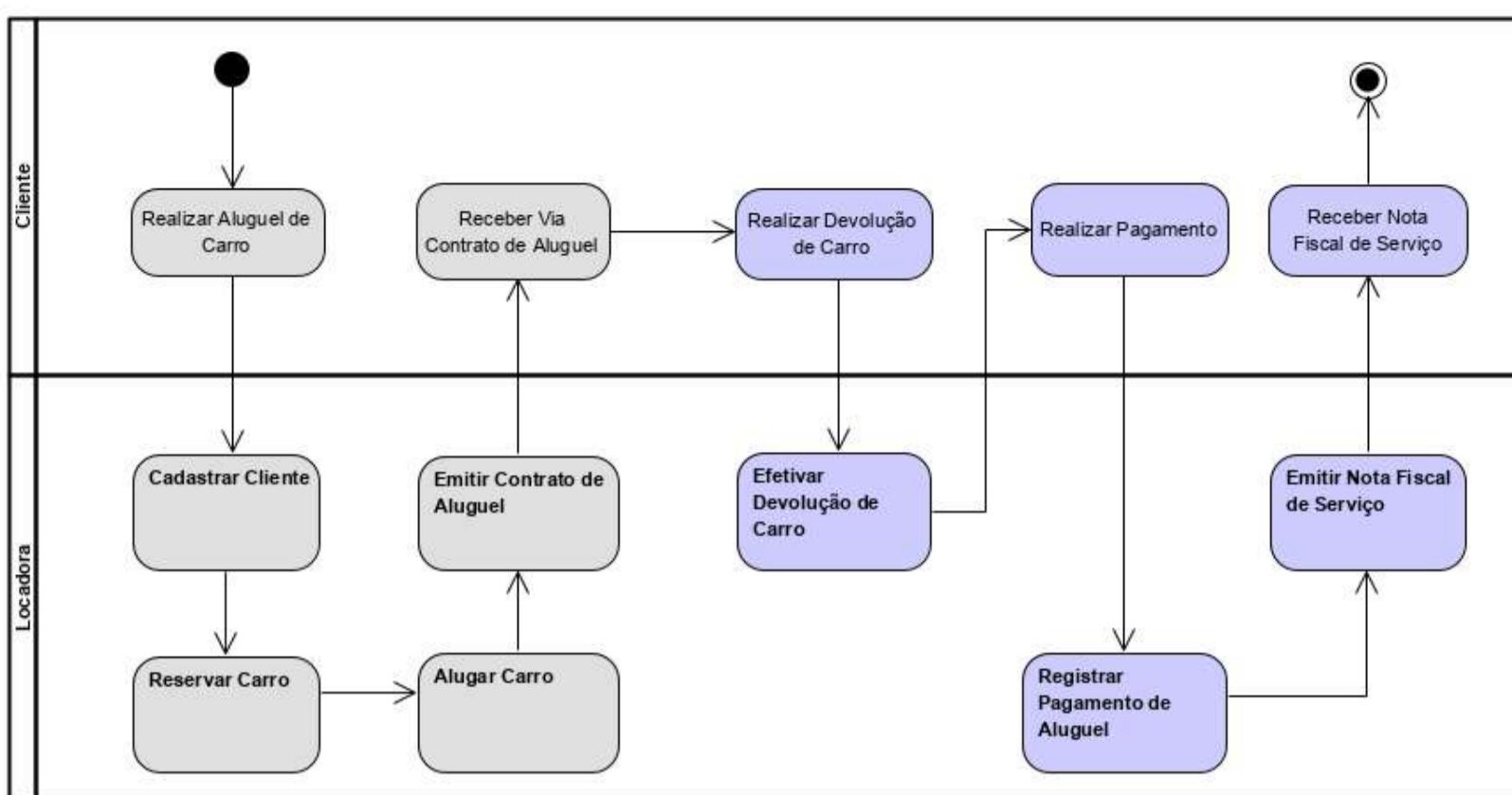
específicas e de como o sistema deve se comportar em determinadas situações, ou seja, representa uma funcionalidade que o sistema deve fornecer para atender a uma necessidade do usuário.

- **Requisitos Não Funcionais (RNF):** expressam restrições aos serviços ou funções, ou qualidades específicas às quais o software deve atender. As restrições referem-se a tempo, ao processo de desenvolvimento e muitas vezes são impostas pelas normas organizacionais, surgindo por meio das necessidades dos usuários, devido a restrições de orçamentos, políticas organizacionais e outros motivos. Um requisito não funcional pode estar vinculado a um requisito funcional ou pode-se aplicar ao sistema com o um todo.

Das técnicas de modelagem da UML, pode-se adotar o **Diagrama de Casos de Uso** para modelar os requisitos funcionais, que posteriormente, guiará o processo de desenvolvimento; o **Diagrama de Atividades** para representar o comportamento de cada requisito funcional do sistema, subsistemas ou de um ou mais processos de negócio do domínio do sistema; e pode-se adotar o **Diagrama de Sequência** para especificar o cenário de cada funcionalidade identificada como requisito funcional.

Para iniciar a modelagem dos requisitos do sistema “Locação de Veículos”, adotamos o Diagrama de Atividades em compartimentos, no formato de *swimlanes* (raias de natação), para facilitar a compreensão e entendimento do domínio do sistema, representando o fluxo de atividades dos principais processos de negócio que demonstram o comportamento do sistema. A Figura 4.3 ilustra o Diagrama de Atividades, demonstrando os atores “Locadora” e “Cliente” que realizam as atividades de locação e devolução de um carro de uma forma geral, sem detalhar as ações específicas que envolvem cada atividade e, ainda, considerando que o aluguel de um carro é realizado a partir de uma reserva.

Figura 4.3 | Diagrama de Atividades – Locação e Devolução de Carro



Fonte: elaborada pela autora.

A partir da descrição do estudo de caso e da representação das atividades gerais que consistem o processo geral de locação e devolução de um carro a partir de uma reserva prévia, conforme ilustrou a Figura 4.3, foram identificados os principais **requisitos funcionais** relacionados no Quadro 4.1 a seguir. No quadro foram listados apenas os requisitos funcionais referentes aos cadastros e controles dos processos de locação e devolução. As consultas e relatórios não foram listados, mas serão representados diretamente no Diagrama de Casos de Uso, na modelagem da próxima atividade.

Quadro 4.1 | Listagem dos Requisitos Funcionais

Nº	Requisito	Descrição
RF1	Cadastro de filial	O sistema deve prover um cadastro de filiais da locadora de veículos. O cadastro será mantido pelo administrador do sistema.

Nº	Requisito	Descrição
RF2	Cadastro de cliente	O sistema deve prover um cadastro de clientes. Um cliente pode se cadastrar via web ou aplicativo. Cada cliente deve ser gerenciado pela sua situação (ativo, preferencial, inadimplente ou encerrado).
RF3	Cadastro de empresa	O sistema deve prover um cadastro das empresas conveniadas com a locadora de veículos.
RF4	Cadastro de grupo de carro	O sistema deve prover um cadastro de grupos de carros para categorizar os carros por características.
RF5	Cadastro de carro	O sistema deve prover um cadastro dos carros disponíveis para o aluguel. Cada carro deve ser gerenciado pela sua situação (disponível, alugado, manutenção interna, manutenção externa ou encerrado).
RF6	Controle de reserva de carro	O sistema deve prover um controle de reserva de carros para clientes da locadora. Uma reserva pode ser realizada por telefone, web ou aplicativo. Cada reserva deve ser gerenciada pela situação (agendada, cancelada, realizada, aprovado ou reprovado). Toda reserva efetivada deve enviar um comprovante da reserva para o cliente via e-mail e/ou SMS.
RF7	Controle de aluguel de carro	O sistema deve prover um controle de aluguel de carros para clientes da locadora.

Ver anotações

Nº	Requisito	Descrição
RF8	Emissão de contrato de aluguel	O sistema deve emitir cópias do contrato de aluguel.
RF9	Controle de devolução de carro	O sistema deve prover um controle de devolução de carros. Uma devolução refere-se à baixa de um aluguel. O controle de devolução estará integrado com o módulo de pagamento.
RF10	Emissão de nota fiscal de aluguel	O sistema deve emitir a nota fiscal de serviço, referente ao aluguel de um carro. A nota fiscal pode ser impressa ou enviada por e-mail para o cliente.

Fonte: elaborado pela autora.

o
Ver anotações

ATIVIDADE ANÁLISE E PROJETO

Na segunda atividade do modelo Processo Unificado – Análise e Projeto, a modelagem da atividade de **Análise** consiste em identificar e definir o que o sistema de software deve fazer em uma visão lógica do negócio e a atividade de **Projeto** consiste em definir como será o desenvolvimento do software, em consonância com as tecnologias de linguagens de programação e banco de dados que serão adotados para implementação do software.

Em um processo de desenvolvimento iterativo a atividade de Análise precede o a atividade de Projeto, entretanto, a modelagem da análise e o projeto podem acontecer simultaneamente. Segundo Bezerra (2007, p. 176)

“

[...] os modelos especificados na análise esclarecem o problema a ser resolvido. No entanto, as perspectivas do sistema fornecidas por esses modelos não são suficientes para se ter uma visão completa do sistema para que a implementação comece.

Na atividade de análise, inicia-se a modelagem com a definição dos casos de uso, a partir dos requisitos funcionais identificados na atividade de Requisitos, especificando o Modelo de Casos de Uso. Posteriormente, considerando que o Modelo de Casos de Uso está pronto, a próxima etapa é analisar cada caso de uso e iniciar a identificação das classes de objetos, compreendendo qual classe ou quais classes participam da realização de um caso de uso e como o sistema será estruturado internamente, especificando o Modelo de Classes geralmente em várias perspectivas de visão. A partir da primeira versão do Modelo de Classes, é mais fácil complementar a modelagem dos casos de uso com a documentação descritiva de cada caso de uso, pois nesse momento já se identificou os objetos que colaboram e devem participar da execução de cada caso de uso.

■ MODELO DE CASOS DE USO

A partir da definição dos requisitos funcionais analisamos os requisitos identificados e abstraímos as regras de negócio do sistema, e assim foi decidido estruturar os casos de uso em dois pacotes.

Na UML, os agrupamentos que organizam um modelo (conjunto de diagramas) são chamados de pacotes.

■ DIAGRAMA DE PACOTES

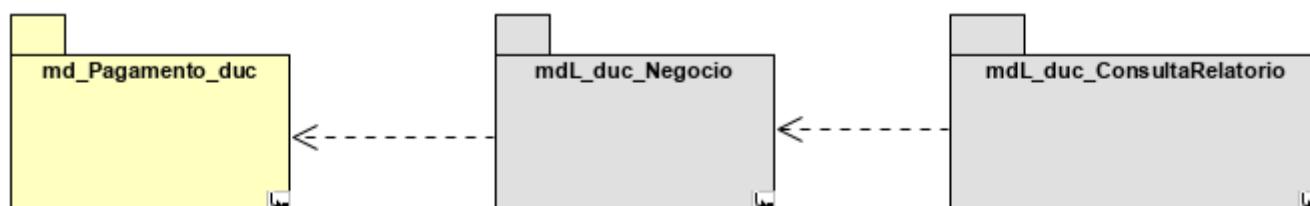
A técnica de modelagem estrutural – **Diagrama de Pacotes** demonstra os elementos do sistema agrupados e organizados em pacotes lógicos ou físicos, com o objetivo de representar os componentes ou módulos que integram um sistema e suas dependências. Para Bezerra (2014), um pacote é um mecanismo utilizado para agrupar elementos semanticamente relacionados, inclusive outros pacotes, sendo que as ligações entre pacotes são indicadas pelo relacionamento de dependência entre pacotes.

O Diagrama de Pacotes tem uma notação gráfica simplificada que demonstra como os elementos do sistema estão organizados em pacotes e suas dependências, a partir da categorização dos elementos em grupos que representam as partes do sistema, sendo que um pacote pode se subdividir em outros pacotes. O diagrama pode ser especificado de maneira independente ou associado a outros diagramas da UML principalmente para representar o Modelo de Casos de Uso e o Modelo de Classes.

Antes de iniciar a especificação dos casos de uso, é importante listar todos os casos de uso identificados, para tomar a decisão de agrupá-los ou não por categoria ou assunto e, com isso, desenhar o Diagrama de Pacotes e o Diagrama de Casos de Uso correspondentes a cada pacote, compondo o Modelo de Casos de Uso.

A Figura 4.4 exemplifica o Diagrama de Pacotes dos casos de uso. Foram definidos dois pacotes – “mdL_duc_Negocio” e “mdL_duc_ConsultaRelatorio”, correspondente ao módulo locação, e o pacote “md_Pagamento_duc”, correspondente ao módulo pagamento, que foi integrado ao modulo locação que está sendo especificado. Assim, os pacotes representados com suas dependências correspondem aos Diagramas de Casos de Uso que agruparam os casos de uso por assunto, com o objetivo de organizarem os casos de uso identificados, constituindo um Modelo de Casos de Uso.

Figura 4.4 | Diagrama de Pacotes – Modelo de Casos de Uso



Fonte: elaborada pela autora.

A Figura 4.5 ilustra o Diagrama de Casos de Uso correspondente ao pacote “mdL_duc_Negocio”, ilustrando os casos de uso correspondentes aos requisitos funcionais e outros casos de uso que foram identificados a partir da abstração desses requisitos, considerando também o contexto do domínio do sistema. Foram definidos os atores primários “Funcionário Adm”, “Cliente”, “Empresa” e “Gerente”, que interagem com os casos de uso, ou seja, o analista de sistemas deve identificar a pessoa, setor da empresa, um disposto ou outro sistema de software responsável em fornecer os dados para que ocorra a execução do caso de uso, caracterizando a interação com o sistema. É importante também estudar cada requisito funcional definido na atividade anterior e, assim, tomar a decisão de estabelecer os casos de uso e seus relacionamentos, pois um requisito funcional pode ser representado por um ou mais casos de uso. Recomendamos pensar no objetivo de cada caso de uso, considerando a usabilidade do sistema para os usuários.

A Figura 4.5 consiste na representação dos casos de uso base (casos de uso associados a atores) e alguns relacionamentos entre os casos de uso já foram definidos porque completam o papel da funcionalidade, considerando a sua execução. Observa-se essa situação no caso de uso “Reservar Carro”, pelo qual, a partir de uma reserva que um cliente realiza, o cliente receberá um comprovante por e-mail da reserva efetuada automaticamente pelo sistema, sendo representado no diagrama pelo relacionamento de inclusão <<Include>>, e se for de interesse do cliente, ele poderá imprimir o comprovante da reserva efetuada após finalizar a confirmação de reserva, sendo representado no diagrama pelo relacionamento de extensão <<Extend>>.

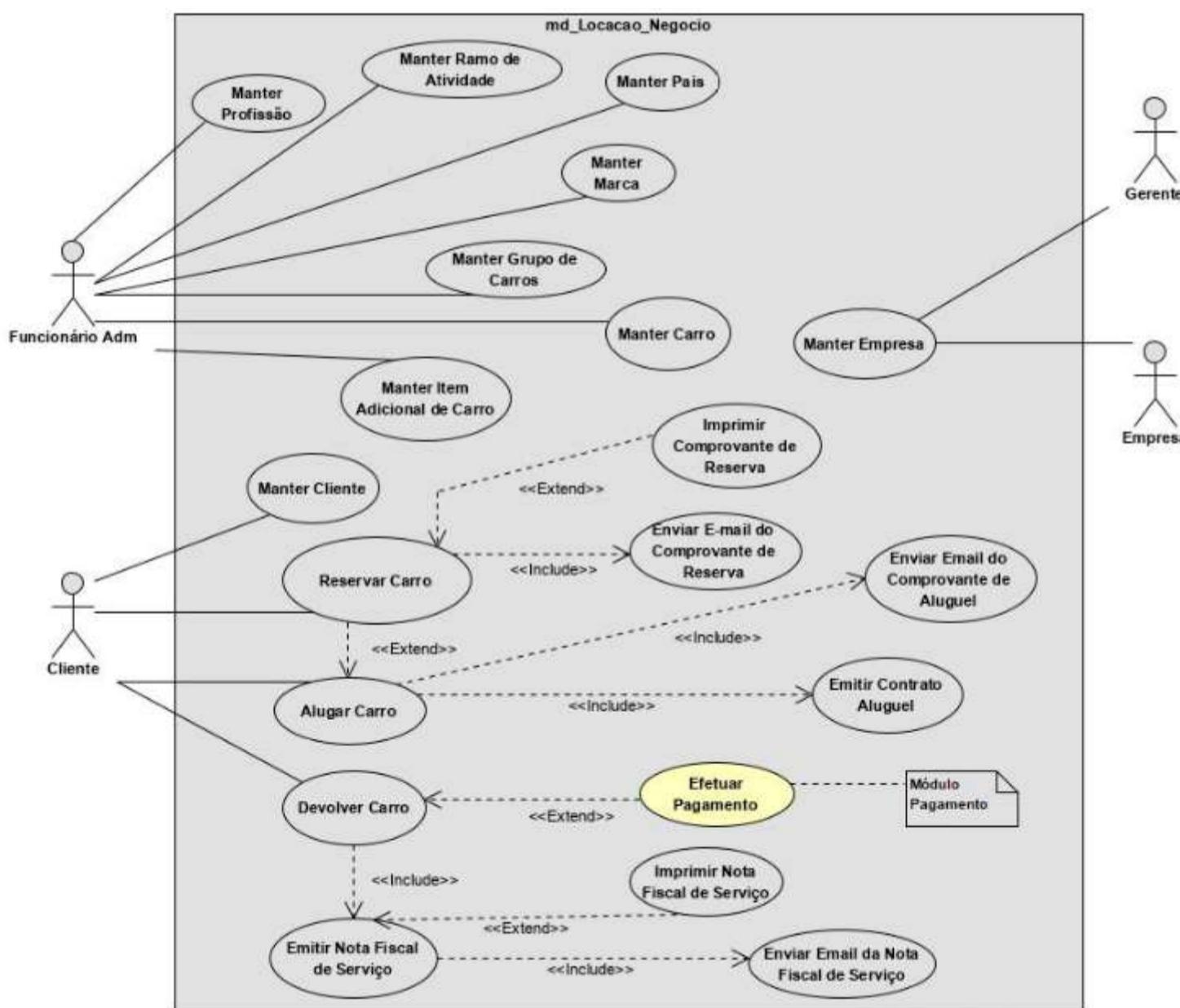
Lembre-se de que o relacionamento de extensão é um tipo de relacionamento de dependência existente somente entre casos de uso. O caso de uso estendido é uma descrição completa de uma sequência de interações, com significado completo de uma outra funcionalidade do sistema. O relacionamento de dependência, representado pelo estereótipo <<Extend>>, é indicado por uma seta que parte do caso de

uso “estendido” para o caso de uso “base”, que é o caso de uso que tem interação com o ator. E o relacionamento de inclusão é um tipo de relacionamento existente somente entre casos de uso para indicar a continuidade de execução obrigatória entre os casos de uso. O relacionamento de dependência, representado pelo estereótipo <<Include>>, é demostrado por uma seta direcionada do caso de uso “base” para o caso de uso “incluído”.

REFLITA

Se os tipos de relacionamento de extensão e inclusão são estabelecidos apenas entre os casos de uso, como podemos identificar os tipos de relacionamentos de extensão e inclusão entre os casos de uso?

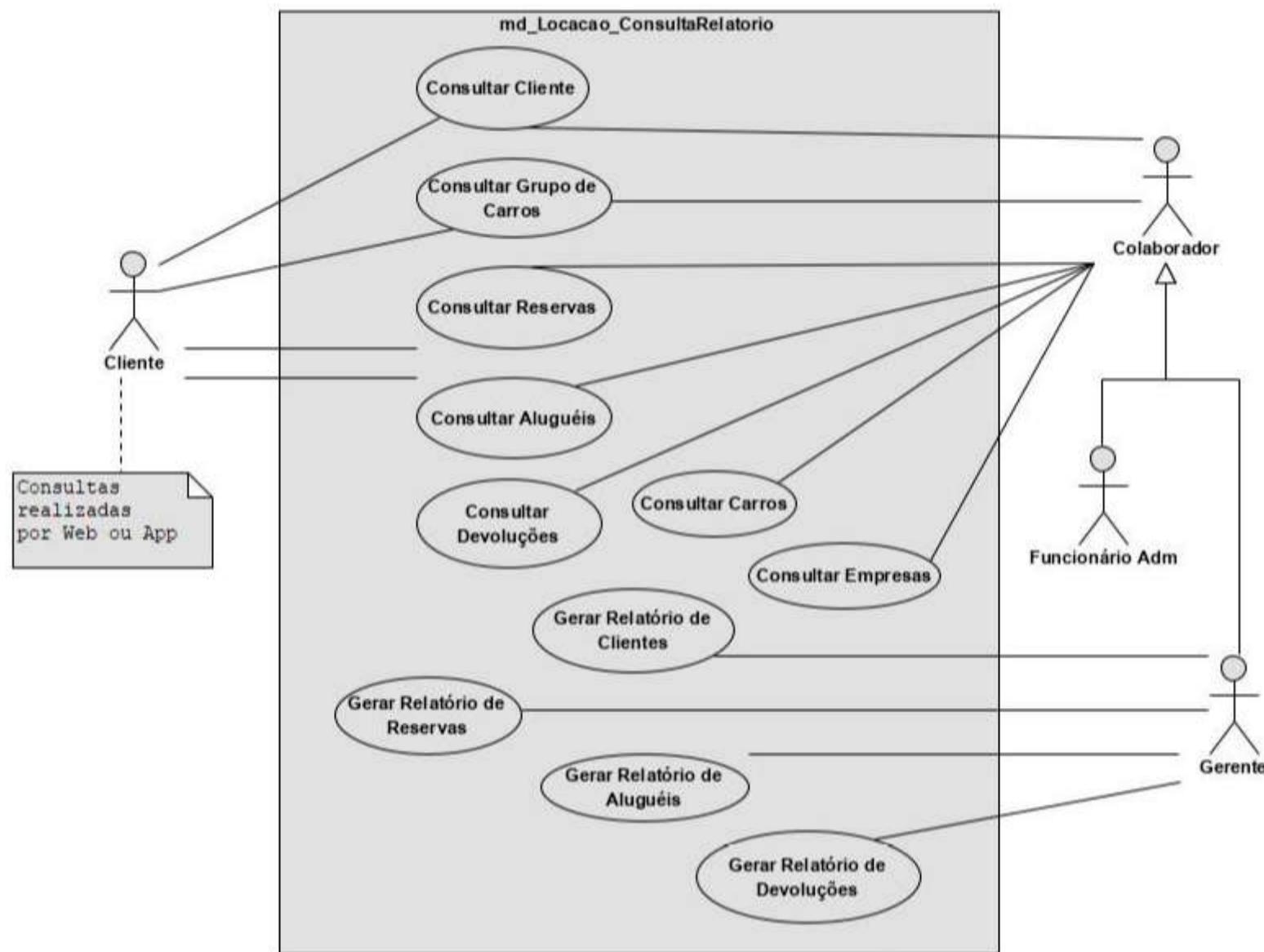
Figura 4.5 | Diagrama de Casos de Uso – Pacote mdL_duc_Negocio



Fonte: captura de tela do software Visual Paradigm Community Edition elaborada pela autora.

