



## UTILIZANDO UML PARA PROJETAR O SOFTWARE

# DESCRIÇÃO

Uso de diagramas de Linguagem Unificada de Modelagem (UML) na fase de projeto de sistemas de informação.

# PROPÓSITO

Compreender os modelos que fazem parte da UML, empregando os diagramas de interação, de classes de projeto, de atividades, de estado, de componentes e de implantação, durante a fase de projeto de software.

# OBJETIVOS

# **MÓDULO 1**

Empregar os diagramas de interação no projeto de sistema

# **MÓDULO 2**

Revisar o diagrama de classes da análise no projeto de sistema

# **MÓDULO 3**

Empregar os diagramas de estado e de atividades no projeto de sistema

# **MÓDULO 4**

Empregar os diagramas de componentes e de implantação

# **PREPARAÇÃO**

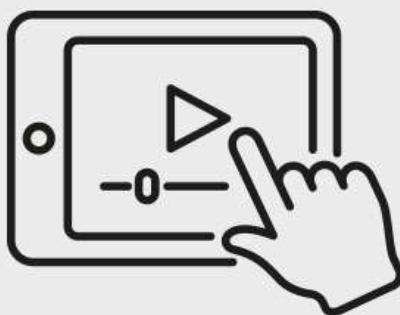
Antes de iniciar o conteúdo, é indicado que seja instalado em seu computador um programa que lhe permita elaborar modelos, sob a forma de diagramas da UML (Linguagem Unificada de Modelagem). Nossa sugestão inicial é o Astah Free Student License e será necessário usar seu e-mail institucional para ativar a licença. Preencha os dados do formulário no site do software, envie e aguarde a liberação de sua licença em seu e-mail. Ao recebê-la, siga as instruções e instale o produto em seu computador.

Sugestões de links adicionais de programas livres para modelagem de sistemas em UML (UML Tools) podem ser encontradas em buscas na internet.

# PASSO A PASSO: INSTALAÇÃO DA LICENÇA ASTAH FREE STUDENT

Neste vídeo prático, será dada uma visão geral sobre modelagem de software e também guiaremos você pelo processo de instalação da Licença Astah Free Student. Desde o download do software até a configuração da licença, cada etapa será detalhadamente abordada.

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



## INTRODUÇÃO

A modelagem de um software pode, de fato, determinar a quantidade de problemas que terão que ser tratados durante o desenvolvimento. Softwares de sucesso, que têm sua implantação quase sem nenhum tipo de erro, certamente contaram com um bom processo de modelagem durante seu desenvolvimento. Ou seja, um bom produto de software é sempre concebido por um bom processo de desenvolvimento, iniciado com a modelagem do sistema.

O processo de desenvolvimento conta com diversos métodos e técnicas para que o produto tenha a melhor qualidade possível, sendo a modelagem uma das técnicas utilizadas. Modelar algo é construir uma representação abstrata de um artefato do mundo real.

A modelagem conta com diversas regras e diretrizes que têm que ser respeitadas para se construir um modelo de qualidade para o software. Todas as funcionalidades implementadas pelo software, assim como todos os elementos de dados que utilize e as regras que devem ser seguidas, precisam ser pensadas, discutidas e estruturadas antes do início de sua construção, de modo que haja um completo entendimento de todo esse universo, não só por parte dos usuários, mas também dos desenvolvedores. Saber identificar cada característica do software

durante seu projeto é importante para que venha de fato atender às necessidades dos usuários e cumprir com todas as regras a serem executadas por ele.

Vamos entender como construir os diagramas ao longo da etapa de projeto (ou design) de software. Nesta parte do desenvolvimento, alguns outros diagramas já terão sido construídos e precisam ser respeitadas suas regras e definições. Para isso, vamos ver as técnicas de concepção e de modelagem desses diagramas, assim como seus atributos e modelos. Por fim, vamos apresentar e discutir como aplicá-los.

## MÓDULO 1

---

- **Objetivo:** Empregar os diagramas de interação no projeto de sistema

## ATIVIDADES DO DESENVOLVIMENTO DE SOFTWARE

## CICLO DE VIDA DE DESENVOLVIMENTO DE SOFTWARE

Neste vídeo, mergulharemos no Ciclo de Vida do Desenvolvimento de Software, com foco na fase de design. Abordaremos como a UML (Unified Modeling Language) é aplicada nessa etapa, explorando diagramas de classes, sequência, comunicação, atividade e muito mais.

