De que se trata o artigo:

Apresentação dos conceitos relacionados ao comportamento dinâmico do sistema, aqui representados pelos diagramas de casos de uso, de sequência e de atividades.

Para que serve:

Visa mostrar ao leitor os conceitos fundamentais sobre os diagramas de casos de uso, de sequência e de atividades para que ele possa criar modelos de sistemas usando esses diagramas.

Em que situação o tema é útil:

Ao projetar um software de grande porte, precisamos conhecer a fundo os diagramas UML. Neste artigo, abordamos os diagramas UML capazes de modelar o comportamento de um sistema.

Introdução:

Neste artigo, aprenderemos os conceitos relacionados aos diagramas de casos de uso, de sequência e de atividades. Veremos como eles podem ser usados no processo de desenvolvimento de software. Os casos de uso são o ponto de partida do processo e representam as funcionalidades do sistema na visão dos atores. Os casos de uso modelam o que o sistema deve fazer. Para criar artefatos que representam como o software deve fazer para alcançar esses objetivos existem os diagramas de sequência e os diagramas de atividade. Os diagramas de sequência modelam a interação entre os objetos de um único caso de uso, enfatizando a ordem em que essas interações ocorrem. Por sua vez, os diagramas de atividades representam a visão de processo de um único caso de uso ou a modelagem da lógica detalhada de uma regra de negócio.

O levantamento de requisitos, possivelmente, é a tarefa mais difícil de todo o processo de desenvolvimento de um software. Em geral, os requisitos são identificados a partir de questionários, entrevistas com os usuários ou ainda análise de documentos usados no sistema corrente, entre outras técnicas. E isto em geral conduz a documentos redigidos pelos engenheiros de software, que descrevem essas funcionalidades. A descrição das funcionalidades comumente é feita em linguagem natural – linguagem usada para escrever este artigo. E não podemos negar que esse tipo de linguagem permite várias interpretações, pois se sabe que escrever algo de maneira que todos entendam o que se está querendo dizer é um grande desafio. Por outro lado, usar desenhos em vez de linguagem natural também pode não ser uma boa solução, visto que um desenho que faz sentido para uma pessoa pode não fazer sentido para outra.

Então, como transformar estes requisitos em um formato que os *stakeholders* entendam, sem perder detalhes que são importantes para o processo de desenvolvimento? A resposta está nos casos de uso. Os casos de uso são o centro de tudo na UML, o ponto de partida para a criação de um sistema. Este artefato da UML é o primeiro tema a ser exposto nesta matéria.

Em seguida, serão estudados os diagramas de sequência, que fazem parte de um grupo da UML denominado diagramas de interação. Neste grupo estão também os diagramas de comunicação e de tempo. Os diagramas de interação modelam as interações entre as partes – os objetos, modelados no diagrama de classes – que compõem o sistema.

Será estudado mais adiante que os casos de uso modelam *o que* o sistema deve fazer, mas não especificam *como* o sistema irá cumprir esses objetivos. Para este fim existem os diagramas de atividade. Tais diagramas normalmente são usados para modelar os processos, e são detalhados na parte final do artigo.

Diagrama de casos de uso

O diagrama de casos de uso é bastante usado no processo de levantamento de requisitos e análise. À medida que os requisitos são coletados, uma visão geral das funções e características do sistema começa a se materializar. No entanto, é difícil avançar para atividades mais técnicas de engenharia de software até que a equipe de software entenda como essas funções e características serão usadas por diferentes usuários finais.

Segundo McLaughlin, Pollice e West [5], um requisito é uma necessidade única que detalha o que um produto ou serviço em particular deve ser ou fazer. É frequentemente utilizado na engenharia de sistemas ou engenharia de software para identificar os atributos, características ou funcionalidades de um sistema que atendam às necessidades dos usuários.

Para que se consiga um melhor entendimento das necessidades de um software, desenvolvedores e usuários podem criar um conjunto de cenários que identifiquem uma maneira de se usar o sistema a ser construído. Esses cenários, conhecidos como casos de uso, fornecem uma descrição de como o sistema será usado.

Segundo Cockburn [4], um caso de uso se refere a um contrato que descreve o comportamento do sistema sob várias condições em que este responde a uma solicitação de um dos seus interessados. Em essência, um caso de uso conta uma história sobre a maneira como um usuário final interage com o sistema sob um conjunto específico de circunstâncias. A história pode ser um texto narrativo, um delineamento das tarefas e/ou interações ou uma representação em diagrama. Independentemente de sua forma, um caso de uso descreve o sistema do ponto de vista do usuário.

O primeiro passo ao se escrever um caso de uso é definir o conjunto de atores que estarão envolvidos na história. Atores representam papéis que pessoas (ou dispositivos) desempenham quando o sistema está em operação. Definindo mais formalmente, um ator é qualquer coisa que se comunique – e interaja – com o sistema e que seja externo ao sistema em si. Durante essa interação com o sistema pode-se notar que cada ator tem um ou mais objetivos.

É importante notar que um ator e um usuário final não são necessariamente a mesma entidade. O usuário típico pode desempenhar vários papéis diferentes quando usa um sistema, enquanto o ator representa uma classe de entidades externas que desempenham apenas um papel no contexto do caso de uso. Como exemplo, pense num sistema em que autores de artigos recebam dinheiro pelas suas publicações. Nesse sistema, cada usuário pode ler artigos e/ou submeter artigos para publicação. Com isso, cada usuário pode desempenhar dois papéis distintos (leitor e escritor). Cada um desses papéis é modelado no diagrama de casos de uso como atores distintos, mas que podem representar as ações de um mesmo usuário. Isso evidencia a diferença entre atores e usuários.

Agora que vimos uma abordagem inicial sobre casos de uso, apresentaremos como devemos escrevê-los. Para iniciar a escrita de um caso de uso precisamos das informações adquiridas durante o levantamento de requisitos. Reuniões de coleta de requisitos e outros mecanismos de engenharia de requisitos são usados para identificar os interessados (os *stakeholders*), definir o escopo do problema, especificar os objetivos operacionais, esboçar todos os requisitos funcionais conhecidos e descrever os dados que serão manipulados pelo sistema.

