ANÁLISE DE SISTEMAS

AULA 4

Profª Adriana Bastos da Costa

CONVERSA INICIAL

Nas aulas anteriores, conversamos sobre modelagem de processos de negócios, sobre análise de sistemas focada na análise estruturada e sobre engenharia de requisitos e sua importância para a correta construção de um software. A cada aula, fica mais claro que a análise de sistemas é fundamental para todo profissional que deseje desenvolver softwares, pois é a disciplina que permite traduzir, por meio de modelos e documentação específica, as necessidades de negócio do cliente em um projeto técnico de software. A análise de sistemas requer que seja feito planejamento técnico de um software antes de ele ser codificado, para que os objetivos de negócio do cliente sejam atendidos. E a base para todo software bem construído está no levantamento e na análise adequada e completa dos seus requisitos.

Nesta aula, vamos conversar sobre a *Unified Modelling Language* (UML), discutindo sobre a sua origem e a sua importância na construção de software. Vamos também compreender como é estruturado o diagrama de casos de uso. Projetar e construir um programa de computador não é uma tarefa trivial, pois envolve conhecer bem o problema que se quer resolver e se traduzir a solução ideal para o problema em linhas de código; portanto, entender bem os requisitos ou o que o software precisa fazer para atender às necessidades do cliente é base de tudo. Se os requisitos de software não forem bem analisados, fica mais difícil projetar uma solução adequada e eficiente para resolver o problema do cliente.

Esta aula está organizada em cinco grandes temas, a saber:

1. O que é UML?
2. Modelo orientado a objetos.
3. Técnicas de construção do diagrama de caso de uso.
4. Componentes de um diagrama de caso de uso.
5. Analisando um exemplo de diagrama de caso de uso.

TEMA 1 – O QUE É UML?

Para falar de UML é preciso conhecer o *Object Management Group* (OMG), uma organização internacional que aprova padrões reconhecidos pela área de tecnologia da informação (TI) para aplicações orientadas a objetos, ou

seja, todos os modelos e padrões utilizados para se criar modelos de software são organizados e mantidos por esse grupo.

A notação UML começou a ser criada em outubro de 1994, quando James Rumbaughq se juntou a Grady Booch na Rational Software Corporation, uma grande empresa de software da época. O constante crescimento da Rational fez com que surgisse a necessidade de organizar e definir as etapas de desenvolvimento adotadas em seus projetos, elaborando-se um ciclo de vida para desenvolvimento de softwares. Foi concebido, assim, o processo de desenvolvimento unificado denominado *Rational unified process* (RUP). Nesse mesmo contexto, o mundo do software também começou a perceber a necessidade de utilizar um processo mais formal e padronizado para desenvolver softwares melhores. Dessa forma, o RUP se popularizou e se tornou o ciclo de vida mais empregado quando se fala em metodologia tradicional de desenvolvimento de softwares, mais até do que o modelo cascata.

A UML é uma linguagem ou notação de diagramas que serve para especificar, visualizar e documentar modelos de software desenvolvidos sob os preceitos da orientação por objetos. Como a UML não é um método de desenvolvimento, ela não diz o que fazer primeiro, o que fazer depois ou como desenhar o sistema, mas ajuda a visualizá-lo e a comunicá-lo aos outros. A UML é controlada pelo OMG e considerada como **a norma da indústria de software** para descrevê-lo graficamente. Não apenas no Brasil, mas em todo o mundo, a UML é a linguagem mais utilizada para modelar software.

Segundo Pressman e Maxim (2016), “a UML fornece uma gama de diagramas que podem ser usados para análise e projeto tanto em nível de sistema quanto de software”. A existência de um modelo visual facilita a comunicação e faz com que os envolvidos na construção do software tenham a mesma visão e conhecimento sobre o sistema. Cada símbolo gráfico utilizado tem uma semântica bem definida, pois atende a um padrão. A padronização facilita o uso e o entendimento do modelo. Por sinal, para se compreender bem a UML é necessário, antes de tudo, entender o que são modelos quando se fala em engenharia de software. A seguir, vamos discutir sobre o que são modelos de análise e de projeto de software.

1.1 O que são modelos?

Um modelo de processo de desenvolvimento de software, ou, simplesmente, modelo de processo, pode ser visto como uma representação ou abstração dos objetos e atividades envolvidas no processo de criação de um software. Dessa forma, um modelo é uma simplificação da realidade, construído para se compreender melhor o sistema que será desenvolvido e codificado.

Algumas facilidades que podem ser alcançadas pela modelagem do software são que:

- Os modelos ajudam a visualizar o sistema como ele é ou como desejamos que seja, mostrando visualmente suas interfaces e modo de funcionamento.
- Os modelos permitem especificar a estrutura ou o comportamento de um sistema. Existem vários tipos de modelos, cada um com um objetivo e focando uma parte do software.
- Os modelos proporcionam um guia para a construção do sistema, possibilitando analisar a melhor solução possível para o software em questão.
- Os modelos documentam as decisões de projeto tomadas e apoiam a tomada de uma melhor decisão com base em oferta de soluções possíveis para o problema que se quer resolver.
- Os modelos permitem visualizar o sistema em diversos níveis de abstração. Por isso, os diversos modelos criados para um software precisam ser coerentes entre si, se complementarem e darem uma visão completa de como o software deve ser construído e de como ele funcionará.
- Os modelos facilitam a comunicação entre os membros da equipe de formulação do software, pois, na UML, os modelos expressam duas visões diferentes, porém complementares:
    1. **visão estrutural**: os modelos criados tentam capturar a estrutura do sistema, ou seja, quais elementos compõem a estrutura do sistema e como eles se relacionam;
    2. **visão comportamental**: os modelos criados tentam capturar a dinâmica do sistema, ou seja, como os elementos que compõem o

sistema se comunicam e como se comportam e respondem a diversos estímulos.

1.2 Modelos de UML

Resumindo, modelos são abstrações ou uma simplificação da realidade que vai ser automatizada por meio de um software; e a UML é a linguagem mais conhecida no mundo da TI para fazer modelagem de softwares. A UML é composta por muitos elementos e por diferentes modelos, que representam as partes de um sistema de software. A junção de todos os modelos criados deve dar a visão completa de como o software será construído e funcionará.

Os elementos UML são usados para criar diagramas que representam uma determinada parte ou um ponto de vista do sistema. Os diagramas mostram visualmente a solução definida para a construção do software. A UML é muito rica e dispõe de vários diagramas diferentes, com objetivos próprios. Como já foi explicado, o OMG é responsável por manter e fazer evoluir a UML, mediante novas versões lançadas periodicamente. A versão atual da UML é a 2.0 e pode ser encontrada no site da própria empresa.