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



As atividades típicas de um processo de desenvolvimento de software compreendem, de modo geral, independentemente do processo adotado, as seguintes etapas:

## LEVANTAMENTO DE REQUISITOS



## ANÁLISE



## PROJETO (DESIGN)



## IMPLEMENTAÇÃO



# TESTES



# IMPLEMENTAÇÃO

Como se percebe, essas atividades são realizadas por etapas ou fases, em que uma etapa depende dos artefatos resultantes das anteriores. Neste conteúdo, o foco é a etapa de projeto (ou design) do sistema, na qual existe uma dependência de seus artefatos em relação aos artefatos elaborados na fase de análise (ou modelagem). A figura a seguir ilustra os diagramas da UML que, usualmente, são construídos nos modelos de análise e de projeto. Trata-se de uma visão prática, uma vez que a especificação da linguagem UML não determina quais diagramas devam ser usados em quais etapas e tampouco em que ordem devem ser elaborados.

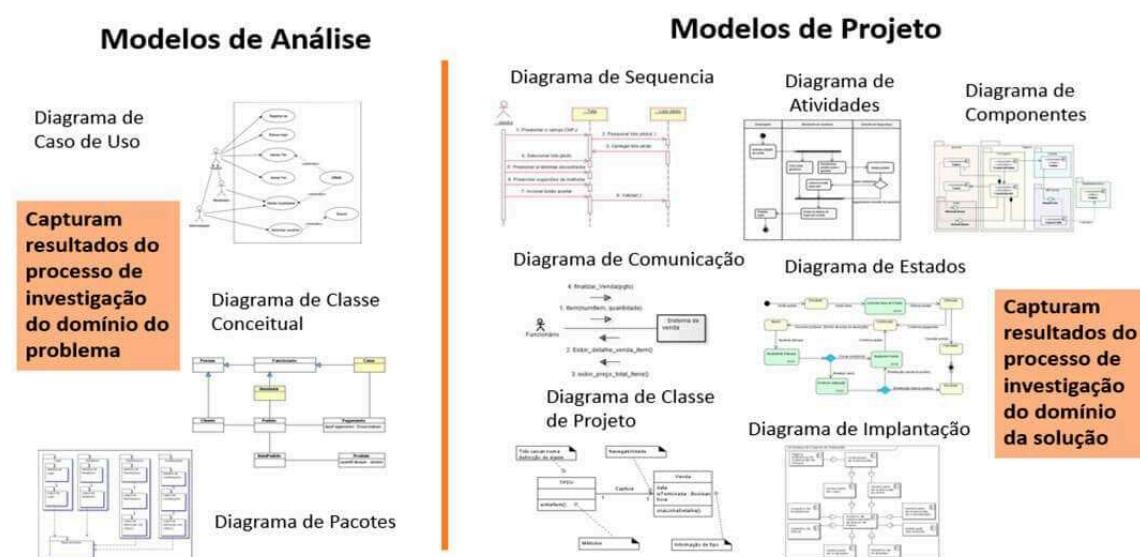


Imagen: Claudia Cappelli

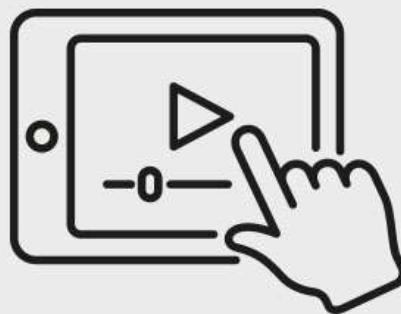
Modelos da fase de análise X modelos da fase de projeto

# DIAGRAMAS DE INTERAÇÃO

## VISÃO GERAL DOS DIAGRAMAS DE INTERAÇÃO

No vídeo a seguir são apresentados os componentes do diagrama de sequência e do diagrama de comunicação.

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



Em um sistema computacional orientado a objetos, as funcionalidades (casos de uso) são fornecidas através de serviços resultantes da colaboração de grupos de objetos. Estes interagem através de comunicações, de forma que juntos realizem os casos de uso.

As comunicações são explicitadas através da representação da interação durante as comunicações. Para tal, temos na UML os diagramas de interação, usados para mostrar o comportamento interativo de um sistema. Os diagramas de interação descrevem o fluxo de mensagens e fornecem contexto para uma ou mais linhas de vida de objetos dentro de um sistema. Além disso, os diagramas de interação podem ser usados para representar as sequências organizadas dentro de um sistema e servir como um meio para visualizar dados em tempo real.