A Figuras 4.6 representa o Diagrama de Casos de Uso correspondente ao pacote “mdL_duc_ConsultaRelatorio”, que concentra os casos de uso definidos para as funcionalidades das consultas e relatórios operacionais do sistema. Perceba que alguns casos de uso estão associados ao ator genérico “Colaborador”, o qual indica que os atores especializados “Funcionário Adm” e “Gerente” têm acesso a todos os casos de uso associados ao ator genérico.

Figura 4.6 | Diagrama de Casos de Uso – Pacote mdL_duc_ConsultaRelatorio



Fonte: captura de tela do software Visual Paradigm Community Edition elaborada pela autora.

DIAGRAMA DE CLASSES

A partir da abstração dos casos de uso, iniciou-se a identificação das classes de objetos e a elaboração do Diagrama de Classes, que é considerada a principal técnica de modelagem estrutural da UML, representando a modelagem da parte estática do sistema e simbolizando um conjunto de classes com seus atributos, operações e relacionamentos.

Entre as diferentes técnicas de identificação de classes, como a da Análise Textual de Abbott; a da Análise dos Casos de Uso; a de Cartões Classes, Responsabilidades e Colaboradores (CRC); a da Taxonomia de Conceitos e outras; adotamos a técnica da Análise dos Casos de Uso. Assim, para cada caso de uso definido no Modelo de Casos de Uso, analisa-se qual ou quais objetos é preciso criar ou acessar durante a execução do caso de uso, ou seja, em tempo de execução do sistema. Por exemplo, se foi definido o caso de uso “Manter Cliente” é porque o

sistema deve permitir a manipulação dos objetos do tipo cliente com as operações básicas de inclusão, alteração, exclusão (ou controlar o status do objeto, ativo ou não) ou pesquisa, assim é necessário definir uma classe para esses objetos com os atributos e operações pertinentes ao contexto do domínio do sistema. Entretanto, pode acontecer de um ou mais casos de uso manipularem objetos da mesma classe ou vice-versa.

Após a identificação das classes e atributos deve-se verificar a consistência entre as classes e os casos de usos já definidos previamente e, assim, observar a necessidade de novas classes ou eliminar incoerências e redundâncias. A partir da elaboração de uma primeira visão do Diagrama de Classes, deve-se refiná-lo e incrementá-lo com novos detalhes correspondentes às tecnologias de implementação que serão adotadas, especificando o modelo ideal do Diagrama de Classes da atividade de Projeto.

Considerando o contexto do sistema de “Locação de Veículos”, decidimos organizar as classes em um único pacote “md_Locacao_dc” com dependência com o pacote “md_Pagamento_dc”, referente ao módulo de pagamentos, conforme ilustra o Diagrama de Pacotes da Figura 4.7.

Figura 4.7 | Diagrama de Pacotes



Fonte: elaborada pela autora.

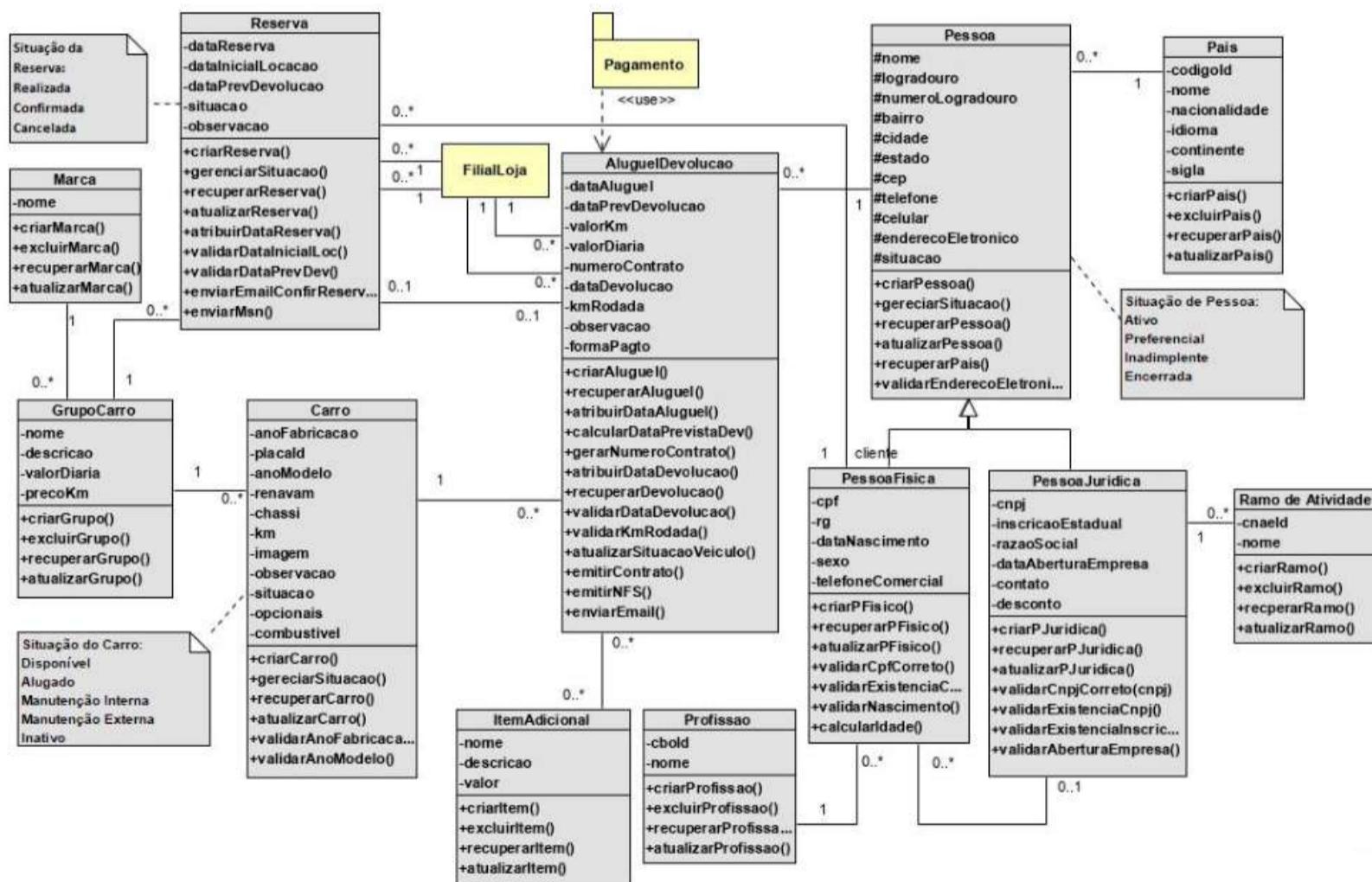
A Figura 4.8 representa o Diagrama de Classes de análise correspondente ao pacote “md_Locacao_dc”, respeitando as regras básicas de modelagem do diagrama e as regras de negócio do sistema.

No Diagrama de Classes, além da representação das classes, forma estabelecidos os relacionamentos entre as classes, os quais indicam o compartilhamento de informações entre os objetos das classes por

meio da troca de mensagens entre os objetos, em tempo de execução do sistema. Observa-se no diagrama da Figura 4.8 o relacionamento do tipo generalização, estabelecido entre a superclasse “Pessoa” e as subclasses “PessoaFisica” e “PessoaJuridica”, e todos os demais relacionamentos estabelecidos entre as classes são do tipo associação binária.

A classe “FilialLoja” está destacada em outra cor e não foram listados os atributos e operações porque é uma classe que existe no sistema, mas seus objetos são criados e manipulados diretamente no banco de dados pelo administrador do sistema. Há necessidade, contudo, de representar essa classe para estabelecer a associação com as classes “Reserva” e “AluguelDevolucao”.

Figura 4.8 | Diagrama de Classes (Análise) – md_Locacao_dc



Fonte: captura de tela do software Visual Paradigm Community Edition elaborada pela autora.

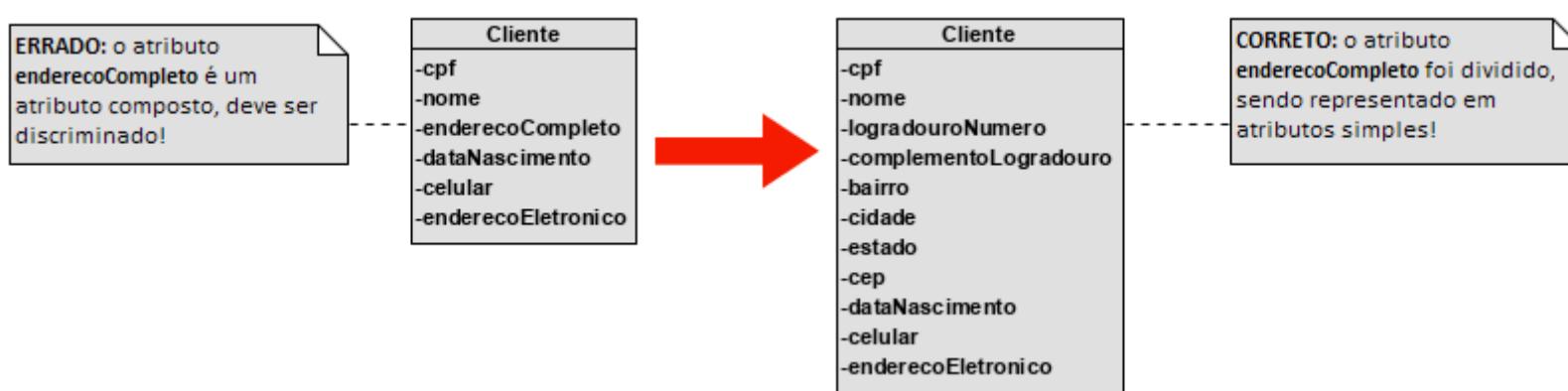
REFLITA

Na definição dos relacionamentos do Diagrama de Classes, existe uma regra ou padrão para definir a multiplicidade do relacionamento do tipo associação?

Lembre-se da nomenclatura recomendada para definir o nome das classes, de seus atributos e operações. Na primeira parte exibe-se o nome da classe, geralmente um substantivo ou expressões breves, devendo ser único no modelo de classes. Por convenção, o nome é apresentado no singular e quando há palavras compostas usa-se concatená-las, começando cada palavra por letra maiúscula. Na segunda parte, são declarados os atributos que representam os dados do objeto. São nomeados por substantivos ou expressões que representam alguma propriedade da classe, tipicamente em letra minúscula. Quando há palavras compostas usa-se concatená-las, sendo que a partir da segunda palavra o termo é iniciado com letra maiúscula, por exemplo: *dataNascimento* e *endereçoEletrônico*. Na terceira parte são declaradas as operações que correspondem às ações dos objetos, nomeadas por um verbo ou uma locução verbal breve, usando a mesma convenção de letras minúsculas e maiúsculas dos atributos, como *criarCliente*, *alterarCliente*, *validarDataNascimento* e *calcularIdade*.

Um ponto de atenção que vale a pena reforçar é a definição e listagem dos atributos de uma classe. Lembre-se de que todo atributo definido para uma classe tem que ser um atributo atômico – também denominado de simples –, ou seja, não é possível dividir o atributo em outros campos, como CPF, estado civil, placa, ano do modelo etc. Diferentemente de um atributo composto, por exemplo, filiação, que é composto por nome da mãe e nome do pai; endereço, que é composto de nome do logradouro, número do logradouro, complemento, bairro, cidade, estado e CEP; período da inscrição de um evento, composto de data de início e data de término. Então, em uma classe, está errado listar um atributo composto, conforme mostra a Figura 4.9.

Figura 4.9 | Classes com Atributo Composto e Atributo Simples



Fonte: elaborada pela autora.

Alguns atributos compostos podem ser definidos como um atributo simples na representação de uma classe, como no atributo nome de uma pessoa. Nesse caso, é necessário também se atentar ao contexto do domínio do sistema, ou seja, às vezes, para o domínio de um sistema, é possível considerar o atributo nome de uma pessoa como sendo um atributo simples, mantendo em um único atributo o nome e sobrenome da pessoa; e para outro contexto, é importante definir o atributo nome de uma pessoa separado do sobrenome da pessoa.

EXEMPLIFICANDO

Um tipo de associação importante na definição dos relacionamentos entre as classes do Diagrama de Classes é o tipo chamado Agregação, conhecido como associação “Todo-Parte”. Demonstra que as informações de um objeto (objeto-todo) precisam ser complementadas pelas informações contidas nos objetos da outra classe (objetos-partes) relacionada, representando que ambos os objetos das classes podem “viver” de forma independente, ou seja, não existe uma ligação forte entre eles; contudo são partes de um único todo.

A notação da agregação é representada por um losango na extremidade da classe que contém o “objeto-todo”, conforme mostra o exemplo da Figura 4.10 a seguir, demonstrando que um objeto “Pais” é composto por nenhum ou vários objetos “Estado” e um estado é composto por nenhum ou vários

objetos “Cidade”, ou seja, eu posso criar um objeto “Pais” e posteriormente, em outro momento, criar o objeto que representa as suas partes, o objeto “Estado”.

Figura 4.10 | Relacionamento tipo agregação

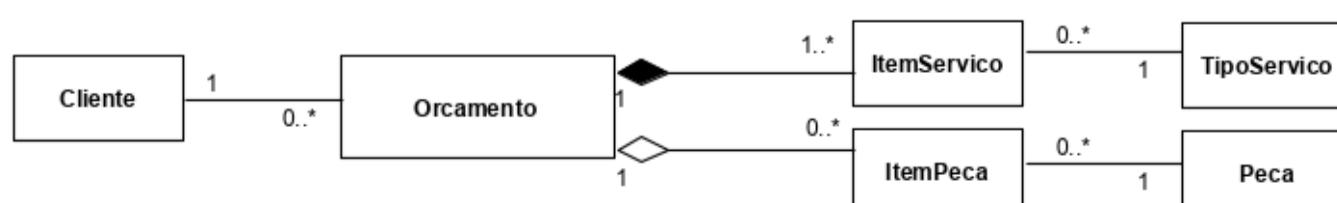


Fonte: elaborada pela autora.

E um tipo especial, uma variação da associação agregação, é a chamada Composição. Na associação desse tipo estabelece-se um vínculo mais forte entre o “objeto-todo” e os “objetos-partes”, demonstrando que os objetos-parte integram um único “objeto-todo”. Assim, os “objetos-partes” não podem viver quando o “objeto-todo” for destruído.

Utiliza-se um losango sólido para indicar a composição, conforme representa o exemplo a seguir. No exemplo da Figura 4.11, um objeto “Orcamento” é composto de no mínimo um objeto “ItemServico” ou de muitos, mas obrigatoriamente todo orçamento deve ter um item de serviço, senão não é um orçamento. E, no caso da composição, o objeto que representa o todo “Orcamento” deve ser criado, e os objetos-partes “ItemServico” devem ser criados na sequência, não podendo ser criados em outro momento de execução do sistema. Essa característica diferencia a Composição da Agregação.

Figura 4.11 | Relacionamento tipo composição



Fonte: elaborada pela autora.

DOCUMENTAÇÃO DE CADA CASO DE USO

Na sequência, agora que já temos a primeira versão do Diagrama de Classes, fica mais fácil elaborar a documentação de cada caso de uso. Não existe um formato específico de documentação para os casos de uso definido pela UML. O formato de documentação de um caso de uso é flexível, permitindo que se documente o caso de uso da forma que se considerar melhor, até mesmo com o uso de pseudocódigo ou de código de uma linguagem de programação. Outra opção é utilizar a prototipação da interface gráfica para facilitar a compreensão da execução do caso de uso.

O formato sugerido para documentação dos casos de uso é o formato numerado de descrição dos cenários – principal e alternativos. O cenário principal relata a descrição de uma tarefa que represente o mundo perfeito, sem exceções, e a descrição dos cenários alternativos relata qualquer situação que represente uma exceção (condição) do cenário principal. A seguir é demonstrada a documentação do caso de uso “Manter Empresa”.

Documentação do Caso de Uso Manter Empresa:

Número: 04

Use Case: Manter Empresa

Descrição: Empresa informa os seus dados para realização do seu cadastro como empresa conveniada da locadora de veículos

Autor Principal: Empresa

Cenário Principal:

1. Empresa informa seus dados para realização do seu cadastro.
2. Sistema exibe o cadastro de empresas.
3. Empresa informa seus dados.
4. Sistema verifica se empresa não está cadastrada.

5. Sistema verifica que ramo de atividade que referencia a empresa está cadastrado.
6. Sistema recupera dados do ramo de atividade vinculado à empresa.
7. Sistema verifica que país que referencia a empresa está cadastrado.
8. Sistema recupera dados do país vinculado à empresa.
9. Empresa confirma o seu cadastro.
10. Sistema registra a empresa.
11. Sistema emite Msg, informando que a empresa foi cadastrada com sucesso.
12. Sistema encerra o caso de uso.

Cenário Alternativo 4:

4. Sistema verifica que existe empresa cadastrada.
 - 4.1 Sistema recupera dados da empresa cadastrada.
 - 4.2 Sistema permite alteração ou exclusão da empresa.
 - 4.3 Empresa escolhe a opção de alteração.
 - 4.4 Empresa informa dados a serem alterados.
 - 4.5 Empresa confirma alteração dos seus dados.
 - 4.6 Sistema atualiza dados da empresa.
 - 4.7 Sistema encerra o caso de uso.

Cenário Alternativo 4.3:

- 4.3. Empresa escolhe a opção “a empresa não está associada a outro objeto”.
 - 4.3.2. Sistema emite Msg, confirmando a exclusão da empresa.
 - 4.3.3. Empresa confirma a exclusão da empresa.

4.3.4. Sistema exclui a empresa.

4.3.5. Sistema encerra o caso de uso.

Cenário Alternativo 4.3.1:

4.3.1 Sistema valida que a empresa está associada a outro objeto.

4.3.1.1 Sistema emite Msg, informando que a empresa não pode ser excluída.

4.3.1.2 Sistema encerra o caso de uso.

Cenário Alternativo 6:

6. Sistema verifica que o ramo de atividade não está cadastrado.

6.1 Sistema permite a execução do use case “Manter Ramo de Atividade”.

Cenário Alternativo 8:

8. Sistema verifica que país não está cadastrado.

8.1 Sistema permite a execução do use case “Manter País”.

Cenário Alternativo 9:

9. Empresa não confirma o seu cadastro.

9.1 Empresa cancela o cadastro.

9.2 Sistema emite Msg, informando que o cadastro será cancelado.

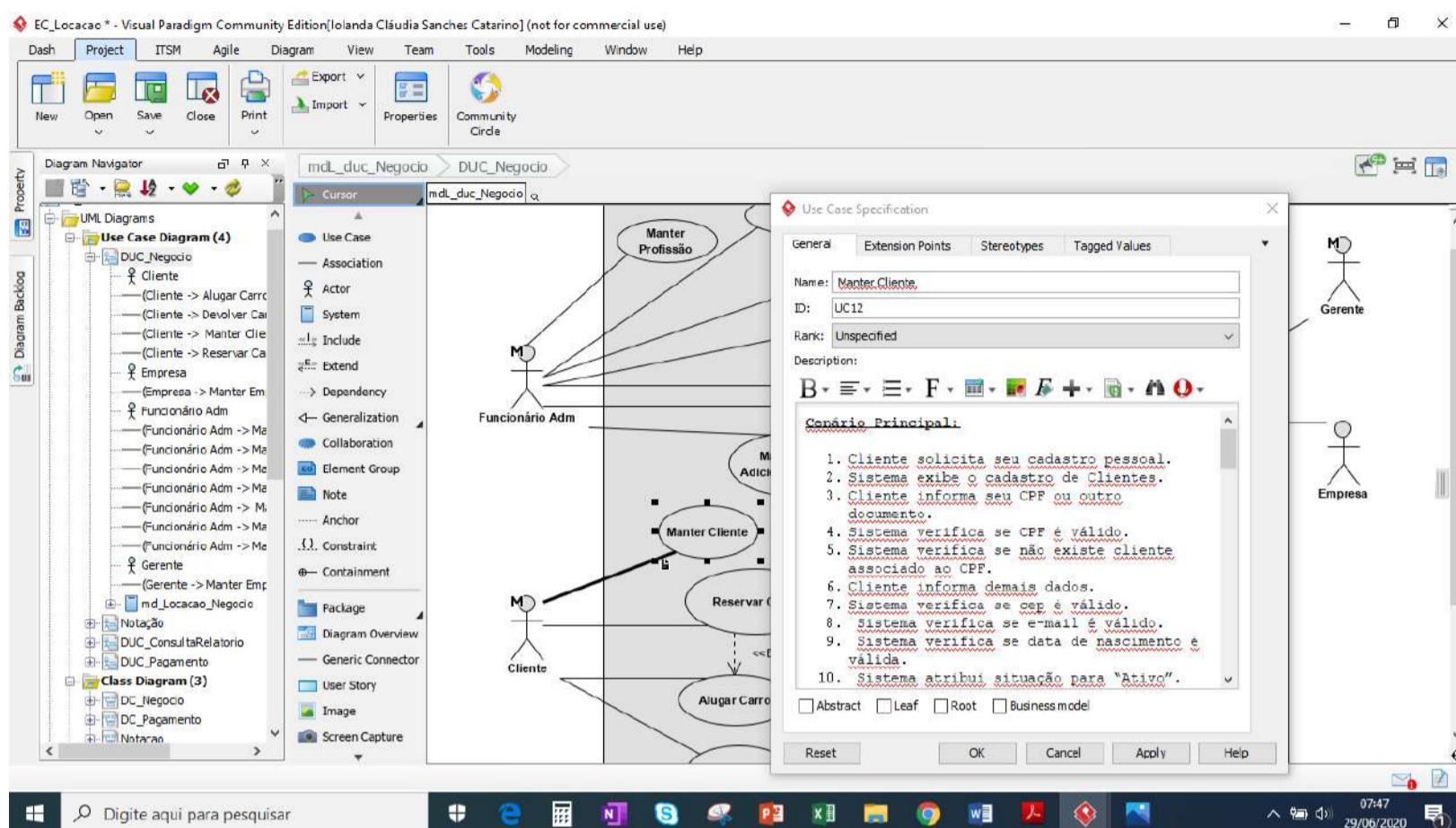
9.3 Sistema encerra o caso de uso.