A seguir, alguns tipos de diagramas previstos na UML 2.0:

a. **Diagramas estruturais ou estáticos**
   - Diagrama de classes.
   - Diagrama de componentes.
   - Diagrama de implantação.
   - Diagrama de objetos.
   - Diagrama de pacotes.
   - Diagrama de estrutura da composição.
   - Diagrama combinado componentes/implantação.

b. **Diagramas comportamentais ou dinâmicos**
   - Diagrama de caso de uso.
   - Diagrama de sequência.
   - Diagrama de atividades.
   - Diagrama de estados.
   - Diagrama de comunicação (antigo diagrama de colaboração da UML 1.4).
   - Diagrama de visão geral da interação.

- Diagrama de tempo.

Cada diagrama tem um objetivo e foca uma parte do software. Agora, vamos compreender o objetivo principal de cada um desses tipos de diagrama:

- **Diagrama de classes**: é o principal diagrama de análise e projeto OO (visão estrutural). Nele, são mostradas as classes do sistema, seus relacionamentos (herança, associação, composição e agregação), seus atributos (informações relacionadas com cada classe) e seus métodos (ações que podem ser realizadas com cada classe).
- **Diagrama de componentes**: modela as partes do software no ambiente onde ele está sendo implementado, ou seja, no seu ambiente de desenvolvimento (códigos-fonte, *dynamic-link library* – DLL, executáveis, *hypertext markup language* – HTML, *cascading style sheets* – CSS, Java *archive* – JAR, *class*, bibliotecas, componentes específicos etc.).
- **Diagrama de implantação**: modela o *hardware* de cada um dos ambientes utilizados, ou seja, modela o *hardware* segregado em ambientes de desenvolvimento, de teste, de homologação e de produção.
- **Diagrama de objetos**: modela instâncias reais das classes e seus relacionamentos, ou seja, modela fatos reais ou exemplos do dia a dia. Usado como complemento do diagrama de classes.
- **Diagrama de pacotes**: mostra como os elementos estão organizados em pacotes e as dependências entre esses pacotes.
- **Diagrama de estrutura da composição**: modela um comportamento ou estrutura complexa em que as classes e/ou os componentes estão envolvidos e os pontos de interação usados para acessar os recursos dessa estrutura.
- **Diagrama combinado componentes/implantação**: une os modelos de componentes e implantação mostrando como os componentes de software estão distribuídos nos componentes do hardware.
- **Diagrama de caso de uso**: nele são especificados e detalhados os requisitos funcionais, com descrição dos cenários nos quais os atores interagem com o sistema. São bastante úteis na comunicação entre todos os envolvidos na construção do software, incluindo os usuários.

- **Diagrama de sequência**: é o modelo mais usado para apoiar a visão dinâmica de um sistema. É empregado na análise de projeto para modelar a interação entre os diversos objetos de uma linha de tempo.
- **Diagrama de atividades**: normalmente usado para modelagem de processos de negócio ou para detalhamento da lógica de negócio. Também pode ser utilizado para descrever casos de uso. Permite a modelagem de fluxo de controle e de dados.
- **Diagrama de estados**: modela como os estímulos ou eventos externos causam mudanças no estado de um objeto, no decorrer do seu ciclo de vida.
- **Diagrama de comunicação**: é equivalente ao diagrama de sequência, mas sem o foco temporal (algumas ferramentas UML geram diagrama de comunicação com base no diagrama de sequência). O diagrama de comunicação também pode ser chamado de *diagrama de colaboração*.
- **Diagrama de visão geral da interação**: usa o mesmo layout do diagrama de atividades. Entretanto, ao invés de modelar atividades, são modelados blocos de interação associados ao diagrama de sequência, ou seja, tem-se com ele uma visão macro, ao invés de uma visão detalhada da interação.
- **Diagrama de tempo**: como o diagrama de estados, também modela mudanças no estado de um objeto. Entretanto, enfatiza a questão do tempo durante o ciclo de vida da informação.

Ao longo das nossas aulas, vamos detalhar os principais diagramas. Quando os classificamos como principais, isso não significa atribuir-lhes um dado grau de importância, pois todos os diagramas são importantes, de acordo com o seu objetivo; só desejamos agregar mais ou menos valor ao entendimento do software. Pensando um pouco nos conceitos de agilidade, tudo o que agrega valor à compreensão e à melhoria de qualidade de um software deve ser utilizado. O que não devemos permitir é o desperdício, ou seja, não devemos gastar tempo de um projeto de software criando diagramas desnecessários, que não ajudem a equipe que o formula a compreender melhor o funcionamento do software.

Dessa forma, o pensamento ágil defende que tudo o que não agrega valor é desperdício e deve ser evitado; mas que tudo o que ajuda no entendimento, na análise e no projeto da melhor solução, aquela baseada na experiência e na

necessidade do usuário, deve ser utilizado, independentemente de se tratar de um conceito, uma técnica ou uma ferramenta chamada de *tradicional* ou de *ágil*. É por isso que muitos autores continuam defendendo o uso da UML mesmo quando a metodologia ágil é a empregada no desenvolvimento de um software, dado que a UML agrega valor quando é utilizada de maneira adequada.

Nesta aula, vamos detalhar para que serve e como deve ser construído um diagrama de caso de uso.

TEMA 2 – MODELO ORIENTADO A OBJETOS

No desenvolvimento de sistemas, tratamos a orientação a objetos como um paradigma de programação, ou seja, como uma forma de se implementar um código. No entanto, esse é um tema com aplicação em diversas áreas e, por esse motivo, é possível encontrar na literatura sobre o assunto muito sobre a análise orientada a objetos, que é o nosso foco. No desenvolvimento de software, a orientação a objetos é vista como um conceito da engenharia de software, para o que os elementos de uma solução são representados como objetos. É interessante compreender que os mecanismos que o cérebro humano utiliza para pensar e ver o mundo são totalmente orientados a objetos; por isso, a orientação a objetos, no desenvolvimento de software, é amplamente empregada, uma vez que torna algo complexo, como um software, em algo mais simples de ser compreendido. Assim como costumamos categorizar e agrupar diferentes elementos, em nossa percepção, para poder compreendê-los, também o fazemos em uma análise orientada a objetos, de forma a simplificar o entendimento do mundo real que deve ser modelado e automatizado por meio de um software.

A orientação a objetos é um processo conceitual independente de uma linguagem de programação, pois tem como foco visualizar o domínio do problema a ser automatizado como uma coleção de objetos e métodos associados. Todo objeto é identificável e deve ter um nome claro e direto, que identifique o seu real objetivo, ou seja, as coisas do mundo real são denominadas *objetos*. Um objeto é, portanto, uma entidade, real ou abstrata, que modela um conceito presente na realidade humana, ocupando espaço físico ou lógico. O objeto é, pois, a base para todos os outros conceitos da orientação a objetos, pois facilita a compreensão do mundo real e oferece uma base para a

implementação de um software. Um objeto denota uma entidade de natureza física, conceitual ou de software:

- entidade física: um carro, uma pessoa, um livro;
- entidade conceitual: um diagrama de um sistema;
- entidade de software: um *checkbox* em uma página web.