Os diagramas de interação podem ser implementados em diversos cenários para fornecer um conjunto exclusivo de informações. Eles podem ser usados para modelar um sistema, como uma sequência de eventos organizada por tempo; fazer a **engenharia reversa** ou avançada de um sistema ou processo; organizar a estrutura de vários eventos interativos; mostrar, de forma

simples, o comportamento de mensagens e linhas da vida dentro de um sistema e até identificar possíveis conexões entre elementos de linhas da vida. Seus principais objetivos são:

## ENGENHARIA REVERSA

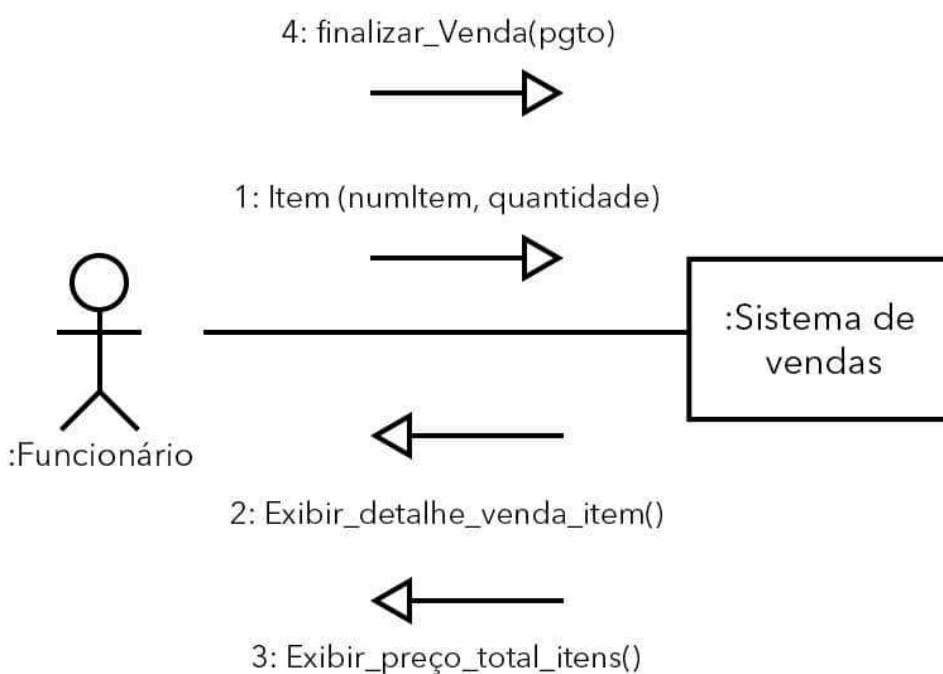
Engenharia reversa, no contexto da Engenharia de Software, significa obter o modelo a partir do sistema implementado.

Obter informações adicionais para completar e aprimorar outros modelos (modelo de casos de uso e de classes).

Fornecer aos programadores uma visão detalhada dos objetos e das mensagens envolvidos na realização dos casos de uso do sistema.

Existem dois **tipos de diagrama de interação**:

## DIAGRAMA DE COMUNICAÇÃO



# DIAGRAMA DE SEQUÊNCIA

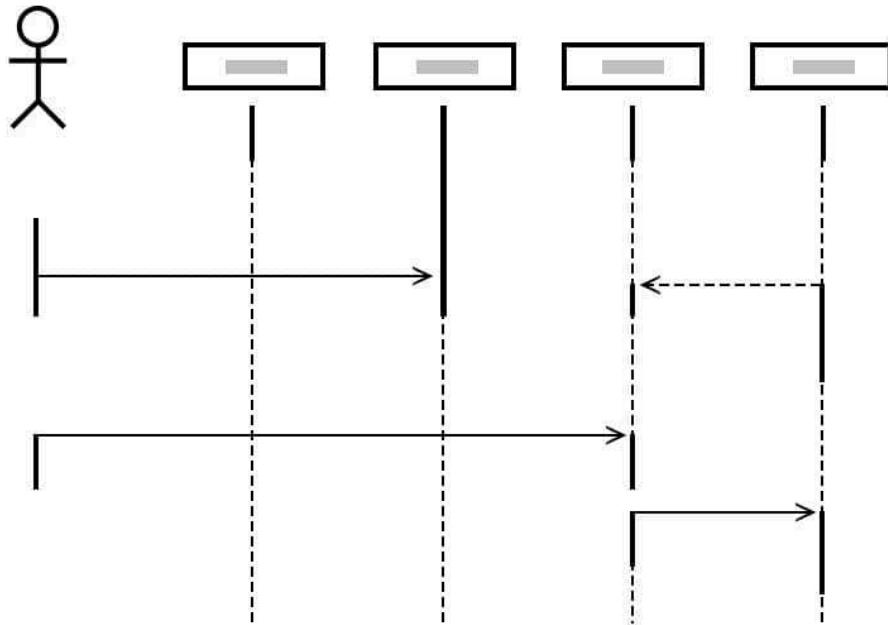


Imagen: Isaac Barbosa

Cada tipo de diagrama aborda uma perspectiva diferente do comportamento de um sistema.