Para iniciar o desenvolvimento de um conjunto de casos de uso, as funções ou atividades realizadas por um ator específico são listadas. Essas podem ser obtidas de uma lista de funções necessárias para o sistema, por meio de conversas com os clientes ou usuários finais, ou por uma avaliação dos diagramas de atividade desenvolvidos como parte da modelagem de análise.

Os diagramas de casos de uso têm uma importância muito grande no processo de desenvolvimento de software. Com ele, há uma melhor comunicação com os clientes, mostrando o comportamento esperado do software sem se preocupar como a implementação será feita.

O diagrama de casos de uso descreve os requisitos funcionais do sistema. Tal diagrama irá servir de base para o projeto do software. É um diagrama muito utilizado atualmente no processo de desenvolvimento de sistemas, bem como na criação de testes. Além disso, este diagrama tem o propósito de representar o escopo do sistema, informando suas funcionalidades e eventuais restrições.

Veremos agora os elementos e mecanismos do diagrama de casos de uso. Veremos também os conceitos de relacionamento entre casos de uso – extensão e inclusão.

Na notação UML um ator é representado pelo *stickman,* ou por uma caixa estereotipada, como pode ser visto na **Figura 1**. Além disso, tanto em uma forma quanto na outra, a representação deve receber um rótulo que identifica o ator. Por sua vez, o caso de uso é representado por uma figura elíptica. Uma descrição interna à figura descreve a interação que o caso de uso representa. Na **Figura 2** mostramos a representação de um caso de uso.

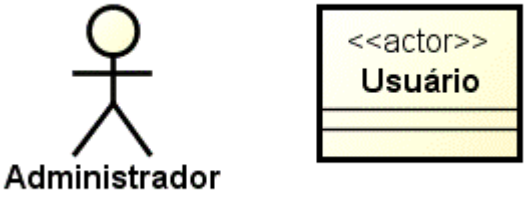

Figura 1. Representação de ator.

Figura 2. Representação de caso de uso.

A associação, no caso do diagrama de casos de uso, indica que um ator poderá executar um determinado caso de uso. Cada ator pode estar associado a vários casos de uso, o que não indica que ele é obrigado a realizar todos eles. A **Figura 3** nos mostra um diagrama de casos de uso, no qual estão representados os conceitos de ator e associações entre este e três casos de uso.

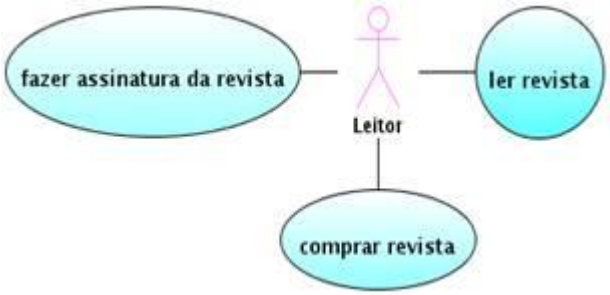

Figura 3. Representação de atores e associações.

A **Figura 3** exemplifica os casos de uso de um leitor: *ler revista*, *comprar revista* e *fazer assinatura da revista*. Vale ressaltar que não há obrigatoriedade de o leitor executar todos os três casos de uso. Além disso, eles podem ser subdivididos em outros casos de uso. Os segmentos de reta ligando o ator e os casos de uso representam o conceito de associação.

Os diagramas de caso são um bom ponto de partida na análise de um sistema, mas não são suficientes para que os desenvolvedores conheçam os detalhes de como os objetivos do software serão atingidos. A melhor maneira de expressar essas informações é na forma de uma descrição textual. Assim, cada caso de uso deve ser acompanhado de uma descrição.

No entanto, não existe uma definição formal proposta pela UML de como deve ser criada a descrição de um caso de uso. Uma proposta nesse sentido é apresentada na **Tabela 1**, onde são relacionados os itens que podem fazer parte dessa descrição, adaptados do que é sugerido por Pressman [3].

Item	**Significado**
Caso de uso	Nome do caso de uso
Ator principal	O ator principal que participa deste caso de uso
Meta no contexto	O papel deste caso de uso no sistema e porque ele é importante
Pré-condições	Representam o estado em que o sistema deve estar antes que o caso de uso possa acontecer
Disparo	Qual o evento disparado pelo ator que irá provocar a execução do caso de uso
Fluxo principal	A descrição dos passos necessários para execução normal do caso de uso
Exceções	A descrição, se houver, de passos alternativos àqueles descritos no fluxo principal

Tabela 1. Proposta de descrição de casos de uso.

Para exemplificar uma descrição vamos supor o caso em que um cliente faz o pedido de um produto em um site de comércio eletrônico. Neste exemplo, *cliente* é o ator, e *fazer pedido* é o caso de uso. Uma possível descrição seria como a que mostramos na **Tabela 2**.

Caso de uso	Fazer pedido
Ator principal	Cliente
Meta no contexto	Um cliente previamente cadastrado no site faz o pedido de um produto
Pré-condições	O cliente precisa estar cadastrado no site
Disparo	O cliente clica no link *Comprar* ao lado do produto escolhido

Fluxo principal	1. O carrinho de compras apresenta o produto selecionado e o valor parcial do pedido; 2. O cliente informa o CEP e solicita o cálculo do frete; 3. O frete é calculado e o valor total do pedido é mostrado; 4. O cliente segue para o próximo passo do pedido; 5. É solicitado o login e a senha do cliente. Se o cliente não possuir um login, ele pode clicar em um link para fazer o cadastro; 6. O sistema solicita a confirmação dos dados para entrega; 7. O cliente confirma e segue para o próximo passo; 8. O cliente informa os dados do seu cartão de crédito; 9. O sistema confirma o pedido e envia um e-mail para o cliente.
Exceções	1. O produto não está disponível no estoque; 2. O cartão de crédito do cliente não foi autorizado.

Tabela 2. Descrição do caso de uso *fazer pedido*.