Algumas outras características dos objetos são também fundamentais para se garantir uma boa análise do software, como:

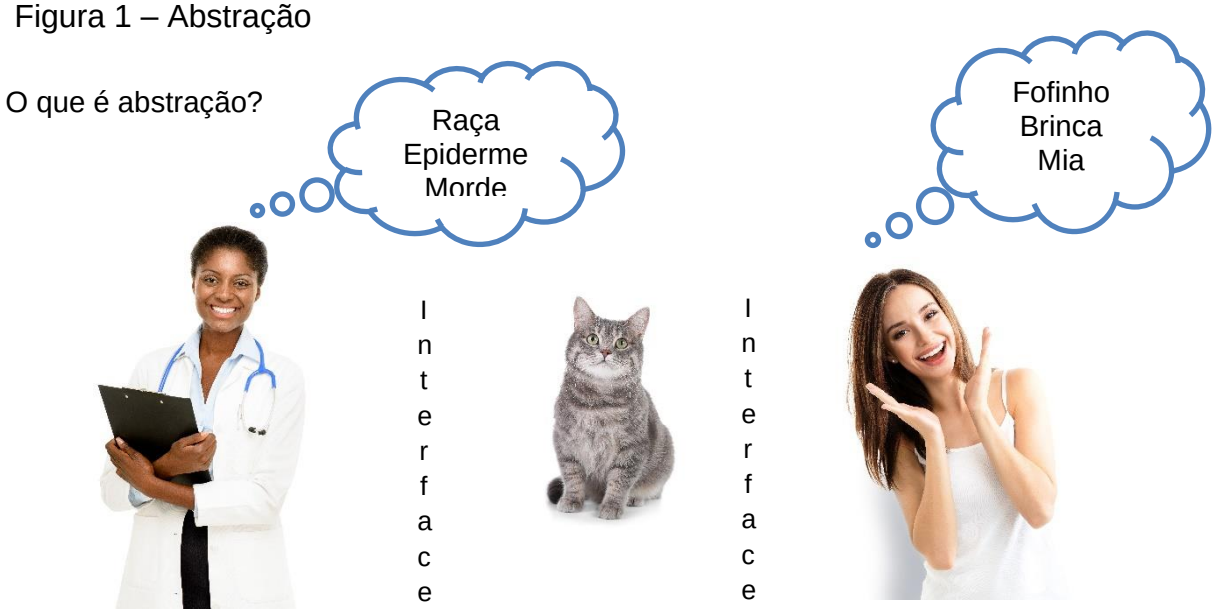
- Identidade de um objeto: é o que identifica univocamente um objeto, dentre os demais existentes. A identidade permite que um objeto seja referenciado.
- Características de um objeto: descrevem propriedades do objeto. São mutáveis, ao longo do tempo, e também chamadas de *atributos do objeto*.
- Comportamentos de um objeto: determinam como um objeto reage a estímulos do mundo real ou de outros objetos. Também são chamados de *métodos do objeto*.

2.1 Conceitos da orientação a objetos

Além de entender o que é um objeto, é importante conhecer os principais conceitos relacionados com a orientação a objetos. São eles: abstração, encapsulamento e herança. Vamos detalhar cada um desses conceitos fundamentais para se compreender como funciona a orientação a objetos.

- **Abstração**: a abstração consiste em se concentrar nos aspectos essenciais, próprios de uma entidade, e em se ignorar suas propriedades acidentais, ou seja, dar foco a aspectos relevantes para um determinado propósito. Esse conceito é muito importante para simplificar os conceitos complexos.

Figura 1 – Abstração



Nota: duas ou mais pessoas podem ver características distintas no mesmo objeto. Ele comporta ambas as características, porém, cada uma das pessoas visualiza aquilo que mais atende à sua realidade, mediante seus próprios filtros internos. Isso se chama *abstração*.

Créditos: Aila Images/Shutterstock; New Africa/Shutterstock; Vgstudio/Shutterstock.

- **Encapsulamento**: consiste na separação dos aspectos externos de um objeto, acessíveis por outros objetos, dos detalhes internos da implementação daquele objeto, que ficam ocultos dos demais objetos. Ou melhor: significa separar o software em partes o mais isoladas possível. A ideia é tornar o software mais flexível, fácil de modificar e de criar novas implementações. É uma forma de restringir o acesso ao comportamento interno de um objeto. Cada objeto possui uma interface, que é o que ele conhece e o que ele sabe fazer, sem descrever como é feito. Por exemplo, para dirigir não precisamos conhecer o funcionamento interno do acelerador e da embreagem, só precisamos saber que, para o carro andar, é preciso pisar na embreagem, passar a marcha e pisar no acelerador. Esse conhecimento já nos basta – a interface embreagem, marcha e acelerador já é suficiente para fazer o carro andar, não importando como isso tudo funciona internamente no carro.
- **Herança**: é o compartilhamento de atributos e operações entre objetos, com base em um relacionamento hierárquico. Permite que uma estrutura comum seja compartilhada por diversos outros objetos relacionados, sem

redundâncias. Cada objeto, em um nível de hierarquia, herda as características dos objetos dos níveis acima.

Portanto, compreender o funcionamento da orientação a objetos facilita o entendimento da análise de um problema de software e do seu funcionamento.

TEMA 3 – TÉCNICAS DE CONSTRUÇÃO DO DIAGRAMA DE CASO DE USO

O diagrama de caso de uso normalmente é o primeiro diagrama a ser construído, após o levantamento de requisitos do software, pois construir um diagrama de caso de uso é uma das formas de se analisar o relacionamento de cada uma das funcionalidades que o software deve executar. Inicia-se a modelagem do diagrama de caso de uso com a descoberta dos atores ou dos casos de uso do sistema, com base nos seus requisitos iniciais. Se descobertos os atores, examinam-se suas necessidades para se determinar os casos de uso; se conhecidos inicialmente os casos de uso, determina-se quem deve interagir com eles para se chegar aos atores. Para cada caso de uso, deve-se descrever os cenários principais e os alternativos relevantes, como já foi estudado.