Vamos ver a seguir os conceitos básicos de cada diagrama, como construí-los e utilizá-los.

Num diagrama de interação, as mensagens trocadas entre objetos, em geral, serão métodos da classe a que pertence o objeto receptor da mensagem, com os mesmos elementos oriundos das mensagens. Isso será percebido ao estudarmos os Diagramas de Classe de Projeto.

Os diagramas de interação servem para refinar o diagrama de classes na etapa de projeto, com o detalhamento dos métodos.

Isso nos mostra que os diagramas da etapa de projeto apresentados na figura anterior estão relacionados entre si e com os diagramas da etapa de análise, de modo que a construção de um diagrama depende de outros.

# DIAGRAMA DE SEQUÊNCIA

Diagramas de sequência recebem esse nome porque descrevem, ao longo de uma linha do tempo, a sequência de comunicações entre objetos de um sistema de informação. Em geral,

são construídos logo após os diagramas de caso de uso e de classes, pois têm como principais objetivos:

Documentar casos de uso.

Mostrar como os objetos do sistema se comunicam através de mensagens em ordem temporal.

Validar se todas as operações das classes foram identificadas e declaradas.

Validar a existência de um objeto necessário ao funcionamento do sistema.

## ATENÇÃO

Não é necessário construir diagramas de sequência para todos os casos de uso do sistema.

Apenas os mais complexos, relacionados com o negócio da aplicação, são explorados por esse tipo de diagrama. Ele pode ser bastante trabalhoso e criá-lo para todas as partes do sistema pode não ser produtivo e nem ter benefícios compensatórios.

Dependendo da complexidade ou do número de objetos envolvidos ou da presença de desvios condicionais no caso de uso, vários diagramas de sequência serão necessários para representar as interações do sistema. Assim, devemos construir um diagrama de sequência principal, descrevendo as sequências normais entre os objetos (fluxo normal) e diagramas complementares, descrevendo sequências alternativas (fluxos alternativos) e tratamento de situações de erro (fluxos de exceção).

## ATENÇÃO

Diagramas de sequência são complementares aos diagramas de caso de uso.

Os diagramas de caso de uso são bastante completos nas descrições textuais, porém não são capazes de mostrar como os objetos do sistema se comunicam por intermédio de mensagens em ordem temporal, nem verificar se todas as operações das classes foram validadas e declaradas. Essas funções são muito bem executadas pelos diagramas de sequência.

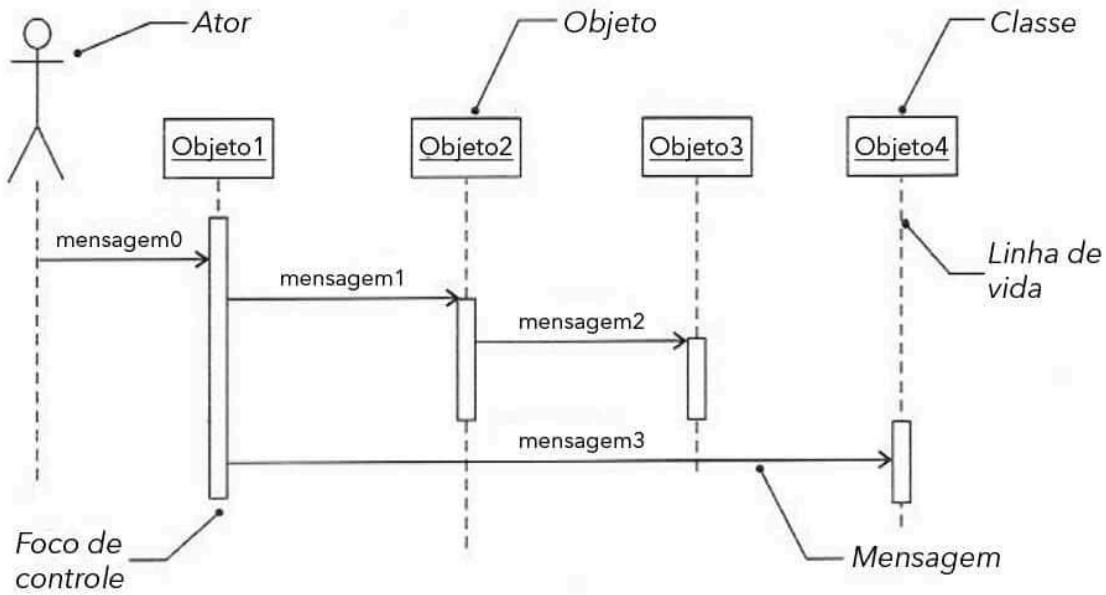


Imagem: BEZERRA, 2015. p. 148.