Quando estamos fazendo essas descrições, pode-se observar, por exemplo, que existem passos similares em diferentes casos de uso. Por exemplo, em um sistema de caixa eletrônico, ambos os casos de uso, *solicitar extrato* e *efetuar saque* possuem passos onde é solicitada a senha do cartão. Por outro lado, existem situações em que os casos de uso funcionam de várias maneiras diferentes ou tem casos especiais. Por exemplo, o caso de uso *efetuar pagamento* pode ter os casos particulares *pagamento em dinheiro* e *pagamento com cartão de crédito*. Além disso, um caso de uso pode ter múltiplos fluxos de execução. Para tornar mais clara a análise do sistema, seria importante que essas situações opcionais fossem representadas nos diagramas de caso de uso. Essa possibilidade existe e chamamos de relacionamento entre casos de uso.

O primeiro desses relacionamentos é a extensão, a qual representa uma condição para que um caso de uso seja executado a partir de outro, ou seja, um caso de uso será executado se uma dada condição for verdadeira. Esse mecanismo é representado por uma seta tracejada, como apresenta a **Figura 4**.

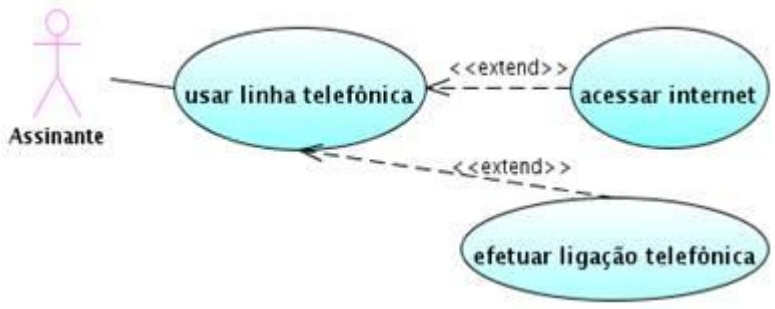

Figura 4. Representação do mecanismo de extensão.

Veremos agora como interpretar o diagrama da **Figura 4**. O Assinante pode executar apenas um caso de uso, que é usar linha telefônica. Há duas extensões para esse caso de uso: *efetuar ligação telefônica* e *acessar internet*. Dependendo de uma condição – a necessidade do assinante em determinado momento – um dos dois casos de uso será executado após a execução do caso de uso *usar linha telefônica*. O objetivo da extensão é representar uma condição para a realização de um caso de uso a partir de outro.

Diferentemente do relacionamento de extensão, a inclusão representa que há uma obrigatoriedade na realização de um caso de uso a partir de outro. O relacionamento de inclusão obriga que um caso de uso seja executado durante a execução de outro caso de uso. Esse mecanismo é representado por uma seta tracejada, mas com sentido contrário ao do mecanismo de extensão, conforme mostra a **Figura 5**.

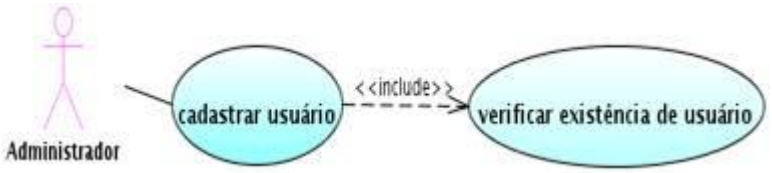

Figura 5. Representação do mecanismo de inclusão.

Nesse caso, o ator **Administrador** possui apenas um caso de uso, *cadastrar usuário*. Entretanto, há um mecanismo de inclusão entre cadastrar usuário e verificar existência de usuário. Isso indica que há uma obrigatoriedade de se executar o caso de uso verificar existência de usuário durante a execução de cadastrar usuário.

Um ponto importante sobre os relacionamentos de extensão e inclusão é quanto à sua representação. Suponha que um caso de uso denominado por **A** seja sempre executado durante a execução de um caso de uso **B**. O mecanismo existente entre **A** e **B** é de inclusão, pois há uma obrigatoriedade de execução. Neste caso, a seta partiria de **B** e apontaria para **A**. Se houvesse uma condição para que **A** fosse executado, haveria um mecanismo de extensão entre **A** e **B**. Neste caso, a seta partiria de **A** e apontaria para **B**.

Vale ressaltar que um caso de uso pode ser subdividido em diversos casos de uso. Um desenvolvedor pode estar mais interessado no detalhamento do diagrama para poder implementar o sistema com mais facilidade, enquanto que um cliente pode estar mais interessado numa visão mais geral sobre o comportamento do software. Deste modo, o diagrama a ser criado deve se adequar a uma determinada situação, visto que diferentes envolvidos no sistema, os *stakeholders*, terão preocupações distintas com o mesmo. A **Figura 6** apresenta um diagrama de casos de uso que utiliza todos os conceitos citados.