A documentação do caso de uso também pode ser feita diretamente na ferramenta CASE de modelagem, vinculada à representação do caso de uso, conforme mostra a Figura 4.12, que ilustra a documentação do caso de uso “Manter Cliente”. Na ferramenta CASE adotada – Visual Paradigm, deve-se selecionar o caso de uso, abrir suas propriedades na

opção “*Use Case Specification*”, acessando o menu com o botão direito do mouse e no item “*Description*” descrever o relato do funcionamento do caso de uso.

o

Figura 4.12 | Documentação do Caso de Uso na Ferramenta CASE



Fonte: captura de tela do software Visual Paradigm Community Edition elaborada pela autora.

Posteriormente, deve-se continuar a modelagem comportamental e de interação do sistema, a partir do detalhamento do funcionamento dos casos de uso, adotando o Diagrama de Atividades, Diagrama de Sequência ou o Diagrama de Visão Geral de Interação, bem como especificar os estados e suas transições dos objetos das classes que apresentam estados relevantes, como é o caso das classes “Reserva”, “Carro” e “Pessoa”.

REFERÊNCIAS

BEZERRA, E. **Princípios de análise e projeto de sistemas com UML**. 3. ed. Rio de Janeiro: Elsevier, 2014.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **UML**: guia do usuário. 2. ed. Rio de Janeiro: Campus, 2006.

GUERRINI, F. M. et al. **Modelagem da Organização**: uma visão integrada. Porto Alegre: Bookman, 2014.

Ver anotações

KIRCHOFF, E. **BPMN em exemplos:** aprenda como modelar processos de negócio. Kirchoff, 2015.

MEDEIROS, E. S. **Desenvolvendo software com UML 2.0:** definitivo. São Paulo: Pearson, 2004.

PRESSMAN, R.; MAXIM, B. **Engenharia de software:** uma abordagem profissional. 8 ed. Porto Alegre: AMGH, 2016.

SOMMERVILLE, I. **Engenharia de software.** 10. ed. São Paulo: Pearson Education do Brasil, 2018.

FOCO NO MERCADO DE TRABALHO

MODELAGEM INICIAL DA ATIVIDADE DE ANÁLISE

Iolanda Cláudia Sanches Catarino

Ver anotações

DIAGRAMA DE CLASSES

Desenvolvimento do Diagrama de Classes (Análise) de acordo com as regras de negócio definidas na descrição do estudo de caso.



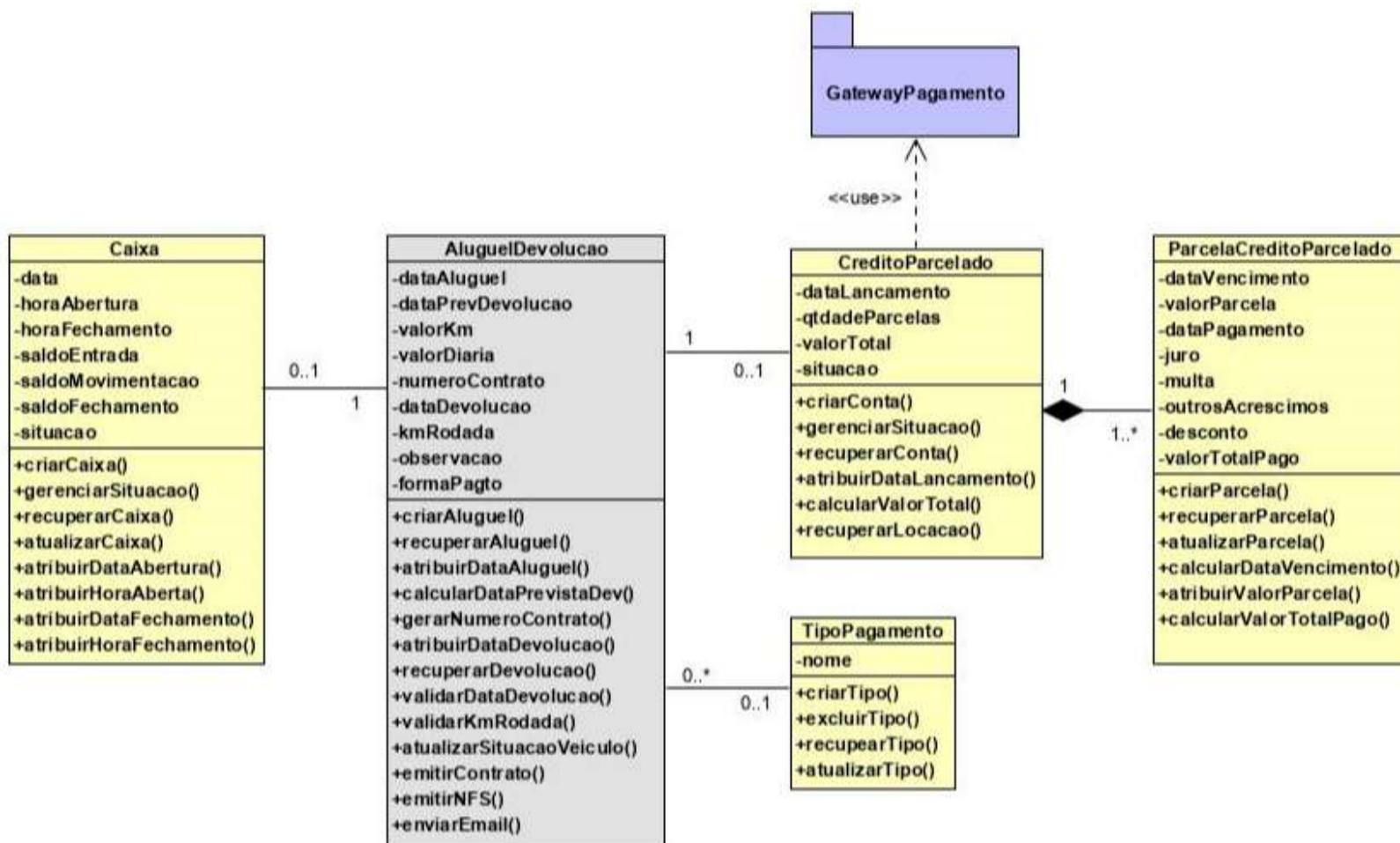
Fonte: Shutterstock.

Deseja ouvir este material?

Áudio disponível no material digital.

Considerando as regras de negócio definidas no complemento da descrição do estudo de caso, para o módulo “Pagamento” segue o Diagrama de Classes como proposta de uma solução possível.

Figura 4.13 | Diagrama de Classes (Análise) – md_Pagamento_dc



Fonte: captura de tela do software Visual Paradigm Community Edition elaborada pela autora.

Observa na figura que foi representado a classe “AluguelDevolucao”, que faz parte do módulo “Locação de Veículos”, e as classes “Caixa”, “CreditoParcelado”, “ParcelaCreditoParcelado” e “TipoPagamento”, que são as classes especificadas para o módulo “Pagamento”.

A classe “CreditoParcelado” foi representada com uma associação do tipo composição, indicando objetos todo-parte, sendo que os objetos da “ParcelaCreditoParcelado” representam as partes com a obrigação de, no mínimo, ter uma parcela. Foi estabelecido também um relacionamento do tipo dependência entre a classe “CreditoParcelado” com o pacote “GatewayPagamento”, o qual indica que para as compras lançadas como crédito a receber (contas a receber), a partir do pagamento efetuado com cartão de débito ou crédito, é reutilizado um gateway de pagamento (um serviço destinado a pagamentos virtuais por

cartão, mantido por uma operadora financeira que autoriza pagamentos de transações feitas on-line em websites de empresas) para efetuar esses pagamentos.

NÃO PODE FALTAR

MODELAGEM COMPLEMENTAR DA ATIVIDADE DE ANÁLISE

Iolanda Cláudia Sanches Catarino

Ver anotações

MODELAGEM DE COLABORAÇÕES, COMPORTAMENTAL E DE INTERAÇÃO DO SISTEMA

Modelagem a partir do Diagrama de Estrutura Composta das principais colaborações do sistema e, posteriormente a modelagem comportamental e de interação do sistema com a especificação dos principais Diagramas de Atividades, Diagramas de Máquina de Estados e Diagramas de Sequência.



Fonte: Shutterstock.

[Deseja ouvir este material?](#)

PRATICAR PARA APRENDER

Caro aluno, seja bem-vindo à seção sobre a modelagem complementar da atividade de Análise e Projeto do estudo de caso proposto!

Nesta seção continuamos a modelagem estrutural do sistema “Locação de Veículos”, a partir do Diagrama de Estrutura Composta. Na sequência, enfatizamos a modelagem dinâmica do sistema, considerando a adoção do Diagrama de Máquina de Estados para especificar os estados relevantes e suas transições de estados dos objetos das classes já identificadas no Diagrama de Classes. Por fim, adotamos o Diagrama de Atividades e o Diagrama de Sequência para documentar a realização dos principais casos de uso do sistema.

O Diagrama de Estrutura Composta é um novo diagrama estrutural da UML 2.0, que representa as colaborações entre elementos que cooperam entre si para executar uma função específica, ou seja, a realização de um ou mais casos de uso. O diagrama apresenta exatamente quais objetos são manipulados na execução de um caso de uso, assim, é importante adotá-lo para representar a colaboração entre os objetos de casos de uso mais complexos ou exatamente os casos de uso que representam as principais funcionalidades de negócio do sistema. Por exemplo, dos casos de uso identificados no sistema “Locação de Veículos”, foi elaborado o Diagrama de Estrutura Composta para os casos de uso “Reservar Carro”, “Alugar Carro” e “Devolver Carro”.

O Diagrama de Máquina de Estados mostra o comportamento de um elemento a partir de um conjunto finito de estados e suas transições de estado que expressam o comportamento de um caso de uso ou dos objetos de uma classe. O diagrama foi elaborado para as classes de objetos que identificamos e definimos estados relevantes, sendo as classes “Reserva”, “Pessoa”, “Carro” e “AluguelDevolucao”.

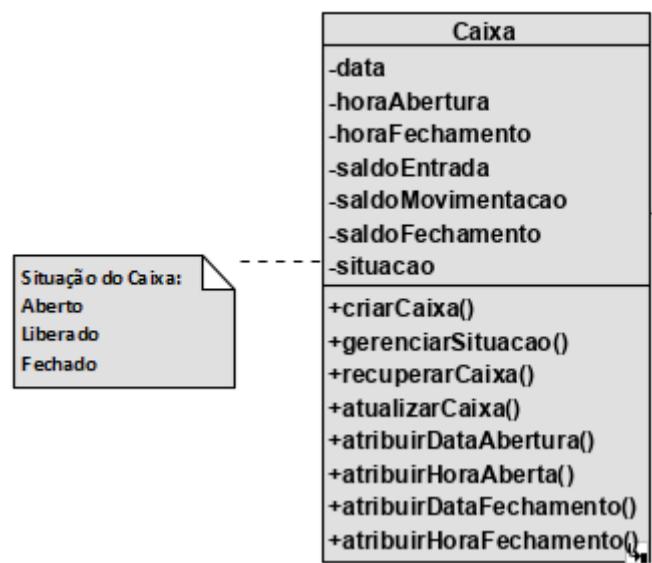
O Diagrama de Atividades mostra o fluxo de controle de um conjunto de atividades que representa a execução do sistema completo de processos de negócio, de casos de uso ou de até mesmo de um procedimento específico, referente a uma funcionalidade. Na modelagem da UML, o diagrama é fortemente recomendado para especificar os casos de uso, demonstrando, assim, o fluxo das ações necessárias para realizar o caso de uso. O Diagrama de Atividades foi elaborado correspondendo aos casos de uso “Reservar Carro” e “Acessar Conta Cliente”.

Finalizamos a modelagem aplicada à análise do sistema com o Diagrama de Sequência. Esse é um importante diagrama comportamental da UML que representa a ordem temporal em que as mensagens são trocadas entre os objetos envolvidos na execução de um processo, ou seja, de um caso de uso. Assim, o diagrama foi elaborado para especificar a realização dos casos de uso “Alugar Carro” e “Devolver Carro”.

Na Seção 4.1 você iniciou a modelagem estrutural do módulo “Pagamento”, que está integrado ao sistema de “Locação de Veículos”. Agora você, como analista de sistemas responsável por esse módulo, precisa avançar com a modelagem comportamental.

Na solução apresentada para a especificação do Diagrama de Classes, você observou que definimos um atributo situação para a classe “Caixa”, conforme ilustra a Figura 4.14. Então, vamos considerar que os objetos dessa classe apresentam os estados definidos a seguir, para o melhor controle e agilidade das consultas e relatórios gerados sobre os pagamentos efetuados.

Figura 4.14 | Classe Caixa – Módulo Pagamento



Fonte: elaborada pela autora.

Para os objetos da classe “Caixa”, os estados definidos são: Aberto, Liberado e Fechado. As regras definidas são as seguintes:

- Todo Caixa ao ser cadastrado deve assumir automaticamente a situação de Aberto.
- Todo Caixa Aberto deve assumir a situação de Liberado a partir das 8h ou sob a autorização de um gerente.
- Todo Caixa Liberado deve assumir a situação de Fechado a partir das 18h ou sob a autorização de um gerente.
- Todo Caixa Fechado deve assumir a situação de Aberto a partir das 6h do próximo dia.

1. Considerando as regras definidas para a classe “Caixa”, elabore o Diagrama de Máquina de Estados.
2. Considerando os diagramas comportamentais apresentados nesta seção e os fragmentos de interação apresentados nos diagramas, elabore o Diagrama de Visão Geral de Interação (com Quadros de Ocorrência de Interação do Diagrama de Sequência) correspondente ao processo de locar um carro, envolvendo o cadastro de um cliente, a reserva de um carro, o aluguel de um carro e o processo que envolve a devolução do carro.

Reforce o seu aprendizado com a modelagem do projeto. Bom trabalho!

Considerando que o Diagrama de Casos de Uso e o Diagrama de Classes da atividade de Análise estão prontos, avançamos com a modelagem do sistema “Locação de Veículos”, enfatizando a modelagem comportamental das funcionalidades do sistema. Assim, vamos aplicar as principais técnicas de modelagem comportamentais da UML para representar o comportamento e a interação entre os elementos do sistema e, dessa forma, documentar a perspectiva da visão dinâmica do sistema.

Conforme a classificação das técnicas de modelagem da UML em estruturais e comportamentais, a perspectiva de interação representa um subgrupo dos diagramas comportamentais, que mostra a interação de como os objetos do sistema se comunicam, apoiando a realização das funcionalidades representadas pelos casos de uso. Para facilitar a compreensão da interação entre os objetos que participam da realização dos casos de uso correspondentes ao negócio do domínio do sistema – reserva, aluguel e devolução –, vamos primeiramente representar o Diagrama de Estrutura Composta.

| DIAGRAMA DE ESTRUTURA COMPOSTA

O Diagrama de Estrutura Composta é um diagrama estrutural da UML que visa identificar a arquitetura do conjunto de elementos que interagem entre si durante a execução do sistema, formando uma colaboração entre esses elementos que se comunicam, contudo não especifica o comportamento da colaboração, que é o objetivo dos diagramas comportamentais da UML.

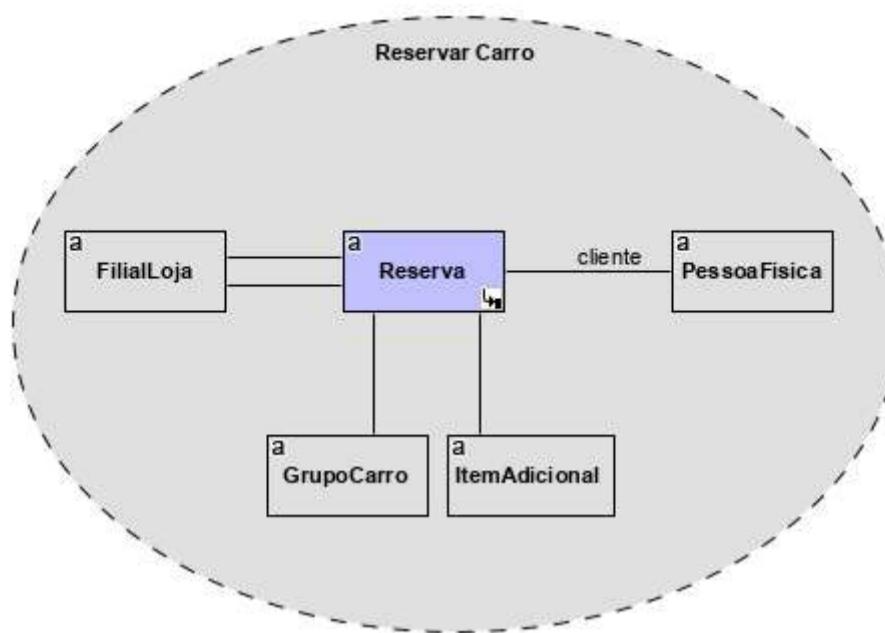
Segundo Guedes (2018), o Diagrama de Estrutura Composta representa a estrutura interna de um componente, classe ou uma colaboração entre um conjunto de instâncias que cooperam entre si para realizar uma tarefa, a partir da descrição das partes que o compõem e se comunicam, ou seja, a estrutura refere-se a uma composição de elementos interconectados por vínculos de comunicação que colaboram

entre si para atingir um objetivo. Assim, foi adotado o Diagrama de Estrutura Composta para representar a colaboração dos casos de uso “Reservar Carro”, “Alugar Carro” e “Devolver Carro”.

A Figura 4.15 ilustra o Diagrama de Estrutura Composta correspondente à colaboração denominada “Reservar Carro”, que abrange o caso de uso “Reservar Carro” e seus relacionamentos – “Imprimir Comprovante de Reserva” e “Enviar E-mail do Comprovante da Reserva”. Observe que o diagrama representa a colaboração, considerando que durante a execução dos casos de uso que integram a colaboração, cada reserva é instanciada na classe reserva (Reserva) e, para cada reserva, estabeleceu-se um vínculo que apresenta a comunicação entre os objetos que interagem para atingir o objetivo de efetivação de uma reserva. Dessa forma, comprehende-se que cada reserva pode referenciar:

- O objeto cliente (PessoaFisica), indicando quem está efetuando uma reserva.
- O objeto loja (FilialLoja), indicando em qual loja será feita a retirada do carro.
- Outro objeto loja (FilialLoja), indicando em qual loja será feita a devolução do carro.
- O objeto grupo de carro (GrupoCarro), indicando qual modelo de carro será escolhido para alugar.
- O objeto item adicional (ItemAdicional), indicando os itens adicionais, se necessário, que podem ser escolhidos para complementar o modelo do carro que será alugado.

Importa destacar que se um Diagrama de Estrutura Composta elaborado corresponde à colaboração de casos de uso, o nome da colaboração pode ser o mesmo de um caso de uso porque uma colaboração pode representar exatamente um único caso de uso ou mais de um caso de uso, desde que sejam casos de uso relacionados



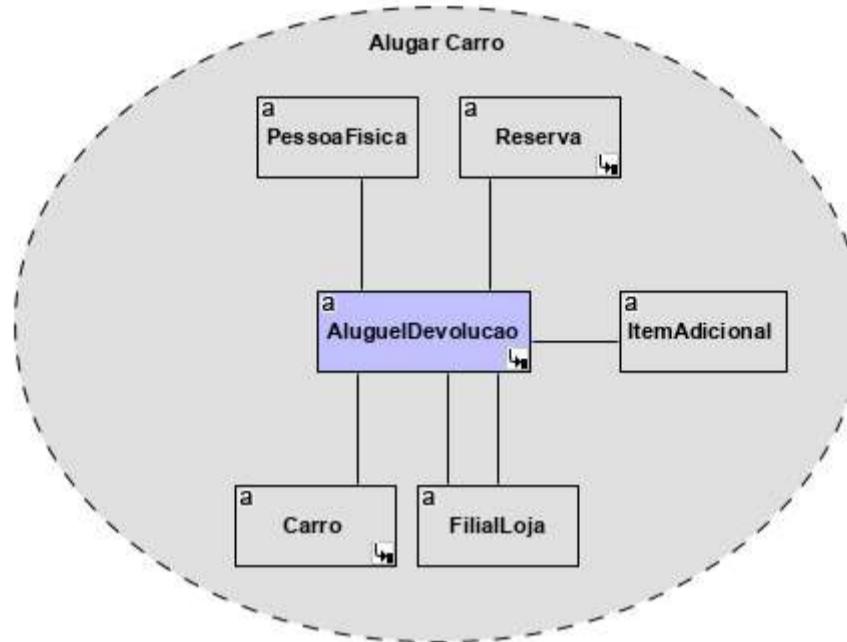
Fonte: elaborada pela autora.

A Figura 4.16 ilustra o Diagrama de Estrutura Composta correspondente à colaboração denominada “Alugar Carro”, que abrange o caso de uso “Alugar Carro” e seus relacionamentos – “Reservar Carro”, “Emitir Contrato de Aluguel” e “Enviar E-mail do Comprovante do Aluguel”. Observe que o diagrama representa a colaboração, considerando que durante a execução dos casos de uso que integram a colaboração, cada aluguel é instanciado na classe aluguel (Aluguel) e, para cada aluguel, estabeleceu-se um vínculo que demonstra a comunicação entre os objetos que interagem para atingir o objetivo de efetivação de um aluguel. Dessa forma, comprehende-se que cada aluguel pode referenciar:

- O objeto reserva (Reserva), indicando se o aluguel será realizado com base em uma reserva prévia.
- O objeto cliente (PessoaFisica), indicando quem está efetuando o aluguel.
- O objeto loja (FilialLoja), indicando a loja em que foi retirado o carro.
- Outro objeto loja (FilialLoja), indicando em qual loja será feita a devolução do carro.
- O objeto carro (Carro), indicando qual carro foi escolhido para alugar e, posteriormente com a confirmação do aluguel, a situação do carro alugado deve ser atualizada para “Alugado”.

- O objeto item adicional (ItemAdicional), indicando os itens adicionais, se necessário, que podem ser escolhidos para complementar o carro alugado.