Exemplo esquemático de diagrama de sequência.

## ELEMENTOS DO DIAGRAMA DE SEQUÊNCIA

O diagrama de sequência é composto de diversos tipos de elementos. Vamos aqui apresentar cada um deles.

## OBJETOS

São apresentados na parte superior do diagrama (um ao lado do outro) na dimensão horizontal.

Não há uma ordem específica de apresentação obrigatória. Sua ordem é relevante somente como forma de tornar o diagrama mais legível. Ou seja, por exemplo, uma boa prática pode ser colocar os elementos que têm mais interação mais próximos. Já no centro e abaixo dos objetos temos sua linha da vida (linha reta tracejada). Todo e qualquer objeto ou classe nesse diagrama pode ser representado com um retângulo com o seu nome, mas existem notações específicas para tipos especiais de objetos. São quatro tipos especiais, a saber:

## ATOR

É uma entidade externa que interage com o sistema enviando ou recebendo dados. Um ator é representado pelo estereótipo de boneco magro (“palito”), como nos diagramas de casos de

uso. Mensagens podem ser trocadas entre atores, ou entre atores e objetos ou classes de fronteira.



Imagen: Claudia Cappelli

Exemplo de Ator. Elaborado na ferramenta Astah UML.

## FRONTEIRA

É uma classe ou um objeto responsável pela interface entre o sistema de informação e seus usuários, ou pela comunicação entre dois ou mais sistemas. Em geral, são programas (formulários, interfaces, janelas) que capturam, processam dados e exibem informações. É através da classe de fronteira que os atores (usuários) terão seus requisitos atendidos pelo sistema.



Imagen: Claudia Cappelli

Exemplo de Fronteira. Elaborado na ferramenta Astah UML.

## CONTROLE

É uma classe ou um objeto usado quando os desenvolvedores optam por separar as regras de negócio, as manipulações de dados e o sequenciamento de tarefas da interface de comunicação do sistema com o usuário, seguindo um padrão de projeto conhecido como MVC (do inglês *Model – View – Controller* ). Dessa forma, a classe de fronteira (*View* ) só existe como interface para receber e exibir dados. Todas as outras tarefas são executadas pela classe de controle (*Controller* ).



Imagen: Claudia Cappelli

Exemplo de Controle. Elaborado na ferramenta Astah UML.

## ENTIDADES

São classes ou objetos responsáveis por armazenar e gerenciar os dados do sistema, bem como por mostrar a estrutura lógica e estática dos dados (Model no padrão MVC).

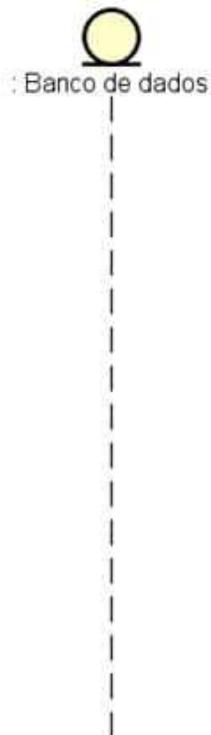


Imagen: Claudia Cappelli

Exemplo de Entidade. Elaborado na ferramenta Astah UML.

# LINDA DA VIDA

Compõe a dimensão vertical (tempo) de cada objeto do diagrama. As linhas de vida representam a sequência na qual a vida do objeto transcorre durante a interação. Podem apresentar criação ou destruição de objetos. Quando são criadas interações entre os objetos, um retângulo sobre a linha é colocado (automaticamente pelas ferramentas de desenho de diagrama), identificando a interação. Mais de um retângulo pode existir em um único objeto. Esse retângulo é chamado de **tempo de vida da interação**.