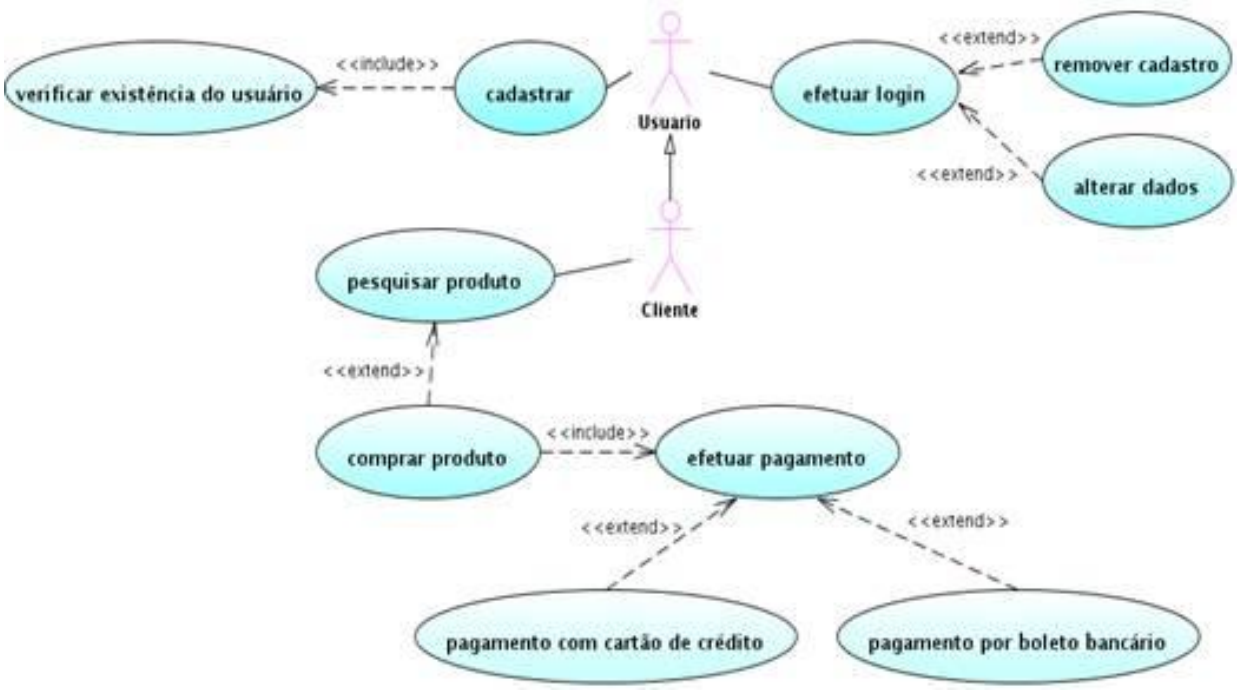

Figura 6. Diagrama de casos de uso.

Na **Figura 6** há uma generalização entre **Cliente** e **Usuario**, o que indica que **Cliente** possui todos os casos de uso de **Usuario** e mais os casos de uso específicos para os clientes do sistema.

Um ponto importante para discorrer é sobre o caso de uso efetuar pagamento. Note que há um mecanismo de extensão entre os casos de uso *pagamento com cartão de crédito* e *efetuar pagamento*. Outra solução, neste caso, seria modelar uma herança entre esses casos de uso no lugar da extensão.

Herança de caso de uso é quando queremos mostrar que um caso de uso é uma especialização de outro. Para representar a herança, usa-se a seta generalização para conectar o caso de uso mais geral àquele mais específico, como mostramos na **Figura 7**.

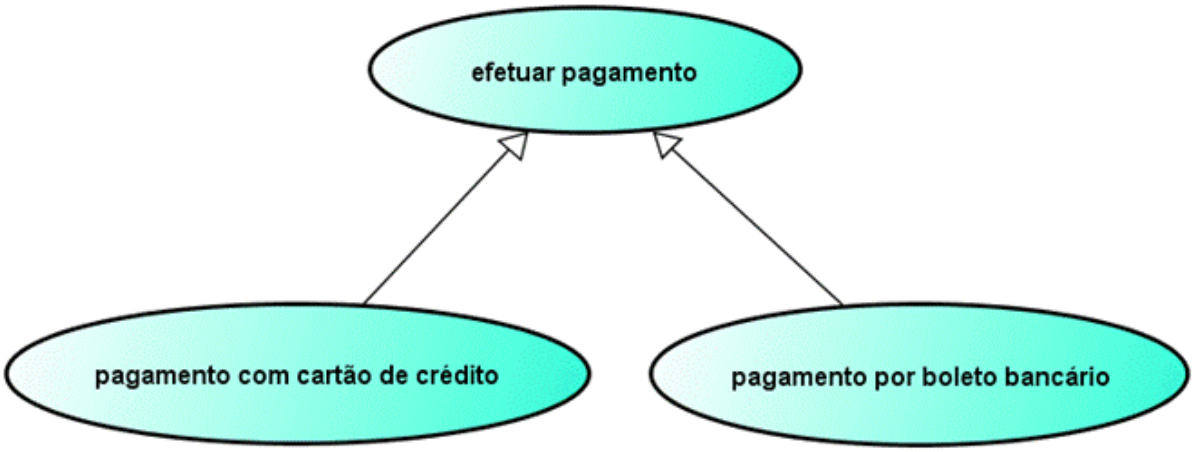

Figura 7. Herança de casos de uso

A escolha de um modelo em relação ao outro – extensão ou herança – deriva-se da experiência do analista. Mesmo que o modelo elaborado esteja correto, isso não indica que ele seja o mais adequado para a situação em questão. Veremos agora a distinção entre essas duas formas de modelagem através dos códigos das **Listagens 1** e **2**.

A **Listagem 1** apresenta a implementação do caso de uso *efetuar pagamento* da **Figura 6**, enquanto que a **Listagem 2** apresenta a implementação do mesmo caso de uso, porém, alterando os mecanismos de extensão por herança.

Listagem 1. Implementação do caso de uso *efetuar pagamento* usando extensão.

```
public class Pagamento {
      public void efetuarPagamento(int opcao){
            switch(opcao){
                  case 0:
                        pagamentoCartaoCredito();
                              break;
                  case 1:
                        pagamentoBoletoBancario();
                              break;
                  default:
                              break;
            }
      }

      public void pagamentoCartaoCredito(){
            System.out.println("pagando com cartão de
      crédito...");
      }
      public void pagamento Boleto Bancário(){
            System.out.println("pagando com boleto bancário...");
      }
}

      public class Teste Pagamento Extensão {
            public static void main(String[] args){
                  int cartaoCredito = 0;
                  int boletoBancario = 1;
                  Pagamento pagamento = new Pagamento();
                  pagamento.efetuar Pagamento(cartaoCredito);
                  pagamento.efetuarPagamento(boletoBancario);
      }
}
```

Listagem 2. Implementação do caso de uso *efetuar pagamento* usando herança.

```
public interface Pagamento {
	void efetuarPagamento();
	}
public class PagamentoBoletoBancario implements Pagamento{
	public void efetuarPagamento(){
		System.out.println("pagamento com boleto
	bancário...");
		}
	}
public class PagamentoCartaoCredito implements Pagamento{
	public void efetuarPagamento(){
		System.out.println("pagamento com cartão de
	crédito...");
	}
}
public class SistemaPagamento {
	public static void efetuarPagamento(Pagamento pagamento){
		pagamento.efetuarPagamento();
	}
}
public class TestePagamentoHeranca {
	public static void main(String[] args){
		Pagamento pagamentoCartaoCredito = new
		PagamentoCartaoCredito(); Pagamento
		pagamentoBoletoBancario = new
		PagamentoBoletoBancario();
		SistemaPagamento.efetuarPagamento(pagamentoCartaoCred
	ito);
		SistemaPagamento.efetuarPagamento(pagamentoBoletoBanc
	ario);
	}
}
```

Analisando o código da **Listagem 1**, percebemos que há uma classe gerenciadora que executa os métodos de pagamento. Há uma instrução switch para verificar o tipo de pagamento a ser utilizado. Para cada tipo de pagamento, devemos acrescentar uma instrução case dentro do switch e implementar um método para satisfazer essa nova requisição.