Figura 4.16 | Diagrama de Estrutura Composta – Alugar Carro

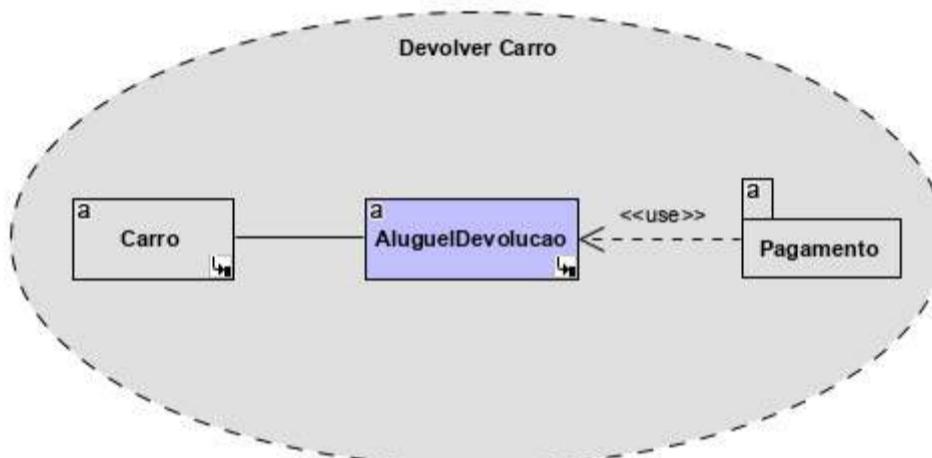


Fonte: elaborada pela autora.

E a Figura 4.17 ilustra o Diagrama de Estrutura Composta correspondente à colaboração denominada “Devolver Carro” que abrange o caso de uso “Devolver Carro” e seus relacionamentos – “Efetuar Pagamento” e “Emitir Nota Fiscal de Serviço”. Observe que o diagrama representa a colaboração, integrando seus elementos que participam da baixa de um objeto aluguel, assim, cada devolução refere-se:

- À recuperação do objeto aluguel (AluguelDevolucao) do qual será realizada a baixa – devolução.
- A dependência com o pacote pagamento “Pagamento”, indicando a integração com outro módulo do sistema.
- O objeto carro (Carro), indicando a comunicação para atualizar a situação do carro devolvido para “Manutenção Interna” ao ser confirmada a devolução do carro.

Figura 4.17 | Diagrama de Estrutura Composta – Devolver Carro



Fonte: elaborada pela autora.

A especificação do Diagrama de Estrutura Composta é importante para apoiar a modelagem comportamental do sistema e, assim, consolidar o entendimento de como funcionará em tempo de execução do sistema cada caso de uso, contribuindo para a melhor compreensão do contexto do sistema.

ASSIMILE

O Diagrama de Estrutura Composta representa as colaborações entre elementos que cooperam entre si para executarem uma função específica a partir da descrição das partes que o compõem e se comunicam. É um novo diagrama da UML 2.0 com notação gráfica simplificada, contudo ilustra exatamente os objetos das classes que participam da realização de um caso de uso ou de vários que integram uma colaboração.

DIAGRAMA DE MÁQUINA DE ESTADOS

Na sequência, vamos iniciar a modelagem dos estados de objetos que têm estados relevantes. Diante do contexto e de algumas regras de negócio já definidas na descrição do estudo de caso, as classes de objetos identificadas com estados relevantes no Diagrama de Classes especificado são “Reserva”, “Carro” e “Pessoa”. Entretanto, para melhor controle e agilidade das consultas e relatórios, também foi definida a classe “AluguelDevolucao”, com estados relevantes.

Segundo Booch, Jacobson e Rumbaugh (2006, p. 285), o Diagrama de Máquina de Estados representa “[...] um comportamento que especifica as sequências de estados pelos quais um objeto passa durante seu tempo de vida em resposta a eventos, juntamente com suas respostas a esses eventos”.

Lembre-se de que na elaboração do Diagrama de Máquina de Estados é fundamental identificar as regras de negócio aplicadas ao contexto dos objetos com estados relevantes, definindo consistentemente os estados e suas transições de estados, que são os elementos básicos do diagrama.

A seguir, são apresentadas as regras de negócio que indicam as transições de estados de cada classe e o seu Diagrama de Máquina de Estados correspondente.

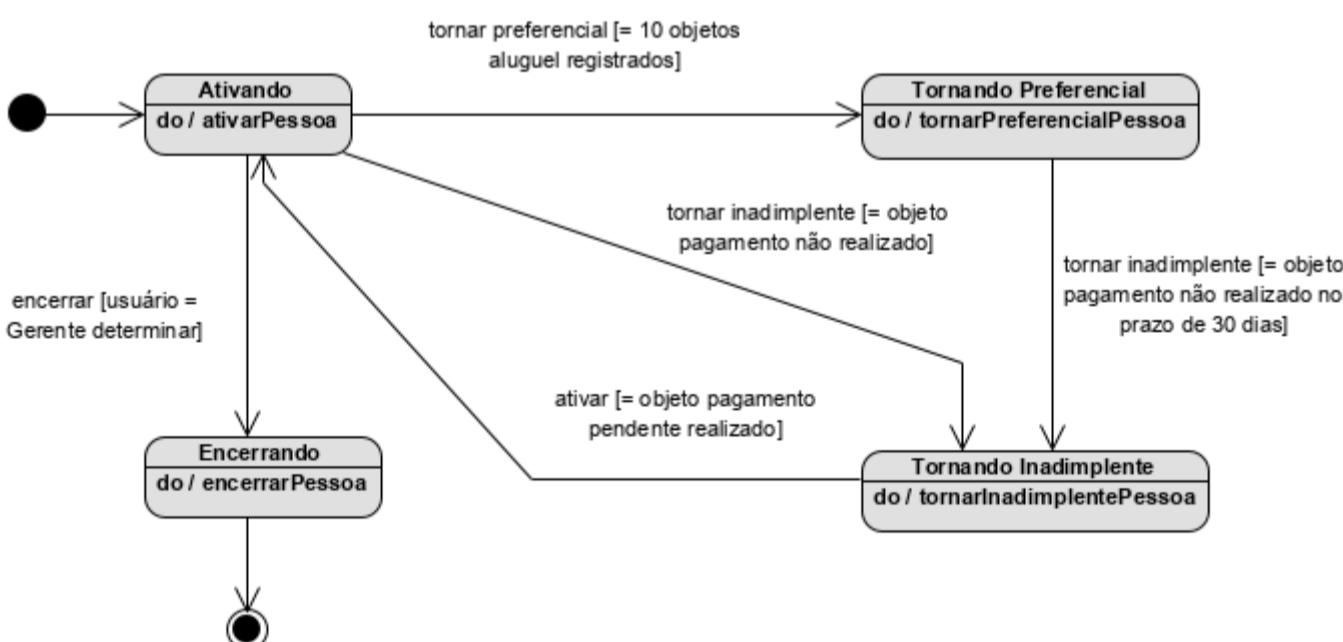
Para os objetos da classe “Pessoa”, que pode ser uma pessoa física no papel de cliente ou uma pessoa jurídica no papel de empresa, os estados definidos são: Ativa, Preferencial, Inadimplente e Encerrada. As regras definidas são as seguintes:

- Toda pessoa ao ser cadastrada deve assumir automaticamente a situação de Ativa.
- Toda pessoa Ativa deve assumir a situação de Preferencial a partir da realização do décimo aluguel.

- Toda pessoa Ativa pode assumir a situação de Inadimplente se o pagamento de um aluguel não for realizado.
- Toda pessoa Ativa pode assumir a situação de Encerrada se a gerência assim determinar por algum motivo pontual.
- Toda pessoa Preferencial pode assumir a situação de Inadimplente se o pagamento de um aluguel não for realizado no prazo de 30 dias.
- Toda Pessoa Inadimplente pode assumir a situação de Ativo, se o pagamento pendente for realizado.
- Toda pessoa Encerrada não deve assumir outro Estado.

O Diagrama de Máquina de Estados ilustrado na Figura 4.18 ilustra os estados definidos para os objetos da classe “Pessoa”, e suas transições de estados são indicadas com as condições de guarda, em conformidade com as regras de negócio indicadas.

Figura 4.18 | Diagrama de Máquina de Estados – Classe Pessoa



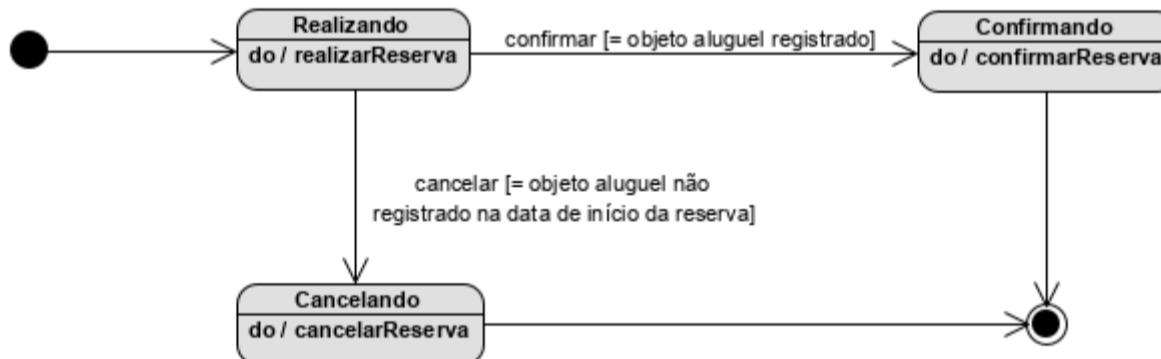
Fonte: elaborada pela autora.

Para os objetos da classe “Reserva”, os estados definidos são: Realizada, Confirmada e Cancelada. As regras definidas são:

- Toda reserva, ao ser cadastrada, deve assumir automaticamente a situação de Realizada.
- Toda reserva Realizada deve assumir a situação de Confirmada, a partir da realização do aluguel.
- Toda reserva Realizada pode assumir a situação de Cancelada, caso o cliente ou o gerente efetivar o cancelamento.
- Toda reserva Confirmada ou Cancelada não deve assumir outro Estado.

O Diagrama de Máquina de Estados ilustrado na Figura 4.19 ilustra os estados definidos para os objetos da classe “Reserva” e suas transições de estados são indicadas com as condições de guarda, considerando as regras de negócio apresentadas.

Figura 4.19 | Diagrama de Máquina de Estados – Classe Reserva



Fonte: elaborada pela autora.

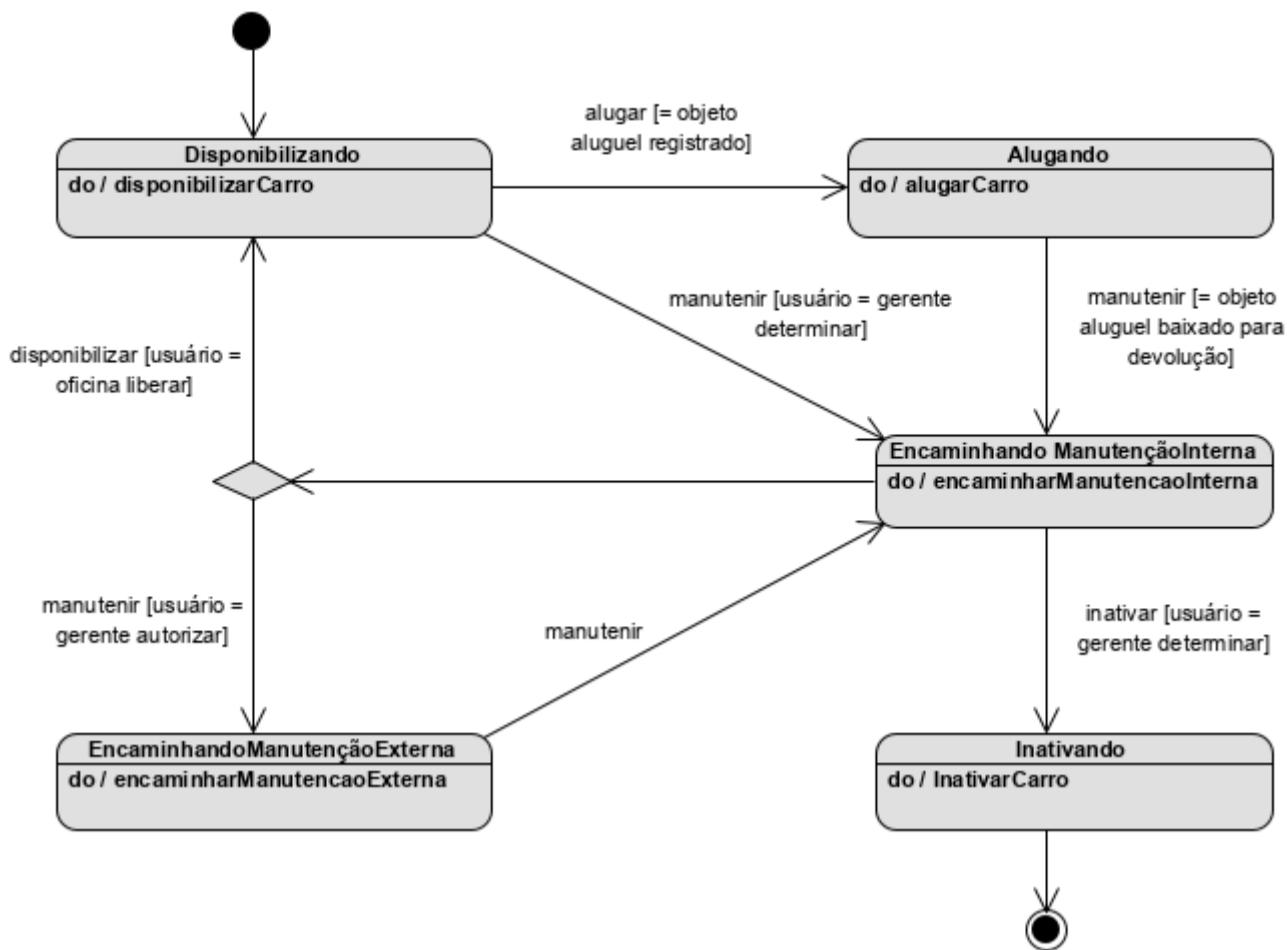
Para os objetos da classe “Carro”, os estados definidos são: Disponível, Alugado, Manutenção Interna, Manutenção Externa e Inativo. As regras definidas são:

- Todo carro ao ser cadastrado deve assumir, automaticamente, a situação de Disponível.
- Todo carro Disponível deve assumir a situação de Alugado quando for realizado um aluguel.

- Todo carro Alugado deve assumir a situação de Manutenção Interna quando for realizada a devolução do carro.
- Todo carro em Manutenção Interna deve assumir a situação de Disponível, se confirmada a vistoria técnica da oficina e a lavagem do carro.
- Todo carro em Manutenção Interna deve assumir a situação de Manutenção Externa, se o gerente assim determinar devido a algum problema técnico identificado que não possa ser resolvido pela oficina da locadora.
- Todo carro Disponível deve assumir a situação de Manutenção Interna, se o gerente assim determinar por algum motivo pontual.
- Todo carro em Manutenção Externa deve assumir a situação de Manutenção Interna para conferência geral do carro após a finalização do serviço realizado pela oficina externa.
- Todo carro em Manutenção Interna deve assumir a situação de Inativo se a gerência determinar.
- Todo carro Inativo não deve assumir outro Estado.

O Diagrama de Máquina de Estados ilustrado na Figura 4.20 ilustra os estados definidos para os objetos da classe “Carro”, e suas transições de estados são indicadas com as condições de guarda, considerando as regras de negócio apresentadas.

Figura 4.20 | Diagrama de Máquina de Estados – Classe Carro



Fonte: elaborada pela autora.

Para os objetos da classe “AluguelDevolução”, os estados definidos para o lançamento de um aluguel são: Realizado, Registrado e Cancelado. As regras definidas são as que seguem:

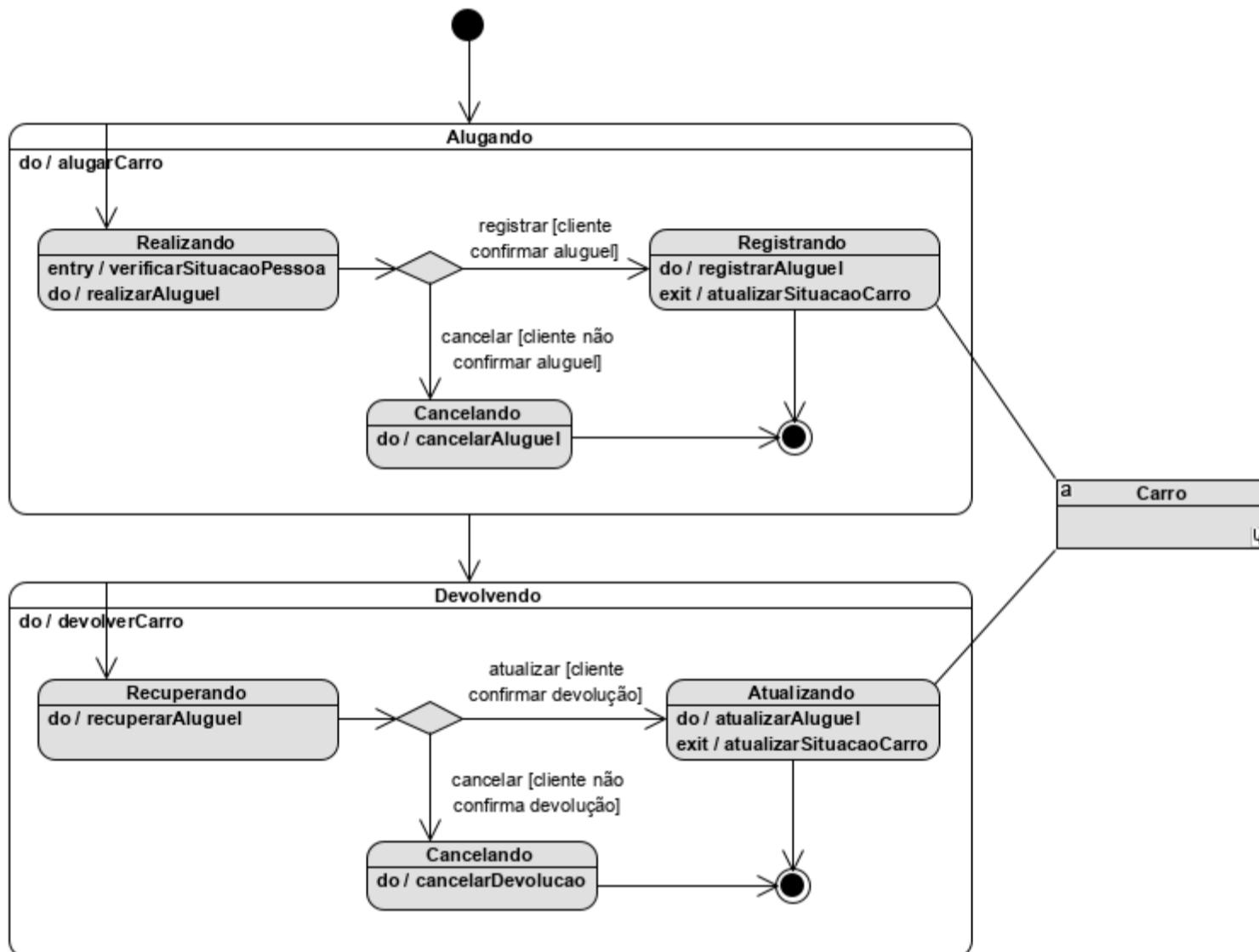
- Todo aluguel ao ser lançado deve assumir, automaticamente, a situação de Realizado.
- Todo aluguel Realizado deve assumir a situação de Registrado quando for confirmado o aluguel.
- Todo aluguel Realizado pode assumir a situação de Cancelado quando não for confirmado o aluguel por algum motivo.

Para os objetos da classe “AluguelDevolução”, os estados definidos para o lançamento da devolução do carro são: Recuperado, Atualizado e Cancelado. As regras definidas são:

- Toda devolução de um carro alugado ao ser lançada deve assumir, automaticamente, a situação de Recuperado.
- Uma vez ao Recuperado, deve assumir a situação de Atualizado quando for confirmada a devolução do carro.
- Uma vez ao Recuperado, deve assumir a situação de Cancelado quando não for confirmada a devolução do carro por algum motivo.

O Diagrama de Máquina de Estados ilustrado na Figura 4.21 ilustra os estados definidos para os objetos da classe “AluguelDevolucao”, e suas transições de estados são indicadas com as condições de guarda, considerando as regras de negócio apresentadas. O diagrama mostra um estado composto para a ocorrência do aluguel de um carro e outro estado composto para a ocorrência da devolução, pois ocorrem em período distinto, sendo a devolução uma atualização ou “baixa” do aluguel.

Figura 4.21 | Diagrama de Máquina de Estados – Classe AluguelDevolucao



Fonte: elaborada pela autora.

Observe no diagrama que em ambos os estados compostos foi indicada uma ação de estado do tipo saída (cláusula “Exit”), referenciando a operação *atualizarSituacaoCarro* nos subestados “Registrando” e “Atualizando” dos estados “Alugando” e “Dovolvendo”, respectivamente. A partir do registro de um aluguel, a situação do carro deve ser atualizada para “Alugado” e, posteriormente, ao efetivar a devolução do carro, a situação do carro deve ser atualizada para “Manutenção Interna”.

Considerando as particularidades de cada domínio de sistema, as regras de negócio aplicadas às transições de estados dos objetos são de total responsabilidade do analista de sistemas?

Posteriormente, na atividade de implementação do processo de desenvolvimento do sistema, a especificação do Diagrama de Máquina de Estados apoiará a implementação consistente das classes, em conformidade com as regras de negócio definidas para os estados dos objetos do sistema.

0

Ver anotações

| DIAGRAMA DE ATIVIDADES

Considerando que os casos de uso estão definidos, é importante evoluir com a modelagem comportamental do sistema para uma melhor compreensão da lógica de funcionamento de cada caso de uso.

A UML não estabelece qual técnica de modelagem comportamental ou de interação é a ideal para especificar cada caso de uso. Você, como analista de sistemas ou de acordo com a metodologia de desenvolvimento da empresa, deverá definir a melhor técnica de modelagem comportamental da UML a ser adotada, conforme as características ou aplicabilidade de cada caso de uso. Assim, adotamos a técnica de modelagem Diagrama de Atividades para especificar dois casos de uso, o “Reservar Carro” e o “Acessar Conta Cliente”.

Segundo Bezerra (2014, p. 307), o Diagrama de Atividades

“

[...] pode ser visto como uma extensão dos fluxogramas. Além de possuir toda a semântica existente em um fluxograma, o diagrama de atividade possui notação para representar ações concorrentes, juntamente com a sua sincronização.