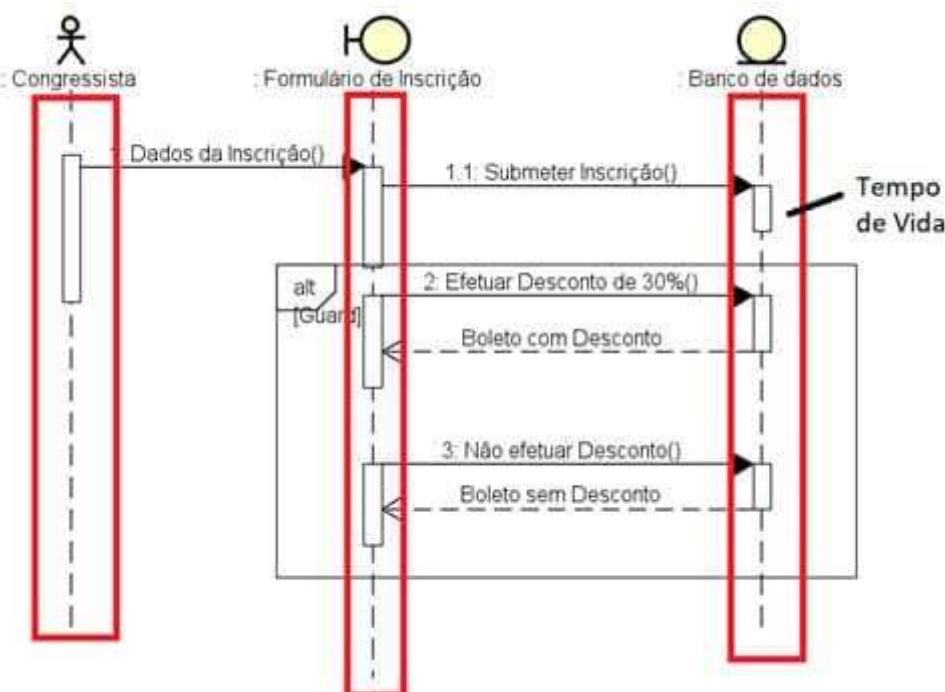


Imagen: Claudia Capelli

Exemplos de linhas de vida (destacados em vermelho). Elaborado na ferramenta Astah UML.

# MENSAGENS

São os serviços solicitados de um objeto a outro ou a ele mesmo, e as respostas a essas solicitações. Uma mensagem é representada por um segmento de reta, contendo uma seta em uma das suas extremidades. As mensagens são acompanhadas de um rótulo contendo seu nome. Pode conter também parâmetros da mensagem. As mensagens podem ser síncronas, assíncronas, de retorno e de criação de objeto. Para cada uma delas, a linguagem usa notações diferentes. Na UML, o rótulo de uma mensagem deve seguir a seguinte sintaxe:

# **[[EXPRESSÃO-SEQUÊNCIA] CONTROLE:] [V :=] NOME [(ARGUMENTOS)]**

## **► ATENÇÃO**

Os colchetes [ ] indicam que o elemento é opcional. Note que o único termo obrigatório corresponde ao nome da mensagem.

Muitas vezes é necessário indicar que o termo controle está associado a uma expressão lógica. Nesse caso, a sintaxe para o termo controle pode ser uma condição ou uma iteração:

## **“\*” ‘[’ CLÁUSULAITERAÇÃO ‘]’ ‘[’ CLÁUSULACONDIÇÃO ‘]’**

Onde o símbolo \* significa repetições da cláusula iterativa. Nessa sintaxe, os elementos entre aspas simples “ ” significam que fazem parte da expressão.

→	Mensagem síncrona
→	Mensagem assíncrona
<-----	Mensagem de retorno
«create» ----->	Mensagem de criação de objeto