Ao analisar o código da **Listagem 2**, percebemos que eles fazem uso do polimorfismo, explicado nos artigo anterior. Para cada tipo de pagamento, cria-se uma nova classe que implemente a interface **Pagamento**. Essa modelagem faz com que a classe **SistemaPagamento** não necessite de alterações com o aumento do número dos tipos de pagamento.

Suponhamos que dois programadores distintos estejam desenvolvendo um sistema que possua um caso de uso *efetuar pagamento*, de acordo com o da **Figura 6**. O programador A opta pelo código da **Listagem 1**, enquanto que o programador B opta

pela **Listagem 2**. Uma discussão começa sobre a maneira de implementar o caso de uso da melhor forma.

O programador A argumenta a simplicidade em se criar os códigos da **Listagem 1** e o fato de não precisar conhecer o paradigma orientado a objetos para se construir um sistema simples. O programador B, especialista em orientação a objetos, argumenta que a **Listagem 1** apresenta uma limitação muito grande ao se trabalhar em equipe.

O programador B percebe que todos os programadores teriam que ter conhecimento de todo o código da classe **Pagamento** e, a cada novo tipo de pagamento a ser implementado, seria necessário colocar uma nova instrução case dentro da instrução switch. Como todos os programadores teriam que fazer alterações no mesmo arquivo e teriam que saber em que ponto do arquivo cada programador está alterando, o rendimento total da equipe estaria comprometido.

Ao analisar a **Listagem 2**, o programador B percebe também que o problema existente na **Listagem 1** não ocorre mais. Para cada novo tipo de pagamento a ser implementado, cria-se uma nova classe que implemente a interface **Pagamento**. Se cada programador fosse responsável por implementar um tipo de pagamento, eles poderiam desenvolver as novas classes de forma independente, ou seja, nenhum programador precisaria ter conhecimento sobre o trabalho do restante da equipe. Essa estratégia possibilita maior escalabilidade da aplicação e melhora sensivelmente a produtividade dos desenvolvedores.

Nesse caso, o polimorfismo auxilia o desenvolvimento de software, pois as novas funcionalidades do mesmo poderiam ser desenvolvidas independentemente. Além disso, o uso do polimorfismo também aumentou a coesão do software e diminuiu o acoplamento do mesmo.

Diagrama de sequência

Os diagramas de sequência e os de colaboração fazem parte de um grupo da UML chamado diagramas de interação. Os diagramas de interação representam como os objetos colaboram entre si para atingir um objetivo dentro de um único caso de uso.

Neste tópico serão estudados apenas os diagramas de sequência, considerado o artefato mais popular da UML para modelagem dinâmica.

Um diagrama de sequência mostra uma interação, isto é, uma sequência de mensagens trocadas entre vários objetos em um determinado contexto. É útil para ilustrar ou validar uma lógica implementada, bem como mostrar o comportamento dinâmico de um software em um determinado caso de uso.

Cada diagrama de sequência está associado a um diagrama de casos de uso, descrevendo a maneira como os objetos interagem para satisfazer aquele caso de uso. Esse diagrama dá ênfase especial na ordem e nos momentos nos quais as mensagens para os objetos são enviadas.

O diagrama de sequência também pode ser utilizado para identificar gargalos na modelagem de um aplicativo. Caso o aplicativo em desenvolvimento exija uma comunicação entre diferentes camadas de software, o diagrama de sequência pode verificar se há uma comunicação excessiva entre as diversas camadas e se isso pode comprometer o bom desempenho do aplicativo como um todo. Com isso, o diagrama de sequência pode ser útil ao se elaborar a arquitetura de um software em camadas.

No diagrama de sequência, cada objeto é representado por uma caixa acima de uma linha vertical tracejada, a qual é conhecida como linha de vida dos objetos. Tal linha representa o período de tempo em que os objetos estão na memória. Os objetos se comunicam através do envio de mensagens, as quais podem ser classificadas em síncrona ou assíncrona.

Nas mensagens síncronas, o emissor para seu fluxo de execução e espera a resposta do receptor. As mensagens síncronas correspondem, tipicamente, à chamada de procedimento no receptor. Nas mensagens assíncronas, o fluxo de execução do emissor não é interrompido e ele não espera uma resposta do receptor. Esse tipo de mensagem corresponde, tipicamente, ao envio de sinal entre dois objetos que realizam tarefas concorrentemente. A **Figura 8** nos mostra a linha de vida de objetos, bem como a troca de mensagens entre eles.

Figura 8. Representação da troca de mensagens no diagrama de sequência.

Esta figura explicita a troca de mensagens entre duas entidades, **Cliente** e **SistemaWeb**. A mensagem *entrar no sistema* é uma mensagem síncrona, pois a entidade **Cliente** precisa aguardar a resposta da entidade **SistemaWeb**. A resposta da mensagem síncrona é caracterizada pela seta tracejada mostrada no diagrama. Essa seta representa a resposta que o **SistemaWeb** dá ao **Cliente**. Analisando a mensagem *carregar página usando AJAX*, percebemos que o Cliente não precisa aguardar uma resposta do **SistemaWeb**, o que caracteriza uma mensagem assíncrona e não há representação de seta tracejada indicando uma resposta.