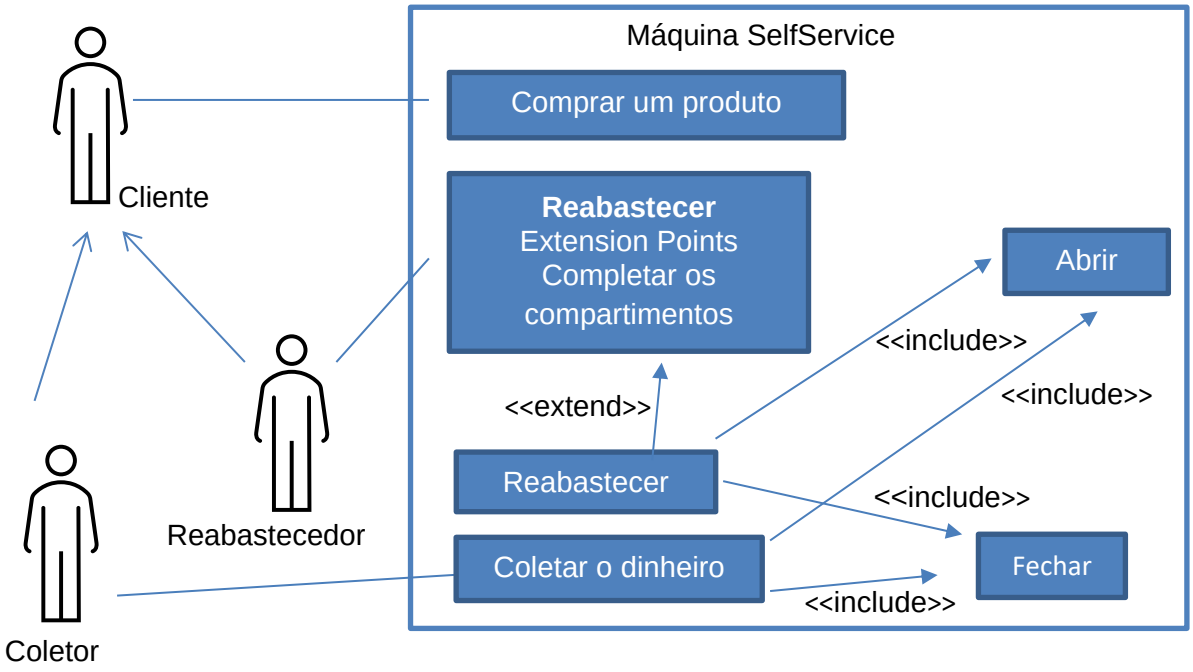
O diagrama de caso de uso é bastante utilizado em projetos por vários motivos, tais como: expressar a fronteira do sistema; mostrar a visão estática do caso de uso, pois a sua visão é especificada na descrição dos cenários; além disso, por ser uma representação gráfica, tal diagrama fornece uma visão geral dos relacionamentos entre os casos de uso e os atores de um sistema. O diagrama de caso de uso serve também para auxiliar a comunicação entre os analistas e os usuários, pois descreve, de forma visual, o relacionamento entre as funcionalidades do sistema. Essas funcionalidades do sistema são apresentadas, no diagrama de caso de uso, do ponto de vista do usuário.

O usuário pode ver, no diagrama de caso de uso, as principais funcionalidades de um sistema, validando, dessa forma, se o levantamento de requisitos está completo e se o entendimento sobre o funcionamento do software é correto, com base no relacionamento entre as funcionalidades identificadas. Ou seja, o diagrama de caso de uso tem como foco mostrar o comportamento externo do software, não se preocupando com como essas funções serão implementadas. Esse objetivo será contemplado em outro diagrama da UML, o diagrama de classe.

Em suma, o diagrama de caso de uso é representado por atores, casos de uso e relacionamentos entre esses elementos. Segundo Booch, Rumbaugh e Jacobson (2005), “um caso de uso especifica o comportamento de um sistema (ou de parte de um sistema), e é uma descrição de um conjunto de sequências de ações para produzir um resultado observável do valor de um ator”.

O diagrama de caso de uso é representado na Figura 2.

Figura 2 – Representação de um diagrama de caso de uso



Os elementos que aparecem no diagrama de caso de uso do exemplo da Figura 2 são: os atores, representados pelos bonecos; os casos de uso, representados pelas circunferências; e os relacionamentos, representados por linhas retas ou pontilhadas, dependendo do tipo de relacionamento. O diagrama de caso de uso desse exemplo mostra um sistema que controla uma máquina de produtos *self-service* como essas que encontramos em um posto de gasolina. Os atores que interagem com esse sistema são o cliente em si, o profissional que reabastece a máquina (reabastecedor) e o profissional que coleta o dinheiro usado pelos clientes para comprar os produtos oferecidos (coletor). Os casos de uso identificados são os descritos nas elipses apresentadas. O diagrama demonstra ainda qual ator interage com cada caso de uso e como os casos de uso interagem entre si. É importante reforçar que o diagrama de caso de uso apresenta visualmente o processo de negócio, modelando como as

funcionalidades se relacionam para garantir que o software faça tudo o que for necessário para atender às necessidades dos usuários.

É uma boa prática colocar os casos de uso do diagrama de caso de uso dentro de um retângulo, que irá representar os limites do software. Ou seja, o diagrama deve conter todos os casos de uso que contemplem as funcionalidades que o software deve executar, pois ele mostra a visão total do escopo do software. Os casos de uso, por serem o escopo do sistema, ficam dentro do retângulo; enquanto os atores, que interagem com o sistema, mas não fazem parte dele, ficam no lado de fora do retângulo.

3.1 Características de um diagrama de casos de uso

Conforme discutido anteriormente, após a identificação dos requisitos de um software, é preciso especificá-los, ou melhor, detalhá-los. Uma técnica comumente adotada para especificação dos requisitos é a modelagem de casos de uso (*use cases*).

As principais características de um diagrama de caso de uso são:

- ele pode ser consultado por qualquer *stakeholder*, pois utiliza um padrão fácil de ser compreendido mesmo por quem não é de TI;
- o seu foco é a definição do problema e não a solução computacional; por isso, ele representa os requisitos funcionais do software;
- o diagrama de caso de uso permite registrar de fato o que o sistema deve fazer e como as funcionalidades do software se relacionam entre si;
- por ser um modelo de análise, o diagrama de caso de uso é independente da abordagem de desenvolvimento;
- ele pode ser descrito com diversos níveis de detalhamento, facilitando a compressão no nível macro do software e também nos níveis de detalhe que permitam visualizar todo o relacionamento entre as funcionalidades identificadas para o software.

TEMA 4 – COMPONENTES DE UM DIAGRAMA DE CASO DE USO

Os componentes de um diagrama de caso de uso, também chamados de *elementos*, fornecem a identidade visual do modelo. Para entender um diagrama, é preciso entender a linguagem utilizada na sua construção. A

linguagem tomada como base é a linguagem UML, já estudada por nós. Vamos conhecer, então, cada um dos elementos de um diagrama de caso de uso.