Imagen: Claudia Cappelli

Tipos de setas X tipos de mensagens

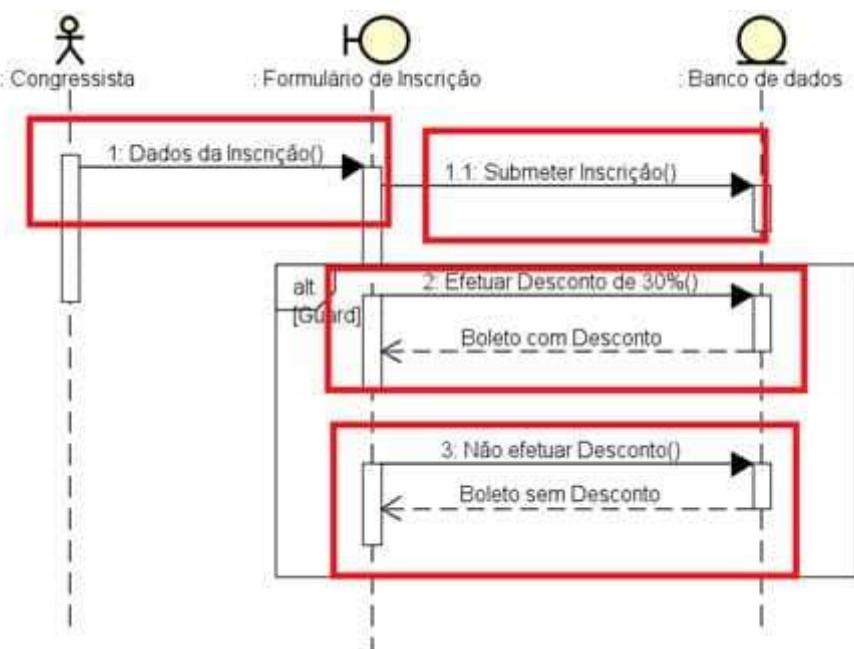


Imagen: Claudia Cappelli

Exemplos de Mensagens (destacadas em vermelho). Elaborado na ferramenta Astah UML

## MENSAGENS DE ITERAÇÃO

São mensagens que podem ser enviadas repetidas vezes. São precedidas de um asterisco e representadas dentro de colchetes.

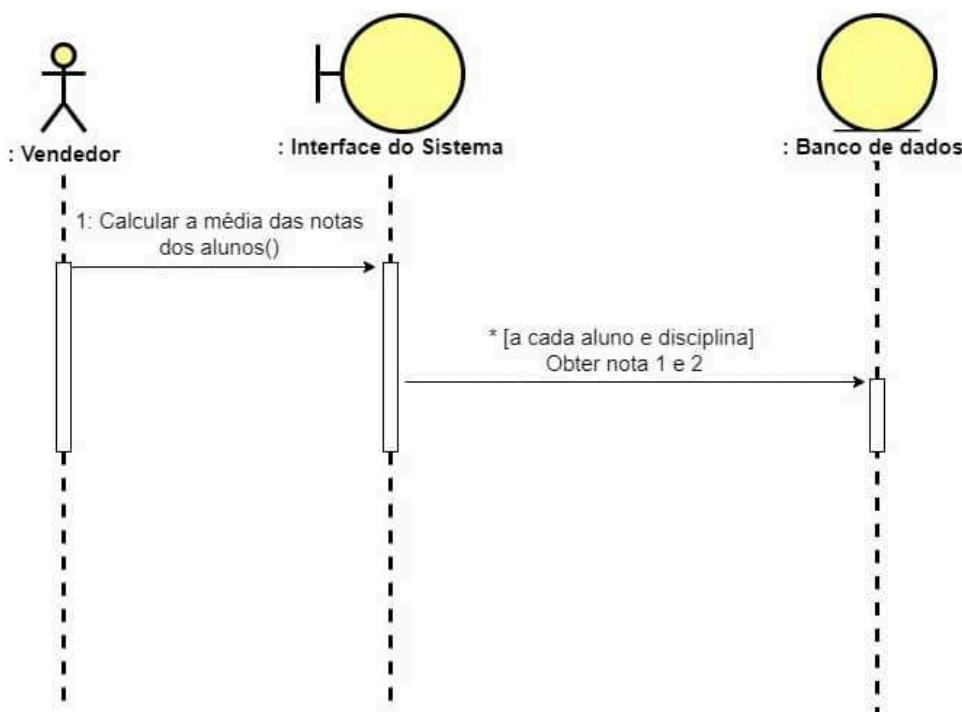


Imagen: Claudia Cappelli, adaptado por Isaac Barbosa

## CONDIÇÕES DE GUARDA

São mensagens que tem indicada uma condição para seu envio.