A **Figura 9** nos mostra um diagrama de sequência um pouco mais elaborado, contendo o envio de mensagens síncronas e assíncronas, bem como a utilização de um fragmento para modelar uma execução condicional. Um fragmento nada mais é

do que um retângulo no diagrama de sequência que envolve parte das interações, e que apresenta no canto superior esquerdo um operador que indica o tipo de fragmento. No nosso caso, o fragmento representa uma execução condicional (como o if de Java) indicada pelo operador alt. Outros operadores podem ser usados em fragmentos. Por exemplo: loop, indica que as interações no fragmento são executadas um determinado número de vezes (como o comando for de Java); break – fragmento normalmente interno a um fragmento loop – define que, se a interação dentro dele ocorrer, então as interações dentro do loop devem ser interrompidas (como o comando break de Java).

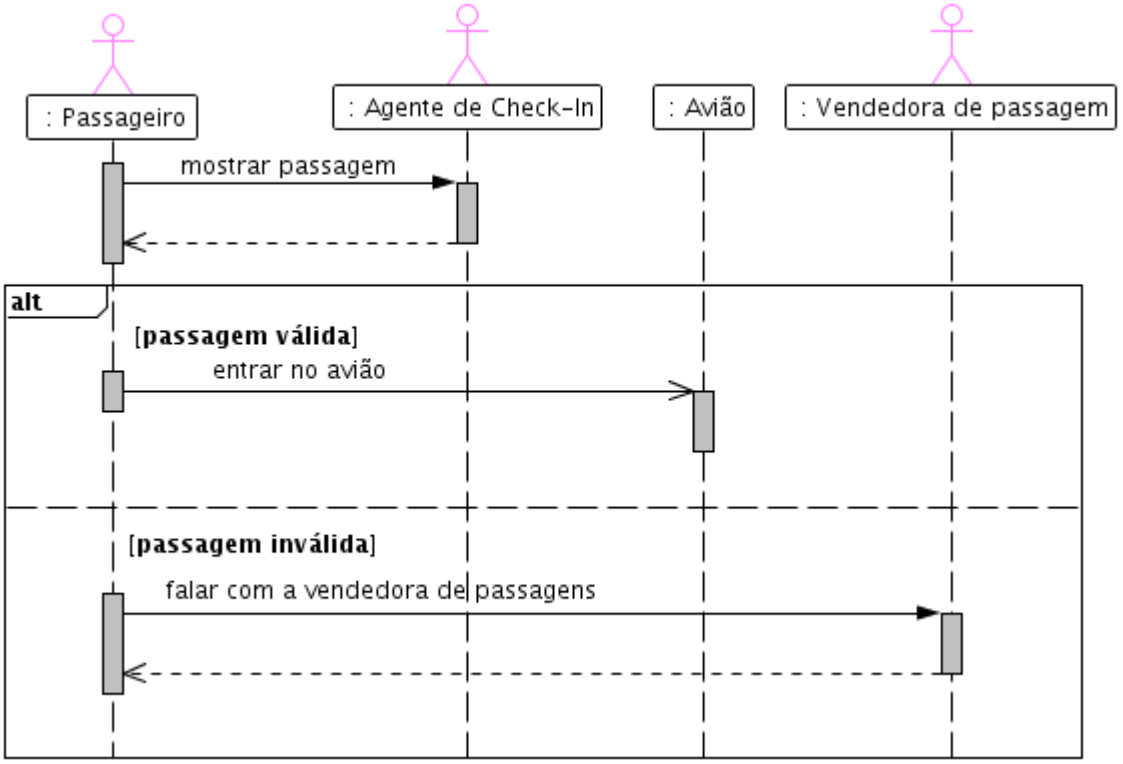

Figura 9. Diagrama de sequência.

De acordo com a **Figura 9**, o passageiro precisa mostrar a passagem para o agente de check-in. Esse agente verifica se a passagem é válida e informa ao passageiro. Caso a passagem seja válida, o passageiro entra no avião normalmente, senão, o passageiro precisa falar com a vendedora de passagens aéreas para averiguar a situação. Esse é o entendimento semântico do diagrama da **Figura 9**.

Podemos melhorar esse modelo usando um outro fragmento para representar um loop, indicando que o passageiro poderá mostrar a passagem novamente ao agente de check-in, caso a sua passagem seja validada pela vendedora de passagem.

Diagrama de atividades

De acordo com Booch, Rumbaugh e Jacobson [1], o diagrama de atividades é essencialmente um gráfico de fluxo, mostrando o fluxo de controle de uma atividade para outra. O propósito de um diagrama de atividades é focar nos fluxos de execução do software e descrever o comportamento de processamentos paralelos.

O diagrama de atividades complementa o caso de uso fornecendo uma representação gráfica do fluxo de interação em um cenário específico. Um diagrama de atividade usa retângulos com cantos arredondados para descrever uma função específica do sistema, losangos para representar pontos de decisão e linhas sólidas para indicar que estão ocorrendo atividades paralelas.

A **Figura 10** representa um diagrama de atividades contendo seus principais elementos. Analisando esta figura, percebemos um fluxo de atividades sendo executadas da esquerda para a direita. Essas atividades são executadas sequencialmente.

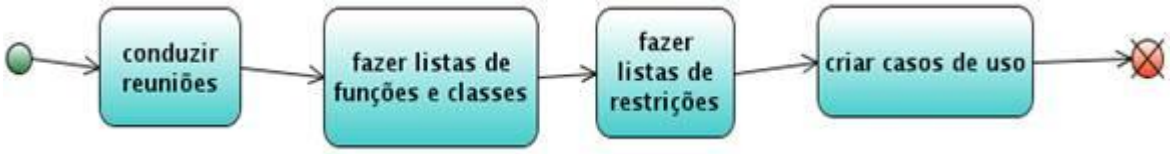

Figura 10. Principais elementos do diagrama de atividades.

De acordo com a **Figura 10**, a primeira atividade a ser realizada é a reunião com os clientes, os quais dizem as necessidades do software. O analista do sistema obtém todas as informações relatadas pelos clientes e as transformam em uma lista de requisitos. Tais requisitos serão mapeados em funções e classes no processo de análise. Ainda no processo de análise, o projetista escreve as restrições do sistema, e com essas informações ele pode criar os casos de uso para o sistema.