4.1 Ator e caso de uso

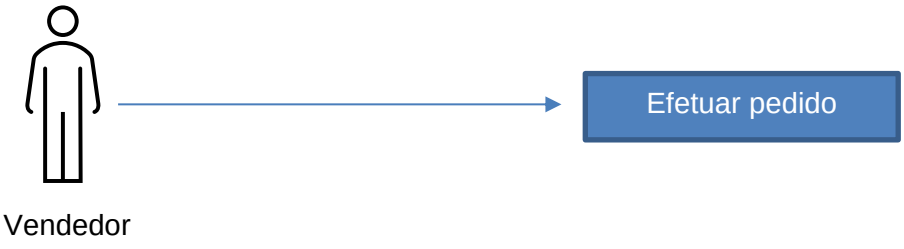
O ator representa o papel executado por uma entidade que interage com o sistema em questão. Um ator modela algo fora da fronteira do sistema e precisa trocar informações com esse sistema, como indivíduos e outros sistemas, ou seja, o ator interage com o sistema, mas ele não faz parte do sistema. Um usuário pode executar o papel de vários atores diferentes e um determinado ator pode ser representado por vários usuários. Dessa forma, é preciso esclarecer que ator não é o mesmo que usuário.

Reforçando o conceito:

- Um ator representa um papel exercido por um usuário ao interagir com determinado caso de uso.
- Usuários podem desempenhar mais de um papel no sistema.
- Em modelagem de software, como a UML é a linguagem utilizada, é preciso conhecer a notação, ou melhor: os símbolos que representam graficamente os elementos do diagrama de caso de uso.

Um ator é representado por um boneco e um rótulo com o nome do ator. Um ator é um usuário do sistema, que pode ser um usuário humano ou um outro sistema computacional.

Figura 3 – Ator



Os atores podem ser classificados em diferentes tipos, como:

- **Ator primário** – é o interessado que acessa o sistema para utilizar diretamente um serviço.
- **Ator secundário (ou ator de suporte)** – interage com o sistema fornecendo algum tipo de serviço ou informação. Por exemplo, um sensor de temperatura, um sistema de cartão de crédito ou um sistema de contabilidade. Os atores secundários são úteis para identificar o que o sistema deverá prover em termos de protocolo de troca de dados e interfaces externas.

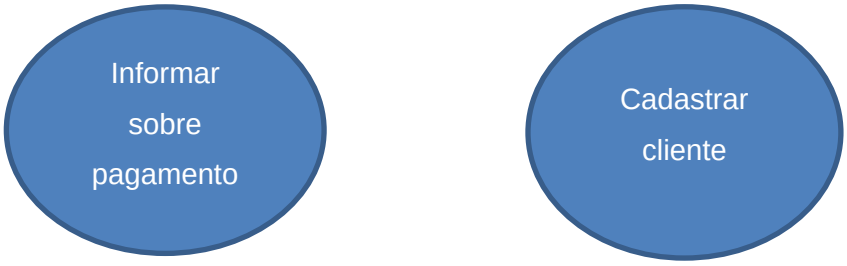
Os atores podem ainda pertencer a categorias diferentes, e algumas das mais comuns são:

- Usuários: usuários finais, administradores, gerentes, entre outros.
- Aplicações: processos individuais, rotinas automatizadas e sistemas de software externos ao sistema modelado.
- Dispositivos: sensores ou qualquer outro hardware que tenha interação com o sistema modelado.
- Eventos externos: controlador de tempo, escalonador de processos ou qualquer outro evento que dispare uma funcionalidade do sistema modelado.

Uma boa dica para identificar os atores é analisar as fontes e os destinos das informações a serem processadas, pois esses são atores em potencial. E também é importante analisar as áreas da organização que serão afetadas ou utilizarão o sistema em construção, igualmente atores em potencial. Algumas perguntas úteis para apoiar a identificação de possíveis atores são: que órgãos, empresas ou pessoas irão utilizar o sistema? Que outros sistemas irão se comunicar com o sistema a ser construído? Alguém deve ser informado de alguma ocorrência no sistema? Quem está interessado em um certo requisito funcional do sistema?

A representação gráfica do caso de uso é feita por meio de um ícone que é uma elipse contendo seu nome. É possível encontrar diagrama de caso de uso cujo nome do caso de uso seja apresentado abaixo da elipse. Um caso de uso define uma função do sistema.

Figura 4 – Representação de caso de uso



É muito importante iniciar a modelagem do diagrama de caso de uso separando as funcionalidades do sistema e levando em conta os seguintes itens:

- Para que cada funcionalidade ou grupo de funcionalidades seja atendido, deve haver um conjunto ou sequência de ações que tenham um objetivo bem definido e sejam detalhadas na descrição de caso de uso.
- Durante a execução dessa sequência de ações, o sistema interage com elementos externos a ele, que são os atores.
- É importante se concentrar inicialmente no fluxo principal e depois nos fluxos alternativos e de exceção. Isso garante a completude do escopo do caso de uso.
- Descreva **o que** o sistema fará, mas não especifique **como** isso será feito, pois esse não é o objeto da fase de modelagem do software. O **como** será modelado em outros diagramas, que estudaremos nas próximas aulas.

4.2 Relacionamentos

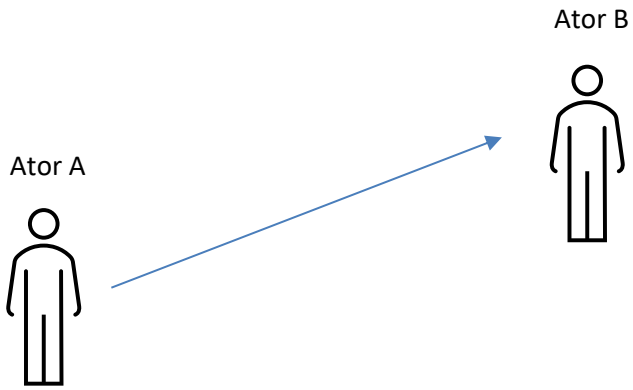
Os relacionamentos mostram a ligação entre os elementos de um diagrama de caso de uso, ou seja, a ligação dos atores com os casos de uso e dos casos de uso entre si. Os relacionamentos ajudam a compreender o funcionamento do software, demonstrando quais atores têm acesso a cada caso de uso e como os casos de uso se relacionam, apresentando a dinâmica de interação entre eles, focando no entendimento do funcionamento dos processos de negócio.

Vamos nos aprofundar nos diferentes tipos de relacionamento que podem ser encontrados em um diagrama de caso de uso.