ASSIMILE

O Diagrama de Atividades demonstra o fluxo de controle de um conjunto de atividades que representa a execução de caso de uso, processo de negócio, subsistema ou até mesmo

do sistema completo, ou seja, descreve os passos a serem percorridos para a realização de uma atividade específica.

Documentação do caso de uso - Reservar Carro:

Nome do caso de uso: Reservar Carro.

Descrição: Cliente solicita reservar um carro.

Autor principal: Cliente.

Cenário Principal:

1. Cliente solicita reserva de carro.
2. Cliente informa dados (loja/filial, data e hora) da retirada do veículo.
3. Sistema recupera lojas cadastradas para retirada.
4. Cliente escolha loja para retirada.
5. Cliente informa dados (loja/filial, data e hora) da devolução do veículo.
6. Sistema recupera lojas cadastradas para devolução.
7. Cliente seleciona loja para devolução.
8. Sistema recupera grupos de carros disponíveis no período indicado.
9. Cliente seleciona o grupo de carro desejado.
10. Sistema recupera itens opcionais do carro.
11. Cliente informa itens opcionais do carro desejado.
12. Cliente acessa sua conta pessoal.
13. Sistema recupera dados do cliente.
14. Sistema calcula valor total da reserva.
15. Cliente confirma reserva.

16. Sistema registra reserva.

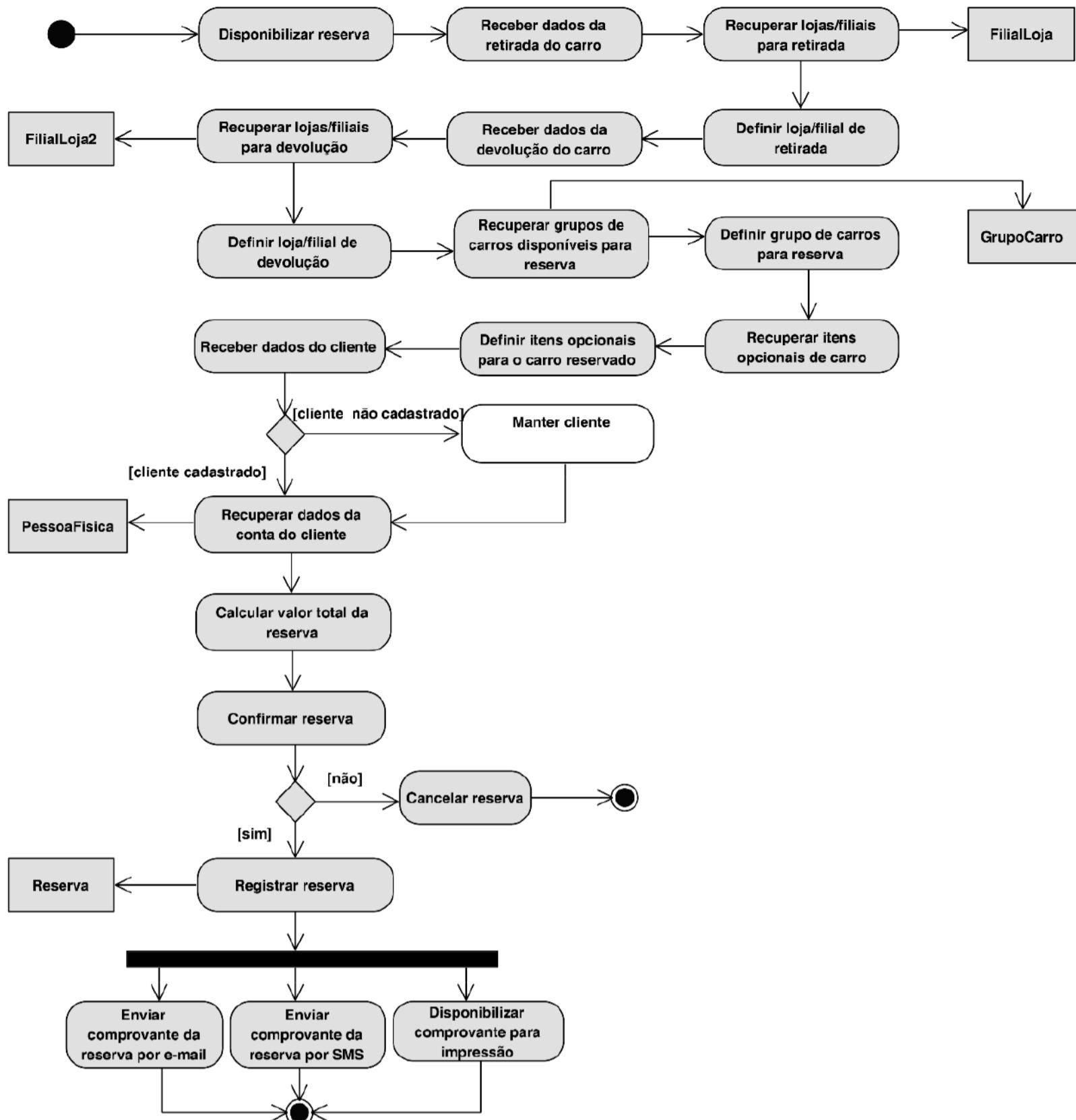
17. Sistema disponibiliza opção de imprimir comprovante da reserva.

18. Sistema envia por e-mail comprovante da reserva.

19. Sistema envia SMS comprovante da reserva.

A partir da descrição do cenário principal do caso de uso “Reservar Carro”, a Figura 4.22 ilustra o Diagrama de Atividades correspondente, no formato de fluxos de controle sequencial. Lembre-se que conforme orientação da UML, a nomenclatura dos elementos básicos do diagrama – Atividade ou Nό de Ação deve-se iniciar com verbo no infinitivo e sempre na perspectiva de execução do sistema, ou seja, se o cliente informou os dados, na perspectiva do sistema, o sistema recebe ou aceita os dados. Ainda, uma Atividade representa a sequência de tarefas em um fluxo de trabalho que resulta no comportamento de um processo, sendo que uma atividade é composta por um conjunto de ações.

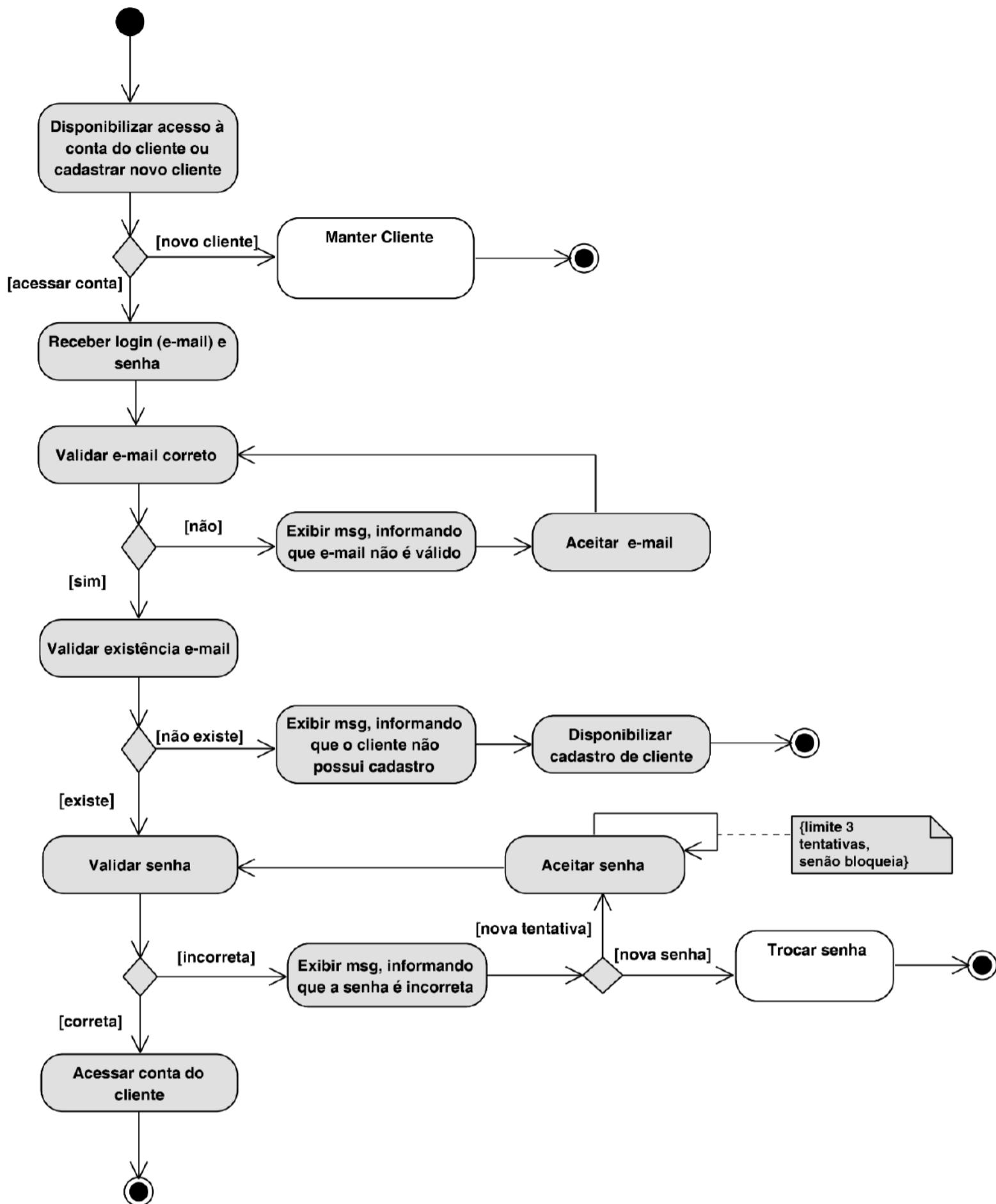
Observe na Figura 4.22 que alguns nós de ação, como o “Recuperar lojas/filiais para retirada” acessa o nó de objeto “FilialLoja”, o “Recuperar grupos de carros disponíveis para reserva” acessa o nó de objeto “GrupoCarro” e assim por diante. No diagrama, todos os elementos representados por um retângulo pequeno com bordas arredondadas e um nome centralizado, na cor cinza, são nós de ação, com exceção do elemento “Manter cliente”, na cor branca, que representa uma atividade correspondente ao caso de uso para cadastrar um cliente novo. Por convenção da ferramenta CASE Visual Paradigm, o nome do elemento Atividade também é centralizado, contudo fica disposto na parte superior do retângulo com bordas arredondadas.



Fonte: captura de tela do software Visual Paradigm Community Edition elaborada pela autora.

A Figura 4.23 ilustra o Diagrama de Atividades correspondente ao caso de uso “Acessar Conta Cliente”, também no formato de fluxos de controle sequencial. Observe que no diagrama constam a chamada a duas atividades – “Manter Cliente”, a partir da condição de cadastrar novo cliente, e “Trocá Senha”, a partir da condição de cadastrar nova senha. No nó de ação “Aceitar Senha” foi indicado uma restrição, representada no elemento nota, indicando o limite de três tentativas para aceitar uma nova senha.

Figura 4.23 | Diagrama de Atividades – Acessar Conta Cliente



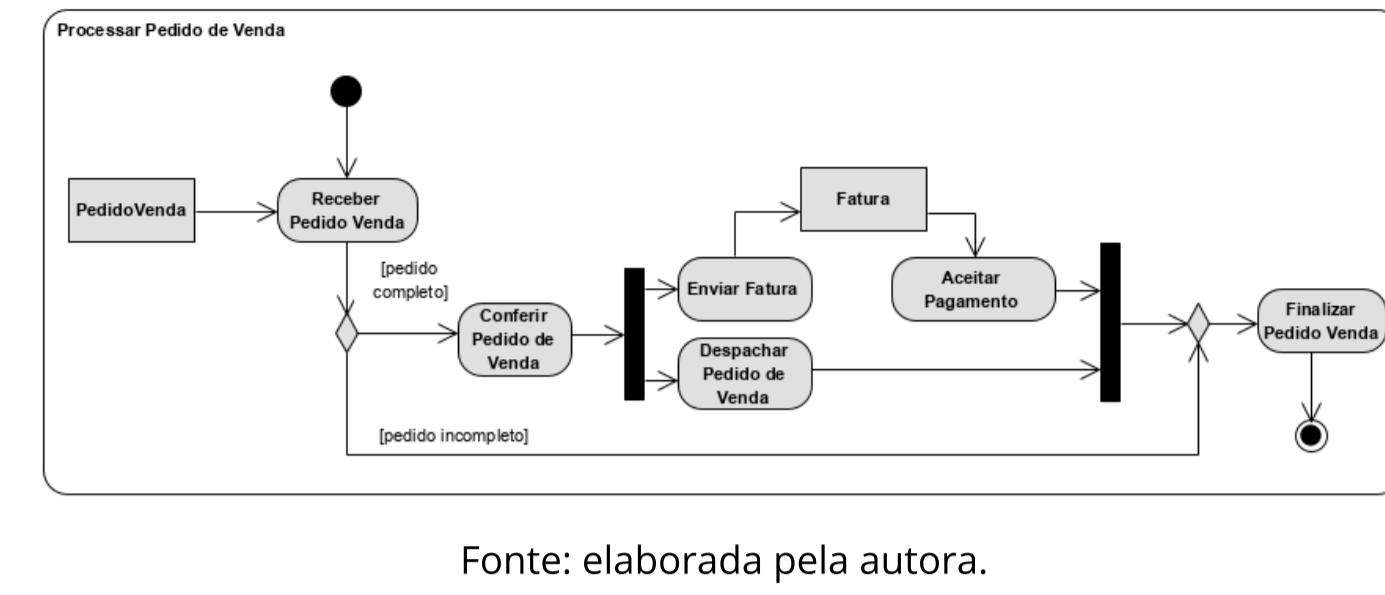
Fonte: captura de tela do software Visual Paradigm Community Edition elaborada pela autora.

EXEMPLIFICANDO

A Figura 4.24 ilustra um exemplo de Diagrama de Atividades correspondente à atividade “Processar Pedido de Venda”, composta pelo fluxo de ações que apresentam o processo de um pedido de venda, específico para um domínio de vendas. O diagrama contempla a representação dos principais elementos de um Diagrama de Atividades.

A atividade inicia-se com a representação do **nó de ação** nomeado de *Receber Pedido Venda*, sinalizada com o **nó inicial**, que acessa o **nó de objeto** *PedidoVenda*, a partir de um **fluxo de objeto**. Na sequência, o fluxo de controle segue para um **nó de decisão** com duas decisões, cada uma indicada por uma condição de guarda, sendo que na condição “pedido completo”, o fluxo segue para um **nó de bifurcação** com uma entrada e dois fluxos de saída concorrentes, seguindo para um **nó de união** com dois fluxos de entrada e um único fluxo de saída direcionado para um nó decisão, que finaliza a atividade com a ação *Finalizar Pedido Venda*, sinalizada com o **nó final**.

Figura 4.24 | Diagrama de Atividades – Processar Pedido de Venda



Fonte: elaborada pela autora.

Para finalizar a modelagem comportamental dos principais casos de uso, na sequência apresentamos os Diagramas de Sequência correspondentes aos casos de uso “Alugar Carro” e “Devolver Carro”.

| DIAGRAMA DE SEQUÊNCIA

Na modelagem da atividade de análise, recomenda-se utilizar o Diagrama de Sequência para descrever a realização dos casos de uso, representando os objetos que colaboram entre si a partir da troca de mensagens entre eles.

Segundo Bezerra (2014, p. 193), “[...] o objetivo do diagrama de sequência é apresentar as interações entre objetos na ordem temporal em que elas acontecem”.

Os principais elementos que compõem o Diagrama de Sequência são: Linha de Vida, Ator, Objeto, Foco de Controle e as Mensagens do tipo – síncrona, assíncrona, de retorno e de autochamada.

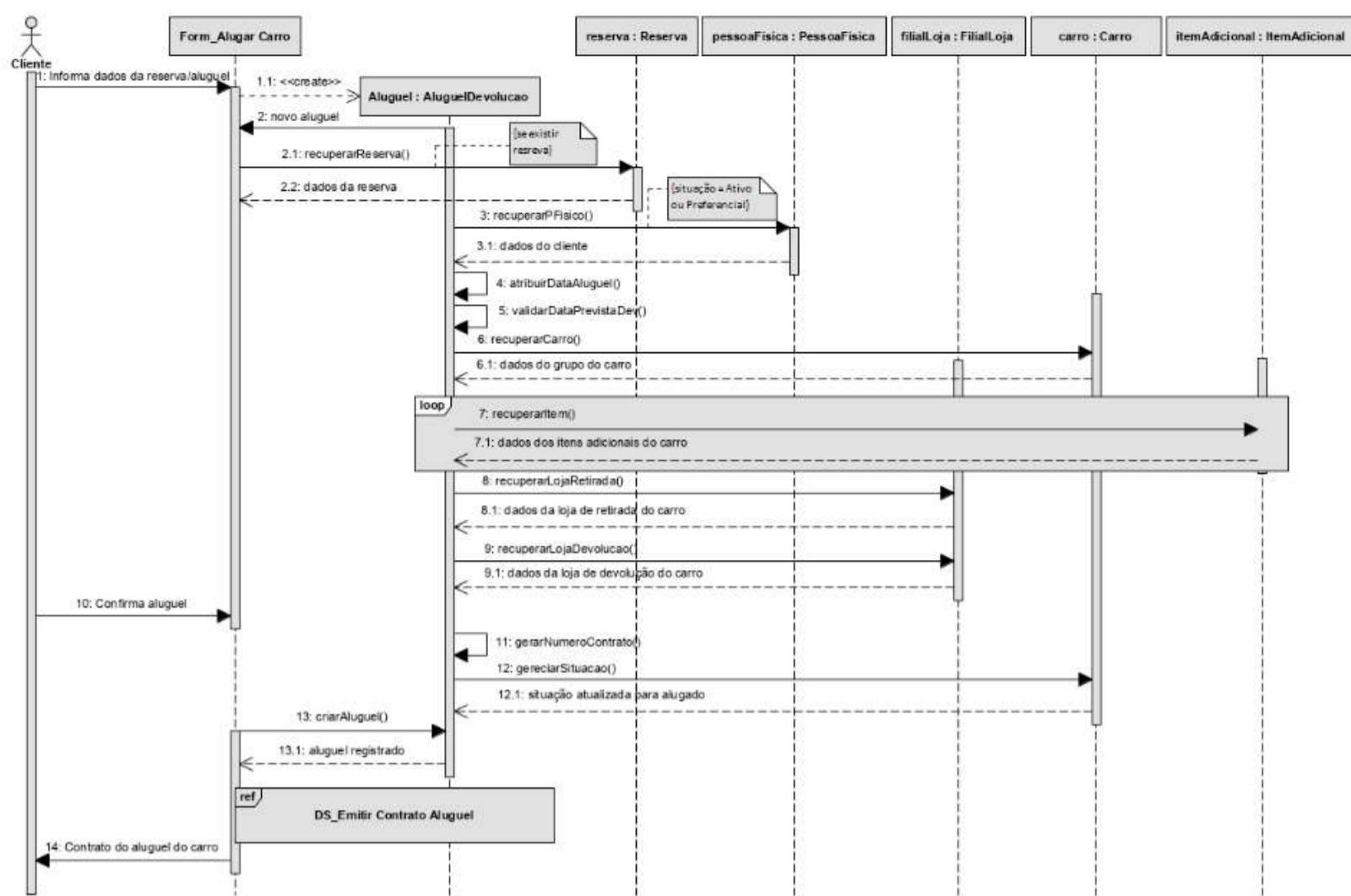
A Figura 4.25 ilustra o Diagrama de Sequência correspondente ao caso de uso “Alugar Carro”. O diagrama ilustra a interação entre os elementos ator Cliente; a linha de vida “Form_Alugar Carro” que representa a interface gráfica correspondente ao caso de uso, sendo que a partir dela o ator interage com o sistema; e as linhas de vida “Aluguel:AluguelDevolucao”, “reserva:Reserva”, “pessoaFisica:PessoaFisica”, “filialLoja:FilialLoja”,

“carro:Carro” e “itemAdicional:ItemAdicional”, que representam os objetos que trocam as mensagens durante a realização do caso de uso. As mensagens 7 e 7.1 fazem parte de um fragmento combinado do tipo *loop*, indicando que vários itens adicionais podem ser selecionados para o carro alugado, enquanto o cliente indicar. Finalizando o processo do aluguel, ao registrar o aluguel do carro é indicado um fragmento de interação que é disparado para chamar o caso de uso de “Emitir Contrato de Aluguel”.

Lembre-se de que a representação dos fragmentos combinados ou dos fragmentos de interação possibilita o alinhamento de interações, sendo que cada fragmento representa uma interação independente e o mesmo fragmento de interação pode ser utilizado em vários Diagramas de Sequência.