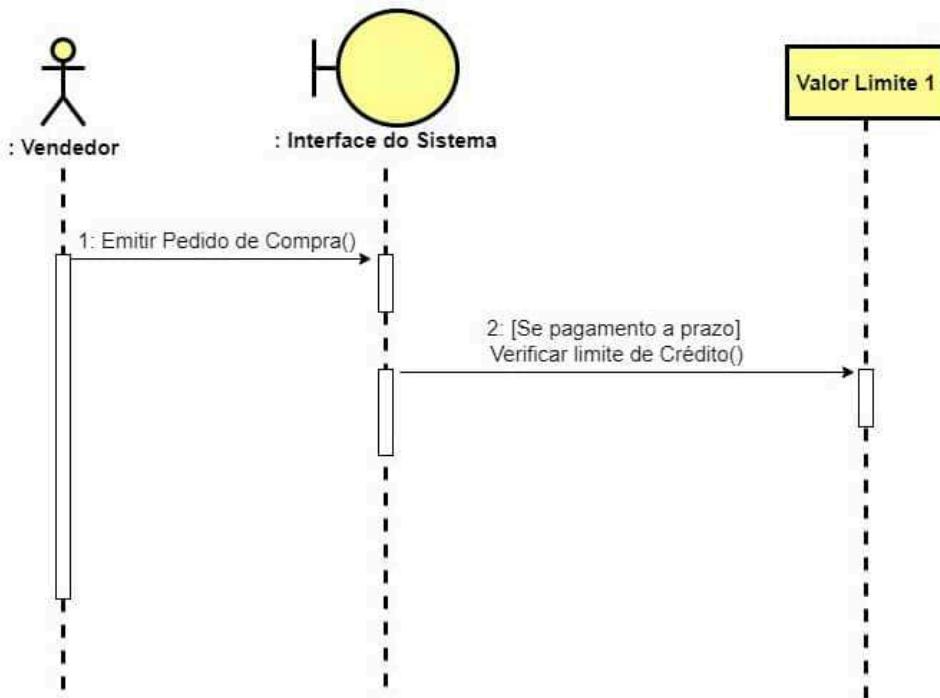


Imagen: Claudia Cappelli, adaptado por Isaac Barbosa

Exemplo de condições de guarda. Elaborado na ferramenta Astah UML.

## AUTOCHAMADA

São mensagens que partem do objeto de origem e atingem a linha de vida dele mesmo disparando operações.

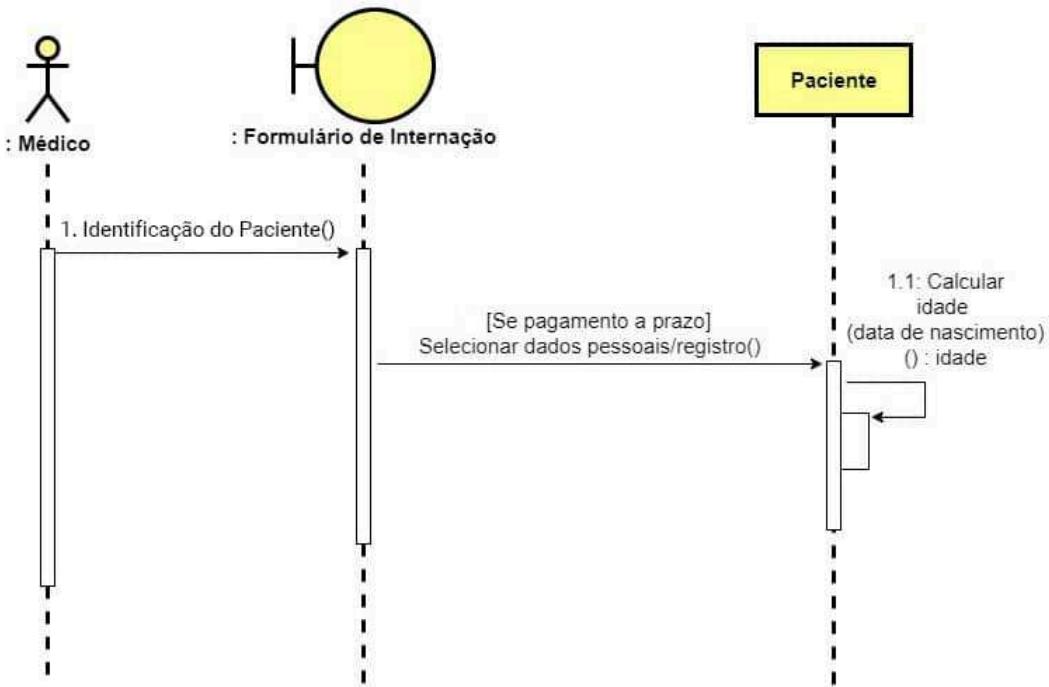


Imagen: Claudia Cappelli, adaptado por Isaac Barbosa

Exemplo de autochamada. Elaborado na ferramenta Astah UML.

## OCORRÊNCIA DE INTERAÇÃO

São elementos de modelagem úteis quando se deseja simplificar um diagrama e extrair uma porção para outro, ou quando existe uma porção de ocorrência que se possa fazer reuso em outro diagrama. É representado por uma moldura com a etiqueta “ref” que se refere a outro diagrama de sequência.