4.2.1 Relacionamento entre atores: generalização

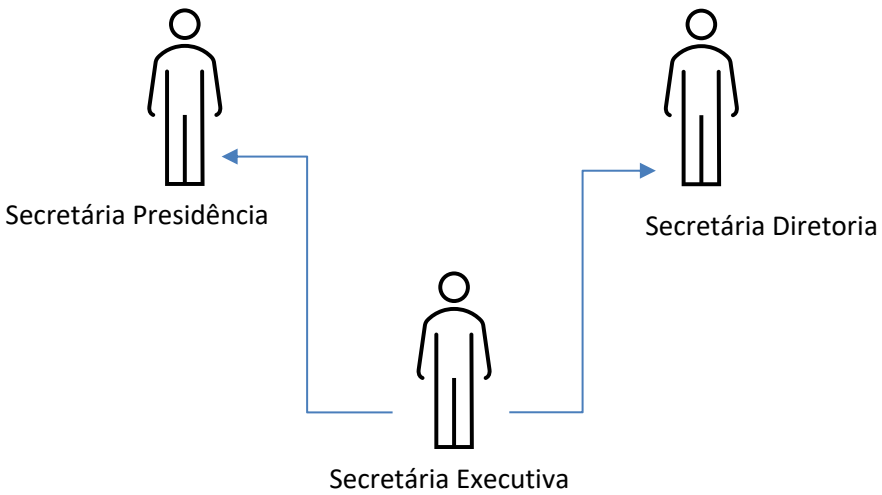
O único relacionamento possível entre os atores é a **generalização**. Esse relacionamento é identificado quando temos dois atores semelhantes, mas com um deles realizando algo a mais que o outro. A generalização é usada para identificar funcionalidades comuns entre atores e pode ser sempre lida como *[algo] é um tipo de*. Ela deve ser identificada e validada com cuidado, para não modelar relacionamentos desnecessários a um determinado ator ou, mesmo, não demonstrar relacionamentos que não devem ocorrer para garantir o bom funcionamento do software. Os relacionamentos precisam representar exatamente as ligações corretas para o software, nem mais e nem menos.

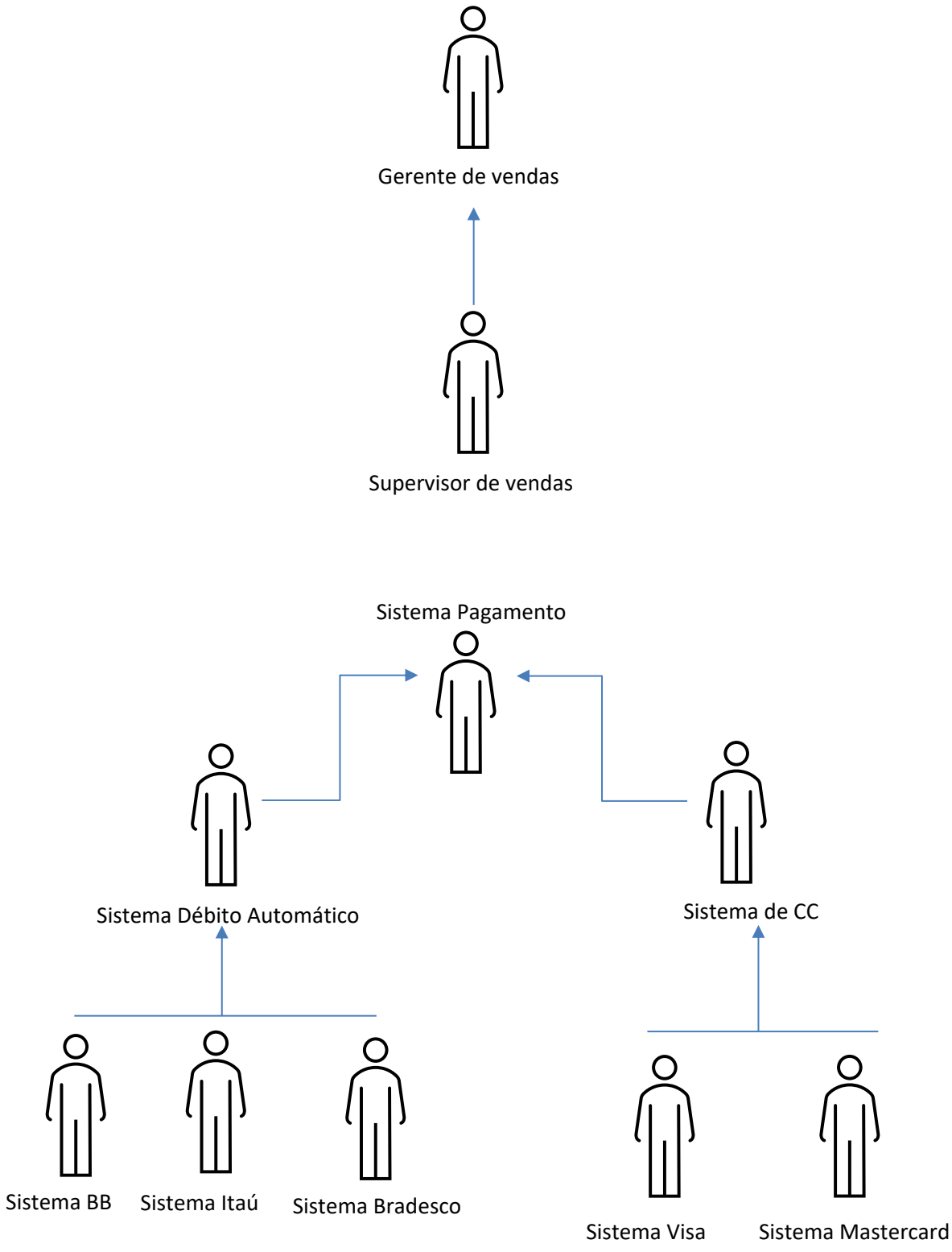
Figura 5 – Generalização



Na Figura 5, os casos de uso relacionados com o ator B são também automaticamente relacionados com os casos de uso do ator A. E o ator A tem seus próprios casos de uso, aos quais o ator B não tem acesso.

Figura 6 – Exemplos de generalização



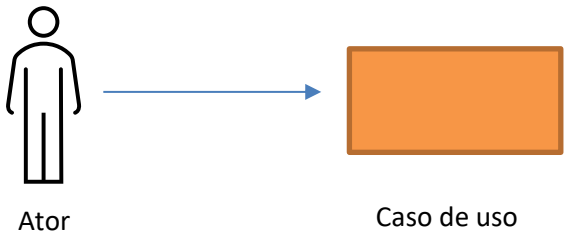


Nos três exemplos apresentados na Figura 6, não importa a quantidade de níveis da hierarquia, o conceito se mantém o mesmo, ou seja, os usuários dos níveis mais baixos herdam os casos de uso do nível superior, enquanto os

casos de uso de nível superior possuem seus próprios casos de uso. Dessa forma, é possível reaproveitar casos de uso que podem ser acessados por mais de um ator, permitindo ainda que alguns atores tenham acesso a outros casos de uso, dependendo do uso que terá o sistema em desenvolvimento.

4.2.2 Relacionamento entre um ator e um caso de uso: associação

Figura 7 – Relacionamento de associação



O relacionamento de associação define uma funcionalidade do sistema do ponto de vista do usuário, mostrando visualmente a quais casos de uso cada ator tem acesso. A associação representa a interação do ator com o caso de uso, por meio de envio e recebimento de mensagens. Associações são representadas por uma linha sólida, ligando o ator ao caso de uso. Se o ator inicia um caso de uso, ele pode se comunicar com vários atores depois. As associações servem para mostrar quais atores se comunicam com o caso de uso em questão.