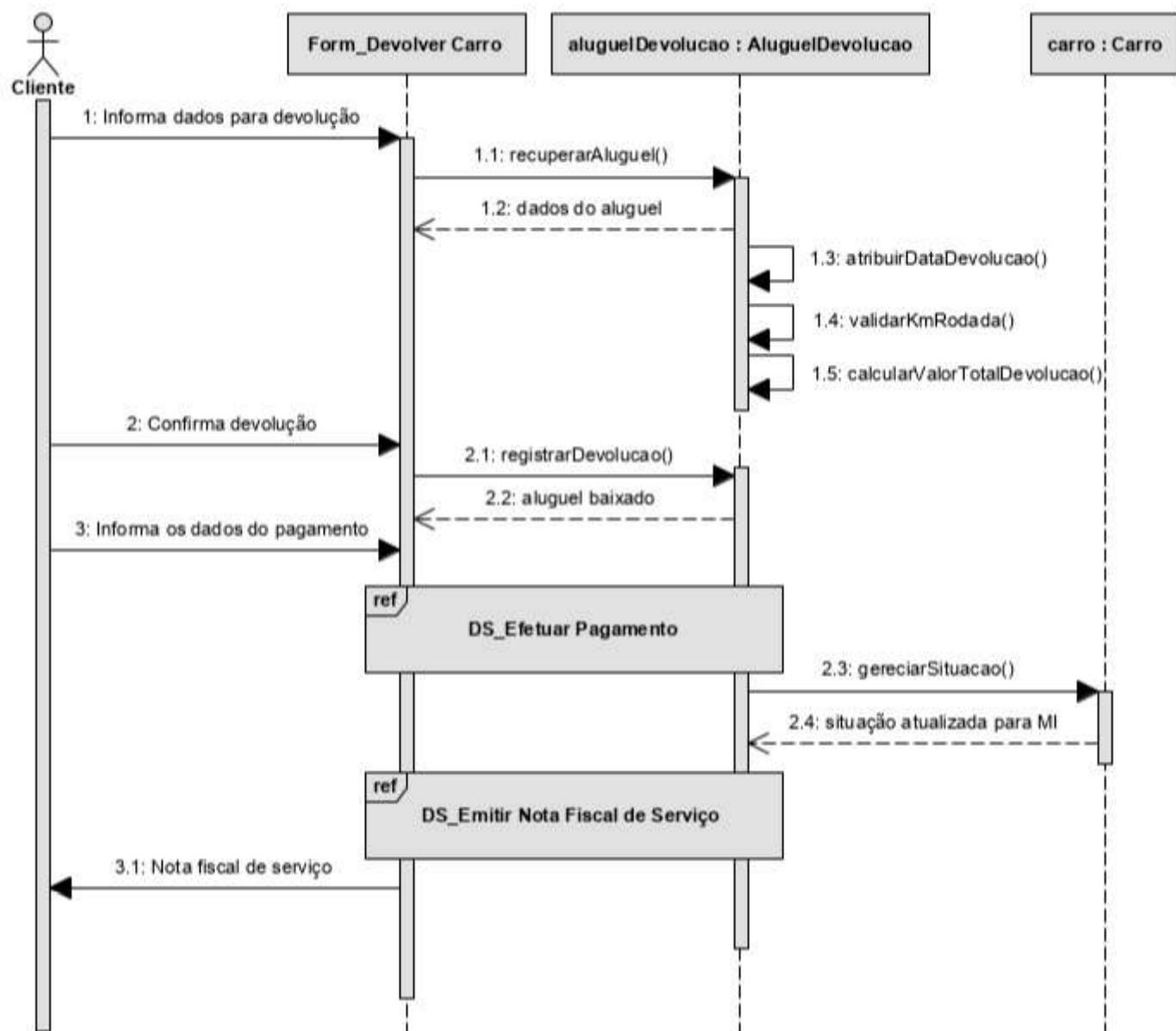
A Figura 4.26 ilustra o Diagrama de Sequência correspondente ao caso de uso “Devolver Carro”. O diagrama ilustra a interação entre os elementos ator Cliente; a linha de vida “Form_Devolver Carro” que representa a interface gráfica correspondente ao caso de uso, sendo que a partir dela o ator interage com o sistema; e as linhas de vida “Aluguel:AluguelDevolucao” e “carro:Carro”, que representam os objetos que trocam as mensagens durante a realização do caso de uso. Após o cliente informar os dados do pagamento do aluguel, é indicado o fragmento de interação que é disparado para chamar o caso de uso “Efetuar Pagamento”. Por fim, outro fragmento de interação que chama o caso de uso é “Emitir Nota Fiscal de Serviço”.

Assim como as demais técnicas de modelagem da UML que podem ser apresentadas em diferentes níveis de detalhamento, o mesmo Diagrama de Sequência pode ser refinado em conformidade com padrões de implementação e compor a modelagem da atividade de projeto.



Fonte: captura de tela do software Visual Paradigm Community Edition elaborada pela autora.

Figura 4.26 | Diagrama de Comunicação – Devolver Carro



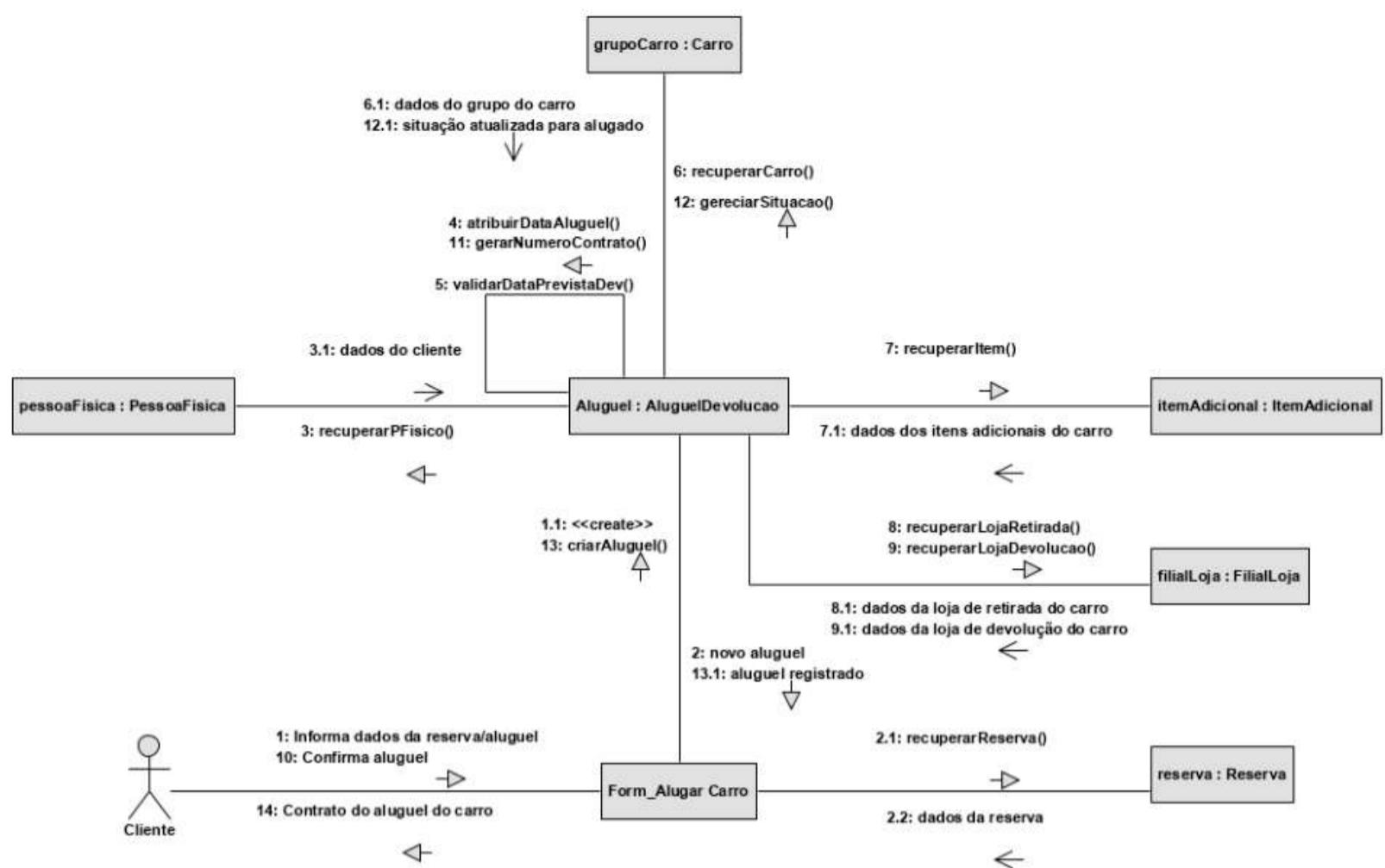
Fonte: captura de tela do software Visual Paradigm Community Edition elaborada pela autora.

REFLITA

Das diferentes técnicas de modelagem comportamentais da UML, o Diagrama de Sequência é a técnica recomendada para especificar as interações entre os objetos que realizam os casos de uso. Assim, obrigatoriamente é necessário elaborar um Diagrama de Sequência para cada caso de uso?

Por fim, a partir do Diagrama de Sequência do caso de uso “Alugar Carro” gerou-se, automaticamente, na ferramenta CASE Visual Paradigm, o Diagrama de Comunicação correspondente ao caso de uso “Alugar Carro”. Observe que o diagrama representado na Figura 4.27 enfatiza exatamente a ligação por vínculos entre os objetos representados pela *Lifeline*, que participam da realização do caso de uso.

Figura 4.27 | Diagrama de Comunicação – Alugar Carro



Fonte: captura de tela do software Visual Paradigm Community Edition elaborada pela autora.

O Diagrama de Comunicação não demonstra a temporalidade da realização de um processo, no entanto, representa o relacionamento entre os objetos envolvidos na realização do caso de uso, agrupando as mensagens entre os objetos que participam de uma interação por vínculos estabelecidos entre os objetos.

REFERÊNCIAS

BEZERRA, E. **Princípios de análise e projeto de sistemas com UML**. 3. ed. Rio de Janeiro: Elsevier, 2014.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **UML: guia do usuário**. 2. ed. Rio de Janeiro: Campus, 2006.

FELDERER, M.; HERRMANN, A. Comprehensibility of system models during test design: a controlled experiment comparing UML activity diagrams and state machines. **Software Quality Journal**, [s. l.], v. 27, n. 1, p. 125–147, 2019. DOI 10.1007/s11219-018-9407-9. Disponível em: Biblioteca Virtual – EBSCO HOST. <https://bit.ly/3bBE3BI>. Acesso em: 17 jun. 2020.

GUEDES, G. T. A. **UML**: uma abordagem prática. 3. ed. São Paulo: Novatec, 2018.

MEDEIROS, E. S. **Desenvolvendo software com UML 2.0**: definitivo. São Paulo: Pearson, 2004.

PRESSMAN, R.; MAXIM, B. **Engenharia de software**: uma abordagem profissional. 8. ed. Porto Alegre: AMGH, 2018.

FOCO NO MERCADO DE TRABALHO

MODELAGEM COMPLEMENTAR DA ATIVIDADE DE ANÁLISE

Iolanda Cláudia Sanches Catarino

ELABORAÇÃO DOS DIAGRAMAS DE MÁQUINA DE ESTADOS E DE SEQUÊNCIA

Identificação das regras de negócio aplicadas ao contexto dos objetos com estados relevantes, definindo consistentemente os estados e suas transições de estados, e apresentação das interações entre os objetos na ordem temporal em que elas acontecem.



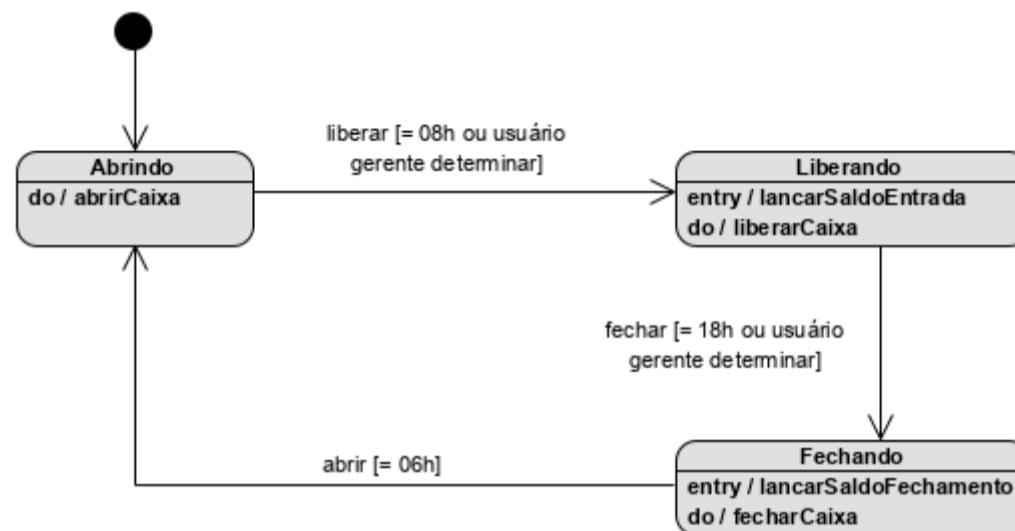
Fonte: Shutterstock.

[Deseja ouvir este material?](#)

SEM MEDO DE ERRAR

Considerando as regras definidas para as transições entre os estados dos objetos da classe “Caixa”, segue o Diagrama de Máquina de Estados, como proposta de uma solução.

Figura 4.28 | Diagrama de Máquina de Estado – Classe Caixa



Fonte: elaborada pela autora.

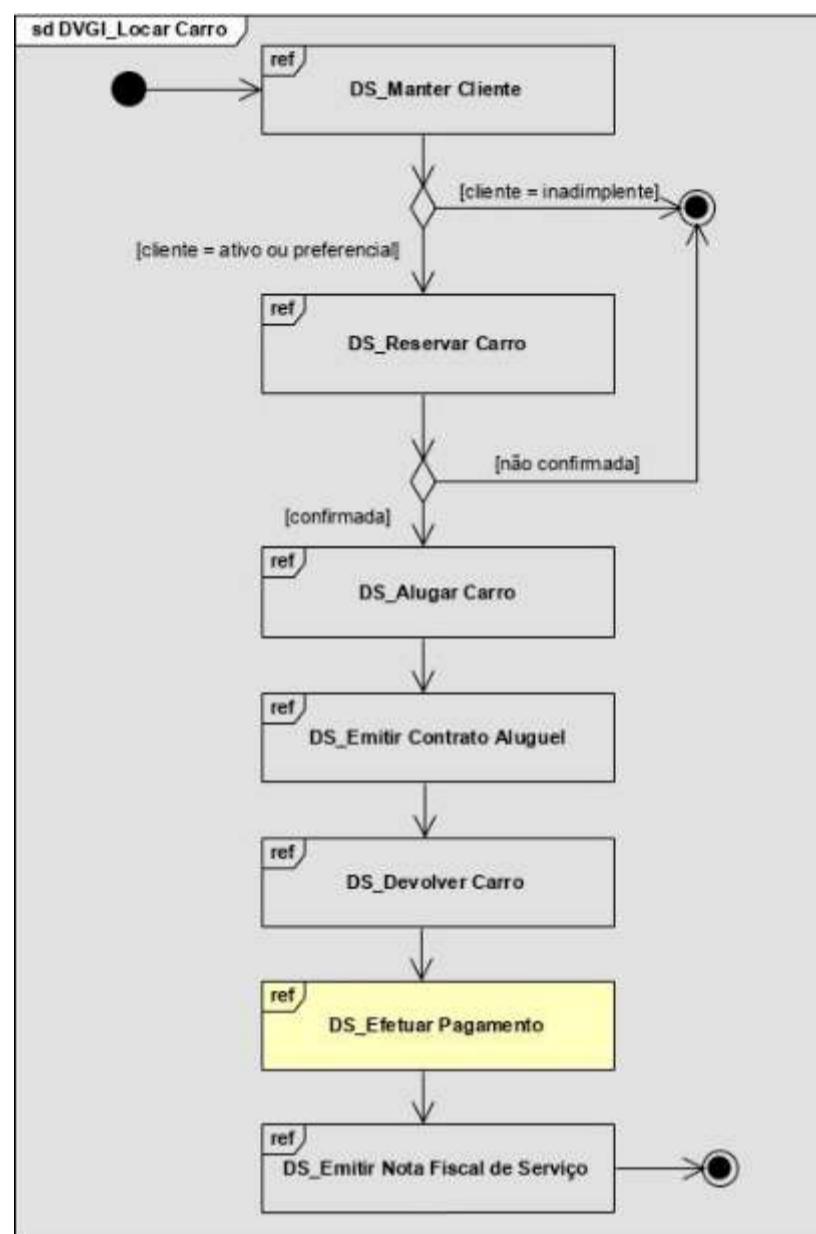
A Figura 4.29 apresenta o Diagrama de Visão Geral de Interação correspondente ao processo de locar um carro, integrando todos os casos de uso envolvidos nesse processo, que são: cadastramento de um cliente, reserva de um carro, aluguel de um carro e devolução do carro.

O Diagrama de Visão Geral de Interação é uma variação do Diagrama de Atividades que integra diferentes tipos de diagramas de interação, demonstrando um processo geral. No diagrama são utilizados dois tipos de quadros:

- **Quadros de Interação:** que contêm a representação completa de qualquer tipo de diagrama de interação.
- **Quadros de Ocorrência de Interação:** que fazem uma referência a um diagrama de interação, no entanto, não apresentam seu detalhamento.

Cada quadro de ocorrência de interação representado no diagrama da Figura 4.29 apresenta os Diagramas de Sequência correspondentes aos casos de uso que envolvem todo o processo de locação de um carro.

Figura 4.29 | Diagrama de Visão Geral de Interação – Locar Carro



Fonte: captura de tela do software Visual Paradigm Community Edition elaborada pela autora.

NÃO PODE FALTAR

TRANSIÇÃO DA ATIVIDADE DE ANÁLISE PARA PROJETO

Iolanda Cláudia Sanches Catarino

Ver anotações

REFINAMENTO DOS ASPECTOS ESTÁTICOS E ESTRUTURAIS DO MODELO DE CLASSES

Complementação das classes com os tipos de dados dos atributos com as demais operações identificadas na especificação dos diagramas comportamentais e de interação, além da revisão dos relacionamentos das classes, apresentando, assim, uma nova versão do Modelo de Classes.



Fonte: Shutterstock.

Deseja ouvir este material?

Áudio disponível no material digital.

Caro aluno, seja bem-vindo à seção sobre a transição da atividade de análise para o projeto!

Nesta seção, apresentamos a transição da análise ao projeto no desenvolvimento de sistemas orientado a objetos, considerando a adoção da Linguagem de Modelagem Unificada (UML – *Unified Modeling Language*) e considerando os princípios do modelo de processo de software Processo Unificado (PU – *Unified Process*) que surgiu para apoiar a UML, enfatizando o desenvolvimento dirigido a casos de uso, de forma iterativa e incremental, e fornecendo uma maneira sistemática e evolutiva de modelar os sistemas sob diferentes aspectos de análise e detalhamento.

Lembre-se de que a UML não faz grande distinção entre a modelagem das atividades de análise e projeto de um processo de desenvolvimento de software, sendo que a atividade de projeto é uma extensão refinada dos diagramas elaborados na análise com detalhes aplicados à implementação, e a análise e o projeto podem estar acontecendo simultaneamente. Essa definição de trabalhar com a modelagem simultânea ou até mesmo de apresentar uma única modelagem como sendo das atividades de análise e projeto com diferentes perspectivas de visão, é uma questão definida na metodologia de desenvolvimento do sistema, conforme cada equipe de desenvolvedores ou empresa prefere adotar.

Nas unidades anteriores foram abordadas todas as características das técnicas de modelagem da UML. Além disso, você compreendeu que a UML privilegia a descrição de um sistema em três perspectivas principais de visões, que são a estrutural, que enfatiza a visão estática do sistema; a funcional, que representa as funcionalidades do sistema; e a dinâmica, que representa a visão do comportamento do sistema em tempo de execução. Assim, de todas as técnicas de modelagem propostas pela UML – estruturais, comportamentais e de interação –, o importante é saber definir qual ou quais são as técnicas ideais a serem adotadas na modelagem de análise e de projeto, conforme o tipo e domínio do sistema.

Nesta seção você vai compreender os pontos essenciais do detalhamento dos aspectos estruturais do Diagrama de Classes, o qual pode ser apresentado como o Diagrama de Classes da atividade de projeto, como define-se o tipo de dados de cada atributo, listando todas as operações identificadas na modelagem dos diagramas comportamentais e de interação e fazendo a revisão dos relacionamentos.

Na modelagem de projeto, recomenda-se também apresentar uma nova versão dos diagramas estruturais e comportamentais que seja possível complementá-los com detalhes de padrões de projeto – *design*

patterns, componentes de softwares, *frameworks* e das tecnologias que serão adotadas para implementação do sistema, assim definindo o projeto da arquitetura do sistema.

Outro aspecto importante a considerar na modelagem de um sistema orientado a objetos, na atividade de projeto, é a forma pela qual os objetos especificados na representação de classes serão persistidos, e uma das principais estratégias adotadas é o mapeamento de classes para um sistema de gerência de banco de dados relacional. O projeto de algoritmos para implementar o sistema e o projeto de interface gráfica dos casos de uso que implicam a necessidade de interfaces, para darem o suporte de interação com os atores e usuários do sistema, também são atividades da modelagem de projeto.

Agora considere que você é o projetista de banco de dados, integrante da equipe de desenvolvedores responsável pelo sistema “Locação de Veículos”, e como parte da documentação referente a modelagem de dados você precisa apresentar o mapeamento das classes para tabelas do Banco de Dados Relacional (BDR) correspondente ao Modelo de Classes, composto de:

- Diagrama de Classes – md_Locacao_dc, ilustrado na Figura 4.31.
- Diagrama de Classes – md_Pagamento_dc, ilustrado na Figura 4.8 da seção 4.1.

Para especificação das tabelas, adote a notação “Nome da Tabela (coluna 1, coluna 2, coluna 3, coluna 4,... coluna n)”, representando, assim, o esquema completo do banco de dados.

Bom trabalho e ótimo estudo!

CONCEITO-CHAVE

Dependendo do domínio e da complexidade dos sistemas de software, é importante fornecer várias visões da modelagem orientada a objetos sob diferentes aspectos de análise e detalhamento, consistindo em uma única modelagem de análise e projeto ou apresentando modelagens

distintas das atividades de análise e projeto. Conforme o modelo de processo de desenvolvimento de software Processo Unificado, no qual preconiza-se a iteração, a análise e projeto integram-se como uma atividade conjunta executada ao longo das fases do processo – Concepção, Elaboração, Construção e Transição. Contudo, adotando a UML para modelagem de sistemas orientados a objetos, não há uma regra de qual ou quais técnicas de modelagem estruturais ou comportamentais devem ser adotadas e em que momento devem ser aplicadas. Essa decisão e definição é de responsabilidade do analista de sistemas, em conformidade com a metodologia da empresa de desenvolvimento.

O importante é ter claro que a UML oferece uma forma sistemática e evolutiva de modelar os sistemas orientados a objetos em várias perspectivas de visão estática e dinâmica, com base nas suas técnicas de modelagem estruturais, comportamentais e de interação.

Na atividade de análise evolui-se a modelagem especificada na atividade de análise de requisitos, concentrando-se na solução do problema, a partir da abstração, identificação, definição e documentação do que o sistema deve fazer, considerando a visão lógica do negócio, independentemente das tecnologias que serão adotadas para implementação do sistema. Posteriormente, a análise especificada vai se transformando em projeto, com novos detalhes e refinamentos que definem os aspectos físicos do sistema e aplicadas as tecnologias que serão adotadas na implementação do sistema.

Segundo Bezerra (2014), no desenvolvimento de sistemas orientados a objetos, as mesmas técnicas de modelagem são utilizadas durante a análise e o projeto, caracterizando uma uniformidade na modelagem do sistema durante o desenvolvimento. No entanto, essa uniformidade de representação tem a desvantagem de tornar bem menos nítida a separação entre o que é especificado na análise e o que é feito no

projeto. Dessa forma, pode-se dizer que a modelagem da análise vai se transformando na modelagem de projeto à medida que o desenvolvimento do sistema evolui.

MODELAGEM DE PROJETO

Como refinamento dos aspectos estáticos e estruturais das técnicas de modelagem da UML, para a atividade de projeto, o foco concentra-se na principal técnica de modelagem estrutural, o Diagrama de Classes.