Este artefato não modela apenas as atividades executadas sequencialmente. Mostraremos, portanto, como modelar processamento concorrente no diagrama de atividades. Processamento concorrente ou em paralelo é o nome dado aos passos que ocorrem ao mesmo tempo.

Figura 11. Modelagem de atividades concorrentes.

A **Figura 11** apresenta um diagrama de atividades que mostra as ações necessárias para o projetista criar um diagrama de casos de uso. A atividade "*desenhar diagrama de casos de uso"* é executada concorrentemente às atividades "*definir atores* e *escrever cenários"*. As barras verticais representam o início e o fim do processamento concorrente das atividades.

A primeira barra, representando um fluxo chegando e dois saindo, indica o início de um processamento paralelo. Ela é chamada de bifurcação (*fork*). Pode haver mais de dois fluxos saindo no *fork*.

A segunda barra, mostrando dois fluxos chegando e apenas um saindo, indica o fim do processamento concorrente. Ela é chamada de junção (*join*). Da mesma forma que no *fork*, mais de dois fluxos podem chegar no *join*.

Os eventos paralelos não precisam terminar ao mesmo tempo, mas a junção indica que todos precisam ser concluídos – independente do tempo de execução – para que o fluxo continue. Ou seja, se os eventos paralelos tiverem tempos de execução diferentes, o sistema deverá aguardar que todos sejam finalizados antes de prosseguir.

Diagramas de atividades são úteis para detalhar os passos de um caso de uso numa visão de alto nível, diferentemente do diagrama de sequência. Eles também são úteis para representar algoritmos complexos, pois são capazes de modelar fluxos de execução de forma eficiente e clara. Os diagramas de atividades representam o que acontece, mas não representam qual classe realiza determinada atividade.

Veremos agora como desenvolver um diagrama de atividades a partir de um diagrama de casos de uso. Para isso, tomaremos como exemplo um sistema de validação de senhas. A **Figura 12** mostra o caso de uso desse sistema.

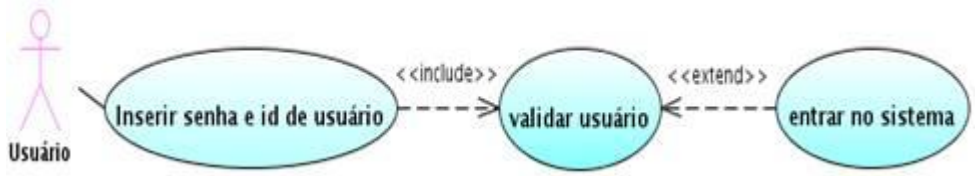

Figura 12. Diagrama de casos de uso do sistema de validação de senhas.

A **Figura 12** nos mostra que o usuário possui um caso de uso *Inserir senha e id de usuário*. Tal caso de uso inclui uma validação de usuário. Se essa validação indicar que o usuário é válido, então ele pode entrar no sistema. A **Figura 13**, por sua vez, nos mostra um diagrama de atividades referente ao diagrama de casos de uso da **Figura 12**.

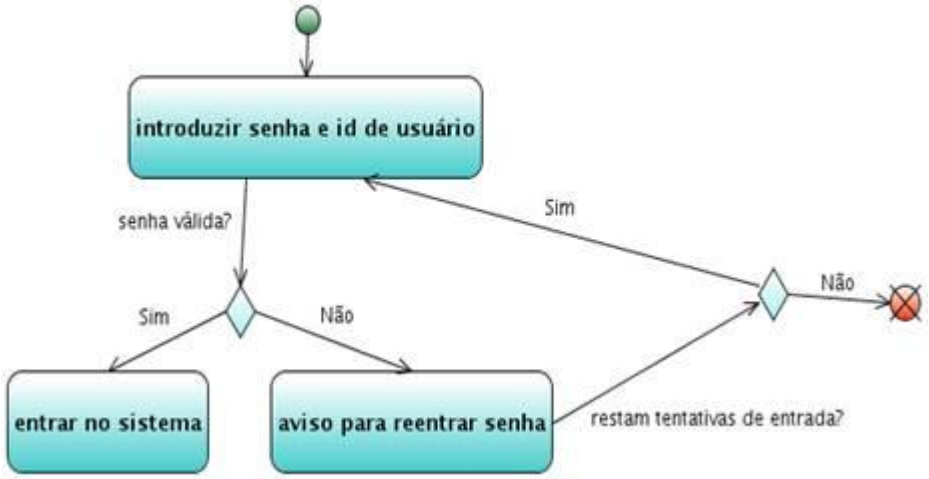

Figura 13. Diagrama de atividades do sistema de validação de senhas

A **Figura 13** demonstra que o usuário é capaz de realizar a atividade *introduzir senha e id de usuário*. Após a realização dessa atividade, teremos que fazer uma validação de usuário, a qual é representada pelo ponto de decisão. Tal ponto de decisão verifica a senha digitada pelo usuário. Caso seja válida, ele direciona o usuário para a atividade *entrar no sistema*, caso contrário, o usuário será direcionado para a atividade *aviso para reentrar senha*.

Caso a senha seja inválida, o sistema irá verificar se ainda restam tentativas para validar a senha. Caso restem tentativas, o processo de envio de senha será repetido, caso contrário, o sistema de validação para.

Até então foram apresentados diagramas de atividades bastante simples, que foram úteis para apresentar os conceitos. Agora veremos um diagrama um pouco mais elaborado. A **Figura 14** mostra um diagrama de atividades para a compra de produtos via internet. Nesse diagrama, utilizamos as barras horizontais para indicar processamento paralelo.