A direção do relacionamento de associação também é importante para compreender o funcionamento dos casos de uso. Dessa forma, é preciso compreender os seguintes pontos:

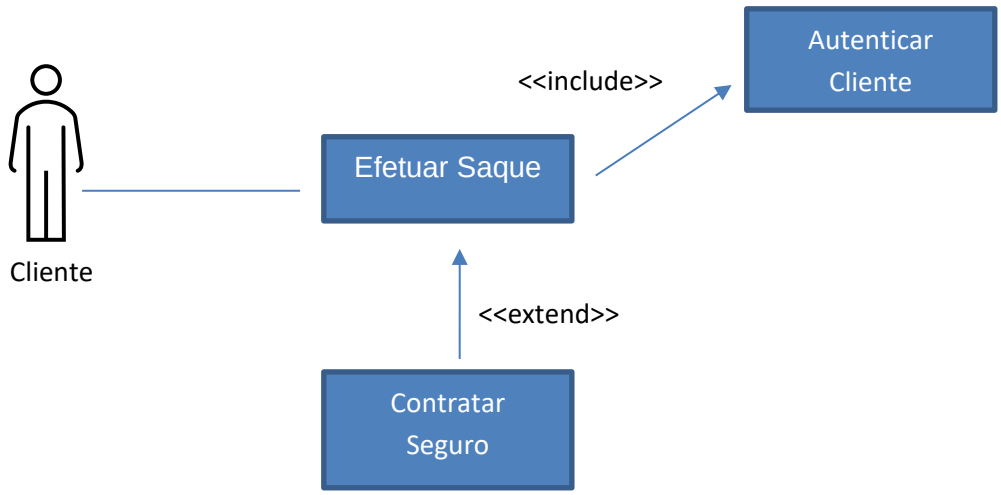
- Cada ator de uma associação tem uma propriedade de navegabilidade, que indica quem inicia a comunicação na interação.
- A direção é representada no diagrama por uma seta: se a seta apontar para um caso de uso, o ator inicia a interação; se a seta apontar para o ator, o sistema inicia a interação.
- A navegabilidade de duas direções é mostrada por uma linha sem setas. Também é possível usar setas bidirecionais.

4.2.3 Relacionamento entre casos de uso

O relacionamento entre casos de uso pode ser:

- ***Include***: um relacionamento *include* de um caso de uso A para um caso de uso B indica que B é essencial para o comportamento de A. Pode ser dito também que *B is_part_of A*.
- ***Extend***: um relacionamento *extend* de um caso de uso B para um caso de uso A indica que o caso de uso B pode ser acrescentado para descrever o comportamento de A, mas não lhe é essencial. Nesse sentido, o ponto de extensão, em um caso de uso, é uma indicação de que outros casos de uso poderão ser adicionados a ele. Quando o caso de uso for invocado, ele verificará se suas extensões devem ou não ser invocadas.

Figura 8 – Exemplo de relacionamento entre casos de uso



Analisando o exemplo da Figura 8, no caso de uso de saque em um caixa eletrônico, percebemos que autenticar o cliente é uma ação obrigatória; por isso, o caso de uso de efetuar saque faz um *include* da opção de autenticar cliente. Já o caso de uso de contratar um seguro é opcional, dependendo do aceite do cliente, por isso ele é chamado com um relacionamento de *extend*, pois carece de uma ação direta do usuário.

Resumindo o que foi estudado neste tema, para construir um diagrama de caso de uso que contemple todo o sistema que se está analisando é preciso ter

identificados todos os casos de uso e os atores relacionados. Dessa forma, é possível modelar, por meio do diagrama, o relacionamento dos atores com os casos de uso e dos casos de uso entre si, representando visualmente as fronteiras do sistema.

TEMA 5 – ANALISANDO UM EXEMPLO DE DIAGRAMA DE CASO DE USO

Vamos, agora, voltar ao nosso estudo de caso, analisando como seria a modelagem de diagrama de casos de uso para o sistema proposto. Vamos relembrar nosso estudo de caso, que é o seguinte: fomos contratados por nosso cliente para modelar o processo de vendas on-line de livros. Nosso cliente tem uma livraria virtual, que vende produtos diretamente aos seus clientes, em um site próprio. O diferencial dessa livraria é ter um estoque próprio, o que garante uma entrega mais rápida de livros a seus clientes; aceitar vários tipos de pagamento, como cartão de crédito, cartão de débito e boleto bancário; administrar um programa de fidelidade que fornece desconto de 10% aos clientes que compram R$ 500,00, ao longo de 1 ano.

Reforçando o que já foi dito nas aulas anteriores, não existe uma única forma de modelar esse estudo de caso. Sendo assim, o diagrama de caso de uso que construiremos é apenas uma forma de modelar o sistema proposto. O estudo de caso proposto, nessa solução, possui os seguintes casos de uso:

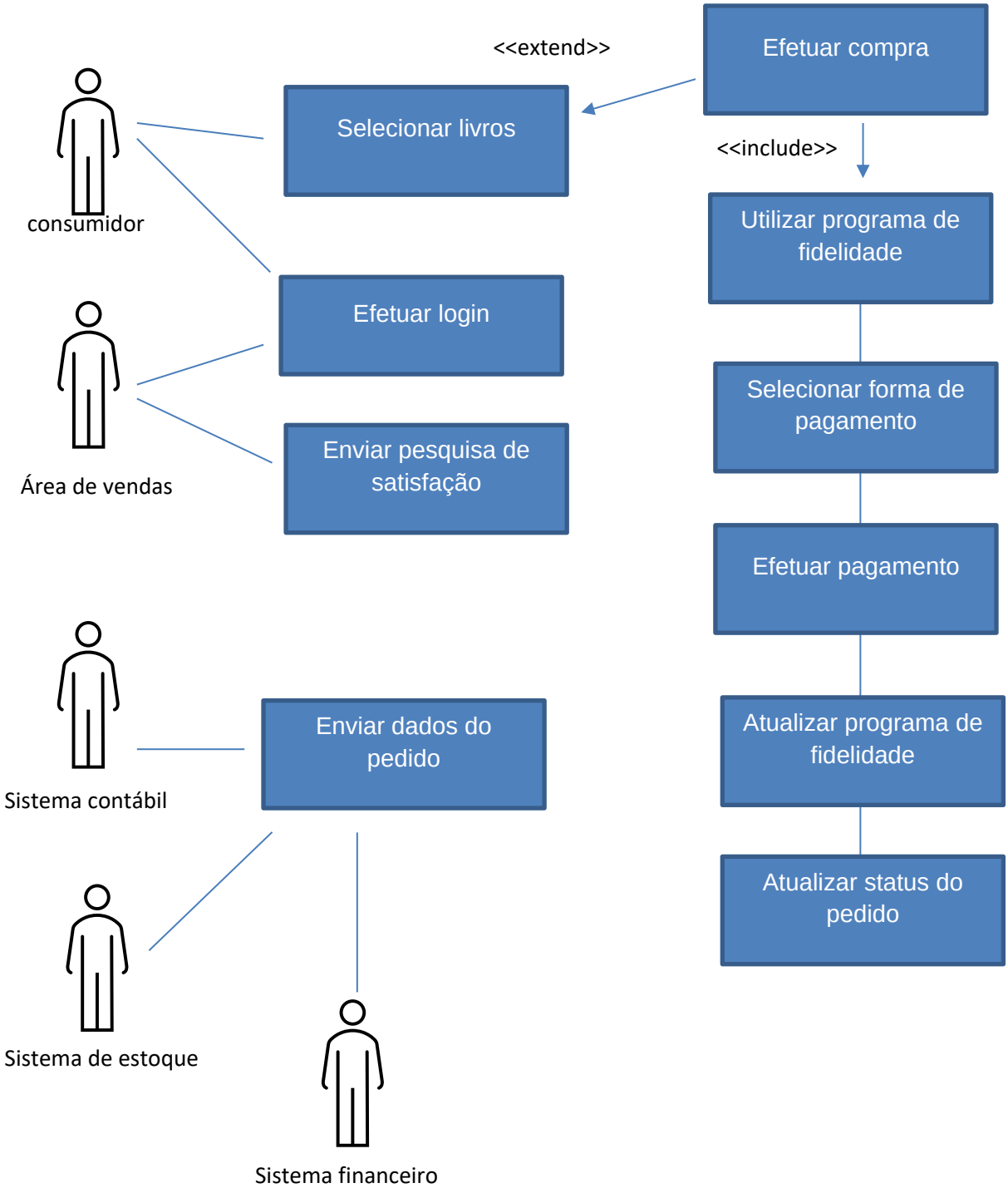
- fazer login;
- efetuar compra;
- selecionar livros;
- selecionar forma de pagamento;
- efetuar pagamento;
- utilizar programa de fidelidade;
- atualizar status do pedido;
- atualizar programa de fidelidade;
- enviar dados do pedido;
- enviar pesquisa de satisfação.

Os atores identificados no levantamento de requisitos são:

- consumidor;

- área de vendas;
- sistema de estoque;
- sistema financeiro;
- sistema contábil.

Figura 9 – Estudo de caso: diagrama de caso de uso



O caso de uso de selecionar livros *extend* o caso de uso de efetuar compra, pois depende da vontade do consumidor, ou seja, efetuar compra não é um caso de uso obrigatório, pois o consumidor pode apenas navegar pelo site

e selecionar alguns livros sem realmente iniciar e efetivar uma compra. E o caso de uso de efetuar compra, por outro lado, obrigatoriamente inclui o caso de uso de utilizar o programa de fidelidade, pois, pela descrição do caso de uso estudado, sempre que o cliente efetuar uma compra, será preciso verificar se ele é elegível ao desconto dado pelo programa de fidelização de clientes da loja, conforme explicado anteriormente.

O diagrama de caso de uso complementa o entendimento sobre o funcionamento do processo de negócio, mostrando o escopo a ser atendido pelo sistema. Com a descrição do caso de uso, é possível entender o funcionamento de cada um dos requisitos do sistema; e, com o diagrama de caso de uso, é possível entender como os casos de uso se relacionam e qual ator tem acesso a cada um dos casos de uso, ou seja, a visão de análise das funcionalidades do software se completa.

Os demais casos de uso são chamados internamente pelo próprio sistema, sem interação com nenhum ator, portanto, possuem um relacionamento de associação simples. Os atores do sistema, ou seja, o sistema de estoque, o sistema contábil e o sistema financeiro, interagem com o mesmo caso de uso de enviar dados do pedido e atualizam seus sistemas conforme a necessidade. Ou seja, o sistema do estudo de caso apenas informa os dados sobre o pedido concluído e cada sistema atualiza o que precisa para manter seus próprios dados.

O sistema de estoque utiliza as informações enviadas para manter o controle de estoque atualizado, conforme os livros vendidos. O sistema contábil atualiza seus bancos de dados para garantir a correta demonstração financeira dos resultados da empresa e o sistema financeiro atualiza seus bancos de dados para mostrar as vendas realizadas e o fluxo de caixa, conforme forma de pagamento selecionada pelo consumidor. É importante perceber que todo o funcionamento do sistema vai ficando mais claro conforme a modelagem vai avançando. Já é possível compreender como o software vai funcionar, logicamente.

FINALIZANDO

Chegamos ao final da nossa aula e esperamos que os conceitos vistos aqui tenham ficado claros. Discutimos sobre a UML e os diagramas mais comuns utilizados na modelagem de um software. Cada diagrama atende a um objetivo

específico, e todos os diagramas construídos devem estar coerentes entre si e agregar valor ao entendimento completo do sistema a ser desenvolvido.

Relembramos também noções sobre orientação a objetos, que é um processo conceitual independente de uma linguagem de programação, pois tem como foco visualizar o domínio do problema a ser automatizado como uma coleção de objetos e métodos associados. A UML modela softwares orientados a objetos, por isso não é possível compreender a UML sem conhecer os conceitos da orientação a objetos. Os diagramas da UML utilizam vários conceitos relacionados com objetos e suas características.

E o foco principal desta aula foi nos aprofundarmos no entendimento do objetivo e na construção de um diagrama de caso de uso. Para isso, foram apresentados os elementos que compõem um diagrama e como eles se relacionam, modelando o funcionamento dos requisitos do sistema e complementando a análise de um software que foi iniciada com o levantamento dos requisitos e com a descrição dos casos de uso. Os elementos estudados foram os atores que interagem com o sistema, os casos de uso identificados e que implementam os requisitos levantados com os usuários e, por fim, o relacionamento dos atores com os casos de uso e dos casos de uso entre si, mostrando de forma visual a lógica de funcionamento dos casos de uso do sistema. O diagrama de caso de uso deve estar coerente com a descrição dos casos de uso, pois os dois documentos de análise do software servem como base para se iniciar a modelagem do projeto de software, portanto, eles precisam ajudar no entendimento e não gerar mais dúvidas nos analistas de sistemas.

Depois de todo o conteúdo discutido durante esta aula, vamos continuar navegando na análise de sistema de um projeto de software. Encontramo-nos nas próximas aulas para falarmos sobre como modelar a parte física de um software, identificando as classes ou objetos e modelando um diagrama de classe.

REFERÊNCIAS

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **UML**: guia do usuário. Rio de Janeiro: Editora Campus, 2005.

PRESSMAN, R. S.; MAXIM, B. R. **Engenharia de software**: uma abordagem profissional. 8. ed. [S.l.]: McGraw-Hill Education, 2016.