Assim, é recomendado:

1. Refinar as classes, definindo as classes de projeto ou novas classes, ou seja, uma classe de análise pode resultar em mais de uma classe de projeto.
2. Definir o estereótipo das classes, que são classes de fronteira (*<<boundary>>*), de controle (*<<control>>*) ou de entidade (*<<entity>>*).
3. Estabelecer o tipo de dados de cada atributo, correspondente à linguagem de programação que será adotada na implementação.
4. Detalhar as operações, listando todas as operações identificadas nos diagramas comportamentais e de interação.
5. Revisar a visibilidade das classes e operações, definindo o nível de acessibilidade de um atributo ou operação por outros objetos, sendo a visibilidade do tipo privada, pública, protegida ou de pacote.
6. Rever os tipos de relacionamentos estabelecidos entre as classes, que são do tipo associação (podendo ser associação do tipo reflexiva, binária, ternária, classe associativa e agregação), generalização, dependência ou realização; além de indicar a navegabilidade de cada associação.
7. Definir as classes abstratas, as interfaces, padrões de projeto (*design patterns*), componentes de softwares reutilizáveis, *frameworks* e demais detalhes pertinentes às tecnologias de desenvolvimento a serem utilizadas durante a implementação, definindo, assim, a arquitetura de um sistema orientado a objetos.

Na representação dos atributos e operações de uma classe, indica-se também, à esquerda do nome desses atributos, o símbolo da visibilidade que determina o nível de acessibilidade de um atributo ou operação por outros objetos, do tipo: **privada** – representada por um símbolo de menos (-) e indica que somente os objetos da própria classe poderão enxergá-la; **pública** – representada por um símbolo de mais (+) e indica que qualquer objeto pode utilizar o objeto acessado; **protegida** – representada pelo símbolo de sustenido (#) e indica que além dos objetos da própria classe, os objetos das subclasses também podem ter acesso ao objeto da superclasse; e de **pacote** – representada pelo símbolo de til (~) e indica que o atributo ou operação é visível por qualquer objeto do mesmo pacote.

MODELAGEM DE CLASSE DE PROJETO

As boas práticas da engenharia de software enfatizam a redução dos esforços e custos da produção, com a reutilização de software que pode ser obtida com a utilização de recursos como os componentes de softwares e os usuais *frameworks* para o desenvolvimento web e para o desenvolvimento de aplicações móveis, aplicados às diferentes tecnologias de desenvolvimento de software. Assim, uma vez adotados esses recursos que agilizam o desenvolvimento de software, eles devem ser integrados à modelagem de projeto, estabelecendo a arquitetura do software.

Na representação do Diagrama de Classes da atividade de projeto, é importante revisar o relacionamento estabelecido entre as classes de objetos. Os relacionamentos usuais estabelecidos no Diagrama de Classes de análise são do tipo associação e generalização, no entanto, na versão do diagrama de projeto é comum inserir componentes de softwares, bem como aplicar *design patterns*. Assim, podem surgir os relacionamentos do tipo realização e dependência. Vamos, então, revisar os conceitos que se referem a classes:

- **Realização:** relacionamento que modela a conexão existente entre uma interface e uma classe ou componente, no qual um dos elementos especifica um contrato de uso com o outro elemento. A representação gráfica do relacionamento de realização é indicada por uma linha tracejada com uma grande seta vazia, apontando para o elemento que especifica o contrato.
- **Dependência:** relacionamento de utilização entre classes, indicando que uma alteração na especificação de um elemento pode afetar outro elemento que a utilize. A representação gráfica do relacionamento de dependência é indicada por uma linha tracejada, apontando o item de que o outro depende.

Um **padrão de projeto** (*design pattern*) descreve uma estrutura lógica comum para criar um projeto orientado a objetos reutilizável, baseado em soluções práticas com mecanismos de cooperação entre componentes que são definidos para encontrar soluções para os problemas de projeto em um contexto específico. Um **framework** consiste em um conjunto de classes abstratas relacionadas, e por um modelo de extensibilidade das suas classes para as aplicações a serem construídas, proporcionando funcionalidades pré-fabricadas para agilizar o desenvolvimento do software. Um **componente de software** é uma parte física e substituível de um software, aplicado a um domínio, ao qual se adapta e fornece a realização de serviços através de uma interface específica.

Ainda faz parte da definição das classes de projeto definir o estereótipo das classes. Uma classe indicada com um estereótipo representa uma classificação do elemento. Alguns tipos de estereótipos de classe adotados no Diagrama de Classes de projeto são:

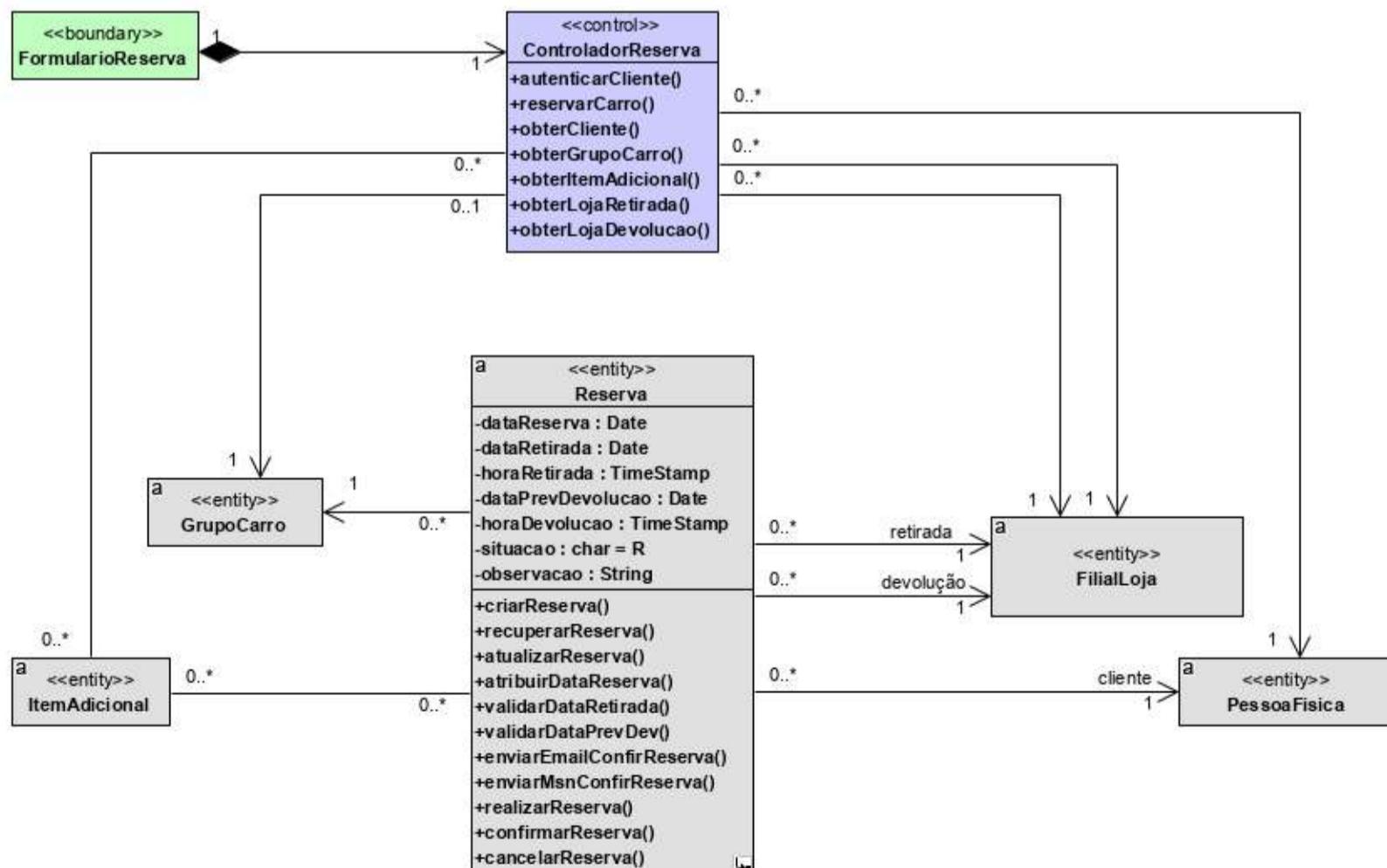
- **De fronteira (***<<boundary>>***):** identifica uma classe que serve de comunicação entre os atores externos e o sistema. Muitas vezes uma classe de fronteira é associada à própria interface do sistema. A utilização de classes de fronteira é importante quando é preciso definir a existência de uma interface para o sistema, sendo desnecessária em sistemas muito simples cujas interfaces não apresentam nenhuma característica especial.
- **De controle (***<<control>>***):** identifica classes que servem de intermédio entre as classes de fronteira e as outras classes do sistema. Os objetos de controle são responsáveis por interpretar os eventos ocorridos sobre os objetos de fronteira, como os movimentos do mouse ou o pressionamento de um botão, e

retransmiti-lo para os objetos das classes de entidade que compõem o sistema.

- **De entidade (<<entity>>):** classes de entidade também são chamadas de classes do negócio, e são aquelas que representam os conceitos do domínio do sistema, ou seja, a classe contém informações recebidas ou geradas por meio do sistema. É com base nas classes de entidade que se define quais delas geram objetos que devem ser persistentes, no qual o mecanismo de armazenamento geralmente é um sistema de gerenciamento de banco de dados.

A Figura 4.30 ilustra um recorte do Diagrama de Classes de projeto do sistema “Locação de Veículos” referente à classe “Reserva”, correspondente à visão de classes participantes do caso de uso “Reservar Carro”, apresentando uma parte do refinamento da classe com os aspectos sugeridos anteriormente. Observe que, por simplificação, apenas o refinamento correspondente a uma classe foi exemplificado. Em uma situação real, todas as classes de análise devem ser consideradas.

Figura 4.30 | Recorte do Diagrama de Classes (Projeto) – md_Locacao_dc

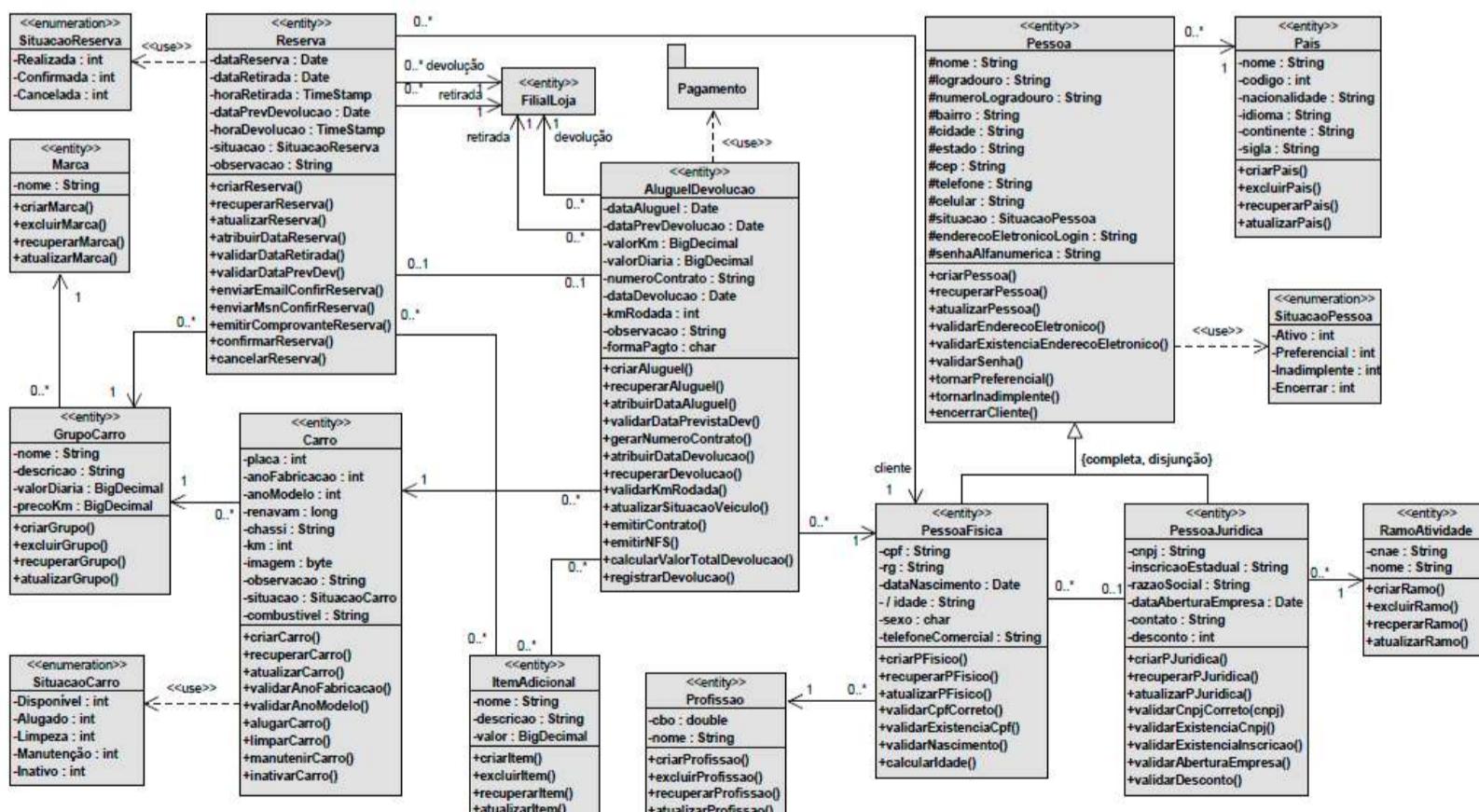


Fonte: elaborada pela autora.

O Diagrama de Classes da atividade de projeto complementa os elementos do diagrama, permitindo enriquecer os detalhes físicos desse diagrama, obtendo um modelo suficientemente completo e aproximando-o com o que será implementado.

A Figura 4.31 ilustra uma visão do Diagrama de Classes de projeto do pacote “md_Locacao_dc”, refinado apenas com a definição do tipo de dados dos atributos correspondentes à linguagem de programação Java; a indicação do estereótipo das classes do tipo entidade (<<entity>>); a representação das classes definidas com o estereótipo enumeração (<<enumeration>>) correspondentes aos valores do atributo situação das classes “Reserva”, “Carro” e “Pessoa”. Foi estabelecido o relacionamento do tipo dependência entre as classes enumeradas e as classes que as referenciam, contudo, não é obrigatório estabelecer esses relacionamentos de dependência entre essas classes e a indicação da navegabilidade nas associações binárias para definir a referência entre os objetos, sendo que a não indicação da navegabilidade representa a referência bidirecional entre os objetos associados.

Figura 4.31 | Diagrama de Classes (Projeto) – md_Locacao_de



Fonte: captura de tela do software Visual Paradigm Community Edition elaborada pela autora.

No diagrama também foi indicada a restrição – também chamado de classificador do relacionamento de generalização. Os classificadores da generalização são utilizados para definir melhor a semântica das classes especializadas derivadas da classe genérica. Os classificadores da generalização são escritos entre chaves e próximos à seta de generalização. Os classificadores predefinidos para classes especializadas são do tipo:

- **Completo:** indica que qualquer instância da superclasse (abstrata) será uma instância de uma das subclasses especializadas.
- **Incompleto:** indica que pode existir um conjunto de objetos de novas subclasses não representadas, e uma instância de seu tipo pode existir apenas da própria superclasse (concreta).
- **Disjunção:** indica que qualquer instância da superclasse pode ser uma instância de apenas uma das subclasses, ou seja, exclusiva de uma subclasse.
- **Sobreposição:** indica que uma instância da superclasse pode pertencer a mais de uma subclasse.

Por padrão, da UML 2.0 até a versão 2.4.1, o classificador da generalização era do tipo “{incompleta, disjunção}”. Na UML 2.5, o padrão foi alterado para “{incompleta, sobreposição}”, assim, uma vez que o contexto do domínio do sistema for diferente desse padrão, então deve-se representar o classificador da generalização; senão, suprime-o. A especificação UML não determina como essa equivalência semântica será implementada e como a integridade entre os objetos será mantida a partir da indicação do classificador da generalização.

REFLITA

O relacionamento do tipo generalização estabelecido entre classes do Diagrama de Classes indica o princípio de herança entre classes genéricas e classes especializadas, o qual representa a propriedade pela qual uma classe pode herdar

atributos e operações de uma classe que generaliza as características e comportamentos comuns de um grupo de objetos. Assim, no Diagrama de Classes refinado para a atividade de projeto é obrigatório estabelecer o relacionamento de generalização entre classes?

REFINAMENTO DOS ASPECTOS COMPORTAMENTAIS

A modelagem de projeto ainda pode ser complementada com demais os diagramas comportamentais e de interação da UML que não foram especificados na análise, ou os diagramas de análise podem ser refinados com detalhes que serão aplicados à implementação.

Segundo Bezerra (2014, p. 177), “[...] embora o estudo dos aspectos dinâmicos do sistema já comece na etapa de análise, é na fase de projeto que esse estudo se concretiza e onde se realiza o detalhamento das colaborações nas quais cada classe participa”.

Na modelagem comportamental de projeto com a UML, é recomendável evoluir o Modelo de Casos de Uso detalhando todos os relacionamentos de inclusão (<<Include>>) e extensão (<<Extend>>) entre os casos de uso, bem como consistir as operações listadas nas classes de objetos com as mensagens que executam operações indicadas nos Diagramas de Sequência e com as ações definidas nos Diagramas de Atividades. É importante também revisar as ações de estados representadas pelas cláusulas predefinidas “*entry*, *exit* e *do*” em cada estado representado nos Diagramas de Máquina de Estados, sendo que cada ação de estado deve indicar uma operação nas respectivas classes de objetos. Assim, a modelagem comportamental deve começar na atividade de análise para representar os aspectos dinâmicos do sistema e posteriormente ser detalhada como modelagem de projeto. Além disso, a modelagem de projeto inclui a definição do projeto de algoritmos para posterior implementação das funcionalidades do sistema, compreendendo os métodos de codificação de algoritmos em consonância com as soluções desejadas, sendo que podemos utilizar o Diagrama de Atividades da

UML para especificar os algoritmos. E ainda, pode-se elaborar o projeto das interfaces gráficas (por exemplo, formulários/telas e relatórios) correspondentes aos casos de uso que implicam uma interface amigável com alta usabilidade e facilidade de operação, seguindo os princípios básicos da Interação Humano-Computador (IHC).

De uma forma geral, Bezerra (2014) sintetiza as seguintes características da evolução da documentação de análise para projeto:

- Refinamento dos aspectos estáticos e estruturais da modelagem do sistema.
- Detalhamento dos aspectos dinâmicos da modelagem do sistema.
- Detalhamento da arquitetura do sistema, tomando-se por base a decomposição lógica e física do sistema.
- Definição dos mecanismos de armazenamento dos dados manipulados pelo sistema.
- Definição dos algoritmos a serem utilizados na implementação do sistema.
- Elaboração do projeto da interface gráfica das funcionalidades do sistema.

REFLITA

A definição da multiplicidade em uma associação estabelecida entre classes depende de pressupostos e das regras de negócio referentes ao domínio do sistema. Assim, a multiplicidade determina o número mínimo e máximo de objetos envolvidos em uma associação. Dessa forma, no Diagrama de Classes refinado para a atividade de projeto devem ser revisadas também as multiplicidades já definidas na atividade de análise.

Por fim, toda documentação da atividade de projeto do sistema deve ser estabelecida de acordo com as definições da metodologia da empresa de desenvolvimento, atendendo às características e particularidades do domínio do sistema de software.

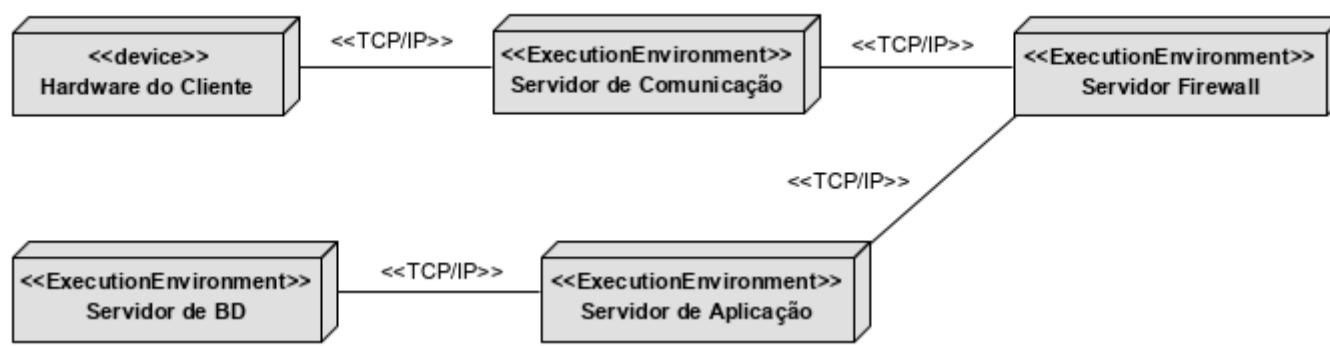
o

EXEMPLIFICANDO

Como parte da documentação de projeto, pode-se utilizar a técnica de modelagem estrutural da UML, o Diagrama de Implantação, para ilustrar a organização da arquitetura física do sistema, a partir da representação de nós e suas ligações físicas. Um nó pode representar um item de hardware do sistema ou um outro dispositivo integrado ao sistema e os ambientes de execução que integram a arquitetura física do sistema. A Figura 4.32 ilustra um exemplo do Diagrama de Implantação, demonstrando a associação entre o nó do tipo dispositivo do “Hardware do Cliente” que representa o acesso do cliente à web para fazer seu cadastro ou uma reserva, por exemplo, com os nós do tipo ambiente de execução “Servidor de Comunicação”, “Servidor Firewall”, “Servidor de Aplicação” e o “Servidor de BD”.

Ver anotações

Figura 4.32 | Diagrama de Implantação – Sistema “Locação de Veículos”



Fonte: elaborada pela autora.

PERSISTÊNCIA DE OBJETOS PARA O MODELO RELACIONAL

Os princípios básicos do paradigma orientado a objetos e do modelo relacional são distintos, considerando que as tecnologias de orientação a objetos se baseiam no princípio do encapsulamento, em que os objetos são abstrações de um comportamento. No modelo relacional, os elementos correspondem a dados no formato tabular que utilizam um Sistema Gerenciador de Banco de Dados Relacional (SGBDR). Dessa forma, outro aspecto importante na modelagem de projeto é relativo ao mecanismo de armazenamento persistente de dados correspondentes ao mapeamento de classes de objetos para o modelo relacional.