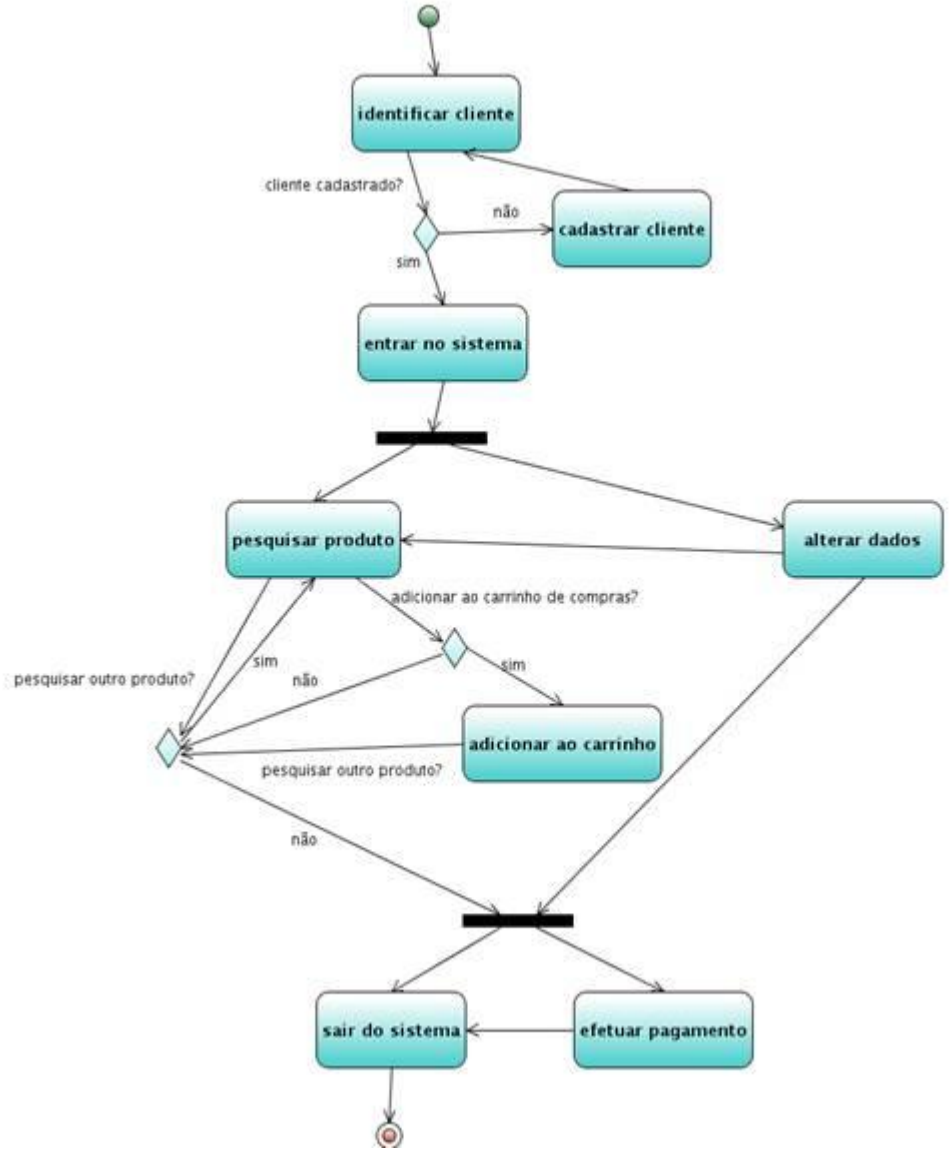

Figura 14. Diagrama de atividades.

A **Figura 14** nos mostra que é necessária a identificação do cliente. Caso o cliente não esteja cadastrado no sistema, ele será direcionado para a área de cadastro, caso contrário, ele pode entrar no sistema com seu id e senha. Ao entrar no sistema, o cliente é capaz de realizar duas atividades concorrentemente: *alterar dados* e *pesquisar produto*.

Se o cliente escolher a atividade *alterar dados*, ele será direcionado para uma área específica para poder realizar essa atividade. O cliente também tem a possibilidade de escolher a opção de pesquisar produto. Nela, o cliente digita um produto que deseja pesquisar e o sistema retorna os possíveis produtos associados à requisição do usuário. Com esses resultados, o cliente pode adicionar alguns produtos ao carrinho de compras ou pesquisar por outros produtos.

Feitos os processos de pesquisa de produtos e adicionar ao carrinho de compras, o cliente pode efetuar o pagamento dos produtos ou sair do sistema. Caso ele saia do

sistema sem efetuar o pagamento, os produtos do carrinho de compras poderão ser armazenados em um banco de dados para que o cliente possa pagar futuramente.

Se o cliente optar por efetuar o pagamento dos produtos, ele será direcionado para uma área de pagamento. Feito o pagamento, o cliente pode sair do sistema. Note que ao finalizar o pagamento, o cliente poderia pesquisar por novos produtos, dependendo do modelo de negócios adotado no desenvolvimento do sistema. De acordo com nosso exemplo, o cliente necessitaria entrar novamente no sistema para pesquisar por novos produtos.

Conclusões

Os diagramas de casos de uso, de sequência e de atividades são diagramas da UML responsáveis por modelar o comportamento dinâmico do sistema, enquanto que o diagrama de classes serve para modelar o comportamento estático do mesmo.

Esses três diagramas apresentam visões diferentes sobre o comportamento do sistema. O diagrama de casos de uso apresenta uma visão de alto nível sobre as funcionalidades do sistema, enquanto que o diagrama de sequência mostra uma visão de baixo nível, indicando como é realizada a troca de mensagens entre partes do sistema. Por sua vez, o diagrama de atividades representa a visão de processo do modelo do sistema, sendo possivelmente o diagrama mais acessível da UML, pois usa notação similar ao amplamente conhecido fluxograma.

Livros

[1] **BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar.** ***UML:***

guia do usuário. 2ª ed. Rio de Janeiro: Campus, 2005.

[2] **SIERRA, Kathy; BATES, Bert.**

Certificação Sun para programador Java 6: guia de estudo. 1ª ed. São Paulo: Alta Books, 2008.

[3] **PRESSMAN, Roger.**

Engenharia de Software. 6ª ed. São Paulo: McGraw-Hill, 2006.

[4] **COCKBURN, Alistair.**

Writing Effective Use-Cases. Addison-Wesley, 2001.

[5] **MCLAUGHLIN, Brett; POLLICE, Gary; WEST, David**.

Use a cabeça – Análise e projeto orientado ao objeto. Rio de Janeiro: Alta Books, 2007.

Disponível em: Devmedia