Para especificar o mapeamento de classes para tabelas do modelo de dados relacional, é usual adotar técnicas de modelagem de dados e/ou definir o uso de frameworks de mapeamento objeto relacional, como estratégia de armazenamento persistente. Assim, como parte da documentação que envolve o projeto de banco de dados, deve-se apresentar no mínimo a construção do esquema do banco de dados.

Considerando que foi definido o uso de SGBDR como mecanismo de armazenamento dos objetos, é necessário fazer o mapeamento dos valores de atributos de objetos das classes persistentes para as tabelas de banco de dados relacional, com base no Modelo de Classes.

A maioria das ferramentas CASE de modelagem orientada a objetos fornece a funcionalidade de mapeamento automático para geração de um esquema relacional, a partir do modelo de classes e da definição do SGBDR a ser utilizado. No entanto, é importante que você tenha

conhecimento dos procedimentos existentes a serem adotados para a conferência ou elaboração do mapeamento. Dessa forma, vamos conhecer algumas regras fundamentais que refletem em termos práticos para fazer o mapeamento de objetos para o modelo relacional.

Primeiramente, deve-se identificar se os objetos das classes são objetos transientes ou objetos persistentes. Normalmente os objetos de entidade são os objetos persistentes, os quais devem ser armazenados em meio físico durante a execução do sistema para serem manipulados. Os objetos transientes existem somente durante uma sessão de uso do sistema e geralmente são os objetos de fronteira e de controle.

Na sequência, são analisadas as classes persistentes e seus relacionamentos e aplicadas as alternativas de mapeamento apresentadas a seguir, considerando a notação indicada para manter uma padronização da representação das tabelas e, assim, constituir o esquema do Banco de Dados Relacional (BDR):

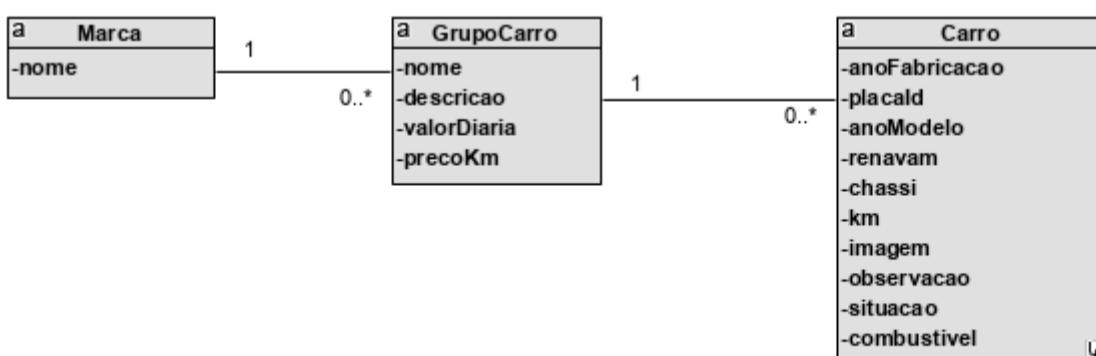
Nome da Tabela (coluna 1, coluna 2, coluna 3, coluna 4,... coluna n)

- Cada coluna representa um atributo da classe mapeada, no entanto, atenção aos atributos derivados, pois eles não são mapeados para uma coluna.
- Destaca-se a coluna que representa a chave primária com sublinhado simples e as colunas que representam chaves estrangeiras com sublinhado tracejado.
- Representa-se em cada tabela derivada de classe, no geral, uma coluna que indica o identificador (Id) para a chave primária. Essa estratégia de notação dos “Ids” define a identidade independente dos objetos, conforme os princípios da orientação a objetos.

Segundo Rumbaugh (1997), as principais alternativas de mapeamento de classes para tabelas são:

- **Mapeamento de associação binária:** para as classes relacionadas com associação binária, com multiplicidade um-para-muitos, mapeia-se cada classe em uma tabela, conforme exemplo ilustrado na Figura 4.33.

Figura 4.33 | Recorte do Diagrama de Classes com Associação Binária (um-para-muitos)



Fonte: elaborada pela autora.

Mapeamento:

Marca (marcald, nome).

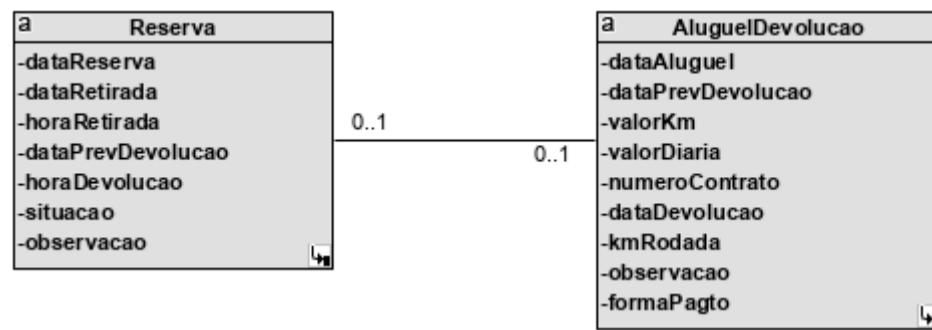
GrupoCarro (grupoCarrold, nome, descricao, valorDiaria, precoKm, marcald).

Carro (carrold, anoFabricacao, placa, anoModelo, renavam, chassi, km, imagem, observacao, situacao, combustivel, grupoCarrold).

Para associação binária com multiplicidade um-para-um, pode-se mapear as classes cada uma em uma tabela ou unir os atributos das duas classes em uma única tabela. Essa decisão depende das preferências do projetista de banco de dados em termos da extensibilidade, do número de tabelas e de desempenho.

A Figura 4.34 ilustra o mapeamento sendo mantido em tabelas separadas. No exemplo ilustrado, não foram representadas as outras classes que se associam com ambas as classes, que seriam outras chaves estrangeiras. Por isso, no final de cada tabela correspondente às classes, foi incluída a sigla “demaisFK”. Essa representação também foi adotada em outros exemplos a seguir.

Figura 4.34 | Recorte do Diagrama de Classes com Associação Binária (um-para-um)



Fonte: elaborada pela autora.

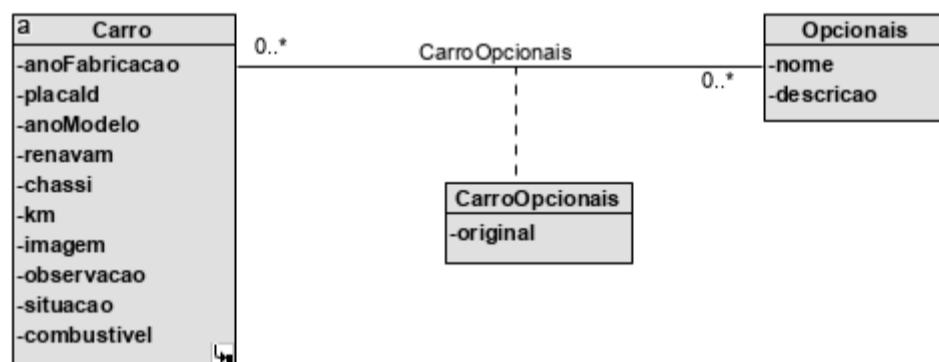
Mapeamento:

Reserva (reservald, dataReserva, dataRetirada, horaRetirada, dataPrevDevolucao, situacao, observacao, demaisFK).

AluguelDevolucao(aluguelDevolucaold, dataAluguel, dataPrevDevolucao, valorKm, valorDiaria, numeroContrato, dataDevolucao, kmRodada, observacao, formaPagto, reservald, demaisFK).

Mapeamento de classe associativa: para as classes relacionadas com associação de classe associativa, mapeia-se cada classe em uma tabela, conforme exemplo ilustrado na Figura 4.35.

Figura 4.35 | Recorte do Diagrama de Classes com Classe Associativa



Fonte: elaborada pela autora.

Mapeamento:

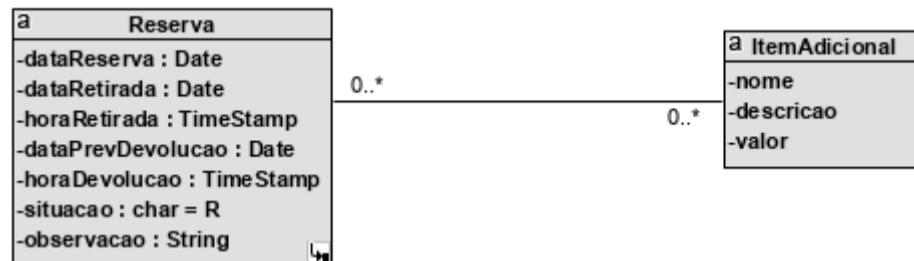
Carro (carrold, anoFabricacao, placa, anoModelo, renavam, chassi, km, imagem, observacao, situacao, combustivel, grupoCarrold).

Opcionais (OpcionaisId, nome, descricao).

CarroOpcionais (carrold, OpcionaisId, original).

Para classes relacionadas com multiplicidade muitos-para-muitos, cria-se a terceira tabela com apenas a chave primária composta, conforme mostra a Figura 4.36.

Figura 4.36 | Recorte do Diagrama de Classes com Associação Binária (muitos-para-muitos)



Fonte: elaborada pela autora.

Mapeamento:

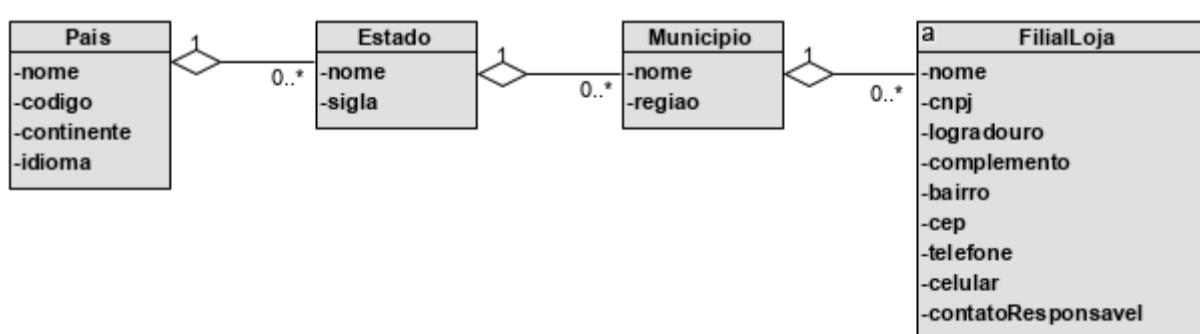
Reserva (reservaid, dataReserva, dataRetirada, horaRetirada, dataPrevDevolucao, situacao, observacao, demaisFK).

ItemAdicional (itemAdicionalId, nome, descricao, valor).

ReservaItemAdicional (reservaid, itemAdicionalId).

- **Mapeamento de agregação:** para classes relacionadas com associação do tipo agregação, mapeia-se a classe “Todo” e “Parte” para tabelas individuais. O identificador da classe “Todo” é indicado como chave estrangeira na tabela que representa a classe “Parte”, conforme é mostrado na Figura 4.37, ou seja, mesma forma de mapear classes com associação binária.

Figura 4.37 | Recorte do Diagrama de Classes com Agregação



Fonte: elaborada pela autora.

Mapeamento:

Pais (paisId, nome, código, continente, idioma).

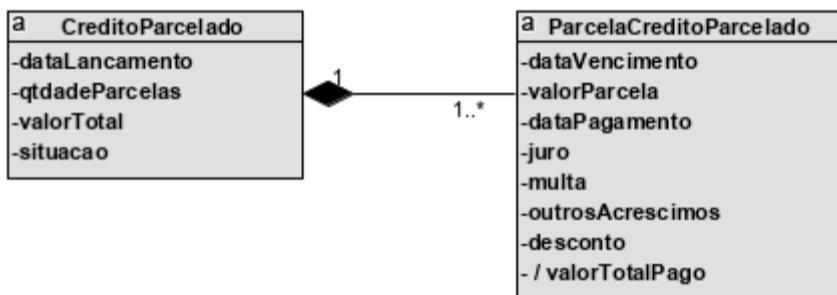
Estado (estadoid, nome, sigla, paisid).

Municipio (municipioId, nome, regiao, estadoId).

FilialLoja (filialLojaId, nome, cnpj, logradouro, complemento, bairro, cep, telefone, celular, contatoResponsavel, municipioId).

- **Mapeamento de composição:** para classes relacionadas com associação do tipo composição (tipo especial de agregação), mapeia-se a classe “Todo” e “Parte” para tabelas individuais. O identificador da classe “Todo” torna-se parte da chave primária na tabela que representa a classe “Parte”, conforme o exemplo ilustrado na Figura 4.38.

Figura 4.38 | Recorte do Diagrama de Classes com Composição



Fonte: elaborada pela autora.

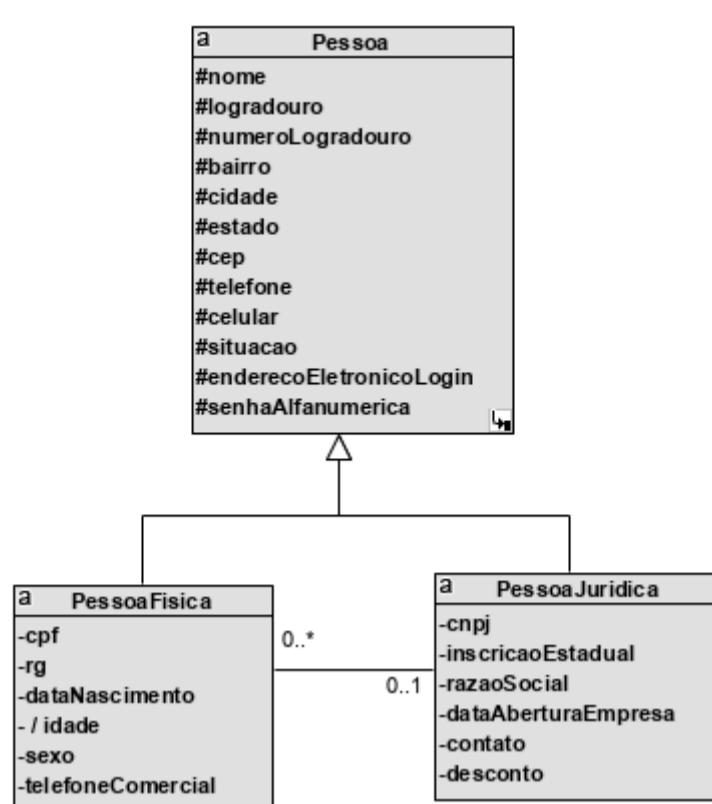
Mapeamento:

CreditoParcelado (creditoParceladoId, dataLancamento, qtdadeParcelas, valorTotal, situacao, demaisFK).

*ParcelaCreditoParcelado (creditoParceladoId,
parcelaCreditoParceladoId, dataVencimento, valorParcela,
dataPagamento, juro, multa, outrosAcrescimos, desconto).*

- **Mapeamento de generalização:** existem três abordagens para o mapeamento do relacionamento do tipo generalização em tabelas. A abordagem normalmente define que a superclasse e as subclasses são mapeadas cada uma em uma tabela, com a utilização de um “Id” compartilhado e a criação de um atributo tipo na tabela que representa a superclasse, para identificar os tipos de objetos representados pelos objetos das subclasses, conforme o recorte ilustrado na Figura 4.39.

Figura 4.39 | Recorte do Diagrama de Classes com Generalização



Fonte: elaborada pela autora.

Mapeamento - Alternativa 1:

Pessoa (pessoald, nome, logradouro, numeroLogradouro, bairro, cidade, estado, cep, telefone, celular, situacao, enderecoEletronicoLogin, senhaAlfanumerica, tipoPessoa [PF/PJ], demaisFK).

PessoaFisica (pessoald, cpf, dataNascimento, sexo, telefoneComercial, pessoajd, demaisFK).

PessoaJuridica (pessoald, cnpj, inscricaoEstadual, razaoSocial, dataAberturaEmpresa, contato, desconto, demaisFK)

A segunda e a terceira abordagens são consideradas alternativas de mapeamento de generalização. Segundo Rumbaugh (1997, p. 506), “ [...] elas são motivadas pelo desejo de eliminar a navegação de superclasse para subclasse, melhorando o desempenho”. A segunda abordagem define a eliminação da tabela correspondente à superclasse e mapeia-se uma tabela correspondente a cada subclasse, reproduzindo todos os atributos da superclasse em cada tabela da subclasse.

Mapeamento - Alternativa 2:

PessoaFisica (pessoaldF, nome, logradouro, numeroLogradouro, bairro, cidade, estado, cep, telefone, celular, situacao, enderecoEletronicoLogin, senhaAlfanumerica, cpf, dataNascimento, sexo, telefoneComercial, pessoajd, demaisFK).

PessoaJuridica (pessoajd, nome, logradouro, numeroLogradouro,

*bairro, cidade, estado, cep, telefone, celular, situação,
enderecoEletronicoLogin, senhaAlfanumerica, cnpj, inscricaoEstadual,
razaoSocial, dataAberturaEmpresa, contato, desconto, demaisFK).*

A terceira abordagem define a criação de uma única tabela correspondente à superclasse, unindo todos os atributos das subclasses ao nível da superclasse.

Mapeamento – Alternativa 3:

*Pessoa (pessoald, nome, logradouro, numeroLogradouro, bairro,
cidade, estado, cep, telefone, celular, situacao, enderecoEletronicoLogin,
senhaAlfanumerica, tipoPessoa [PF/PJ], cpf, dataNascimento, sexo,
telefoneComercial, pessoadj, cnpj, inscricaoEstadual, razaoSocial,
dataAberturaEmpresa, contato, desconto, demaisFK).*

Para garantir a consistência da modelagem de análise e projeto de um software, bem como a evolução da modelagem, é importante utilizar todos os recursos das ferramentas CASE de modelagem e o recurso de gerenciamento de versões ou visões dos modelos especificados, para, assim, facilitar a leitura e interpretação do conjunto de diagramas estáticos e dinâmicos que compõem a documentação de um sistema de software.

REFERÊNCIAS

BEZERRA, E. **Princípios de análise e projeto de sistemas com UML**. 3. ed. Rio de Janeiro: Elsevier, 2014.

OLIVEIRA, M. M. A. *et al.* Um Estudo comparativo entre banco de dados orientado a objetos, banco de dados relacionais e framework para mapeamento objeto/relacional, no contexto de uma aplicação web.

HOLOS, [s. l.], v. 31, n. 1, p. 182–198, 2015. DOI

10.15628/holos.2015.1153. Disponível em: Biblioteca Virtual – EBSCO HOST. <https://bit.ly/3bBE3BI>. Acesso em: 17 jun. 2020.

PRESSMAN, R.; MAXIM, B. **Engenharia de software**. 8. ed. Porto Alegre: AMGH, 2016

RODRIGUES DE OLIVEIRA, L. *et al.* Desenvolvimento e Avaliação de um Perfil UML para Modelagem de Jogos Educacionais Digitais. **Revista**

Brasileira de Informática na Educação, [s. l.], v. 26, n. 2, p. 124–143,

2018. DOI 10.5753/RBIE.2018.26.02.124. Disponível em: Biblioteca

Virtual – EBSCO HOST. <https://bit.ly/3bBE3BI>. Acesso em: 17 jun. 2020.

RUMBAUGH, J. *et al.* **Modelagem e projetos baseados em objetos**. Rio de Janeiro: Campus, 1997.

o

Ver anotações

FOCO NO MERCADO DE TRABALHO

TRANSIÇÃO DA ATIVIDADE DE ANÁLISE PARA PROJETO

Iolanda Cláudia Sanches Catarino

Ver anotações

PERSISTÊNCIA DOS OBJETOS

Mapeamento das classes em tabelas de banco de dados relacional, conforme os tipos de relacionamentos estabelecidos entre as classes de objetos.



Fonte: Shutterstock.

Deseja ouvir este material?

Áudio disponível no material digital.

Considerando as alternativas de mapeamento de classes para tabelas de banco de dados relacional, segue o esquema do banco de dados correspondente ao sistema “Locação de Veículos”.

Uma sugestão para elaborar o mapeamento é iniciar o mapeamento das tabelas que não têm chave estrangeira, posteriormente as demais tabelas e, ao finalizar o mapeamento, classificar as tabelas e listá-las em ordem alfabética.

1. Esquema do módulo md_Locacao_dc:

AluguelDevolucao(aluguelDevolucaold, dataAluguel, dataPrevDevolucao, valorKm, valorDiaria, numeroContrato, dataDevolucao, kmRodada, observacao, formaPagto, reservald, carrold, pessoald, filialIdR, filialIdD, tipoPagamentold)

AluguelDevolucaoItemAdicional (aluguelDevolucaold, itemAdicionalId)

Carro (carrold, placa, anoFabricacao, anoModelo, renavam, chassi, km, imagem, observacao, situacao, combustivel, grupoCarrold)

GrupoCarro (grupoCarrold, nome, descricao, valorDiaria, precoKm, marcald)

ItemAdicional (itemAdicionalId, nome, descricao, valor)

Marca (marcald, nome)

Pais (paisId, nome, código, nacionalidade, idioma, continente, sigla)

Pessoa (pessoald, nome, logradouro, numeroLogradouro, bairro, cidade, estado, cep, telefone, celular, situacao, endereçoEletronicoLogin, senhaAlfanumerica, tipoPessoa [PF/PJ], paisId)

PessoaFisica (pessoald, cpf, dataNascimento, sexo, telefoneComercial, pessoaldJ, profissaold)

PessoaJuridica (pessoald, cnpj, inscriçãoEstadual, razãoSocial, dataAberturaEmpresa, contato, desconto, ramold)

Profissao (profissaold, cbo, nome)

RamoAtividade (ramold, cnae, nome)

Reserva (reservald, dataReserva, dataRetirada, horaRetirada, dataPrevDevolucao, situacao, observacao, grupoCarrold, pessoald, filialIdR, filialIdD)

ReservaItemAdicional (reservald, itemAdicionalId)

2. Esquema do módulo md_Pagamento_dc:

Caixa (caixald, data, horaAbertura, horaFechamento, saldoEntrada, saldoMovimentacao, saldoFechamento, situacao, aluguelDevolucaold)

CreditoParcelado (creditoParceladold, dataLancamento, qtdadeParcelas, valorTotal, situacao, aluguelDevolucaold)

ParcelaCreditoParcelado (creditoParceladold,

parcelaCreditoParcela(parcelaCreditoParcelaId, *dataVencimento*, *valorParcela*,
dataPagamento, *juro*, *multa*, *outrosAcrescimos*, *desconto*)
TipoPagamento (tipoPagamentoId, *nome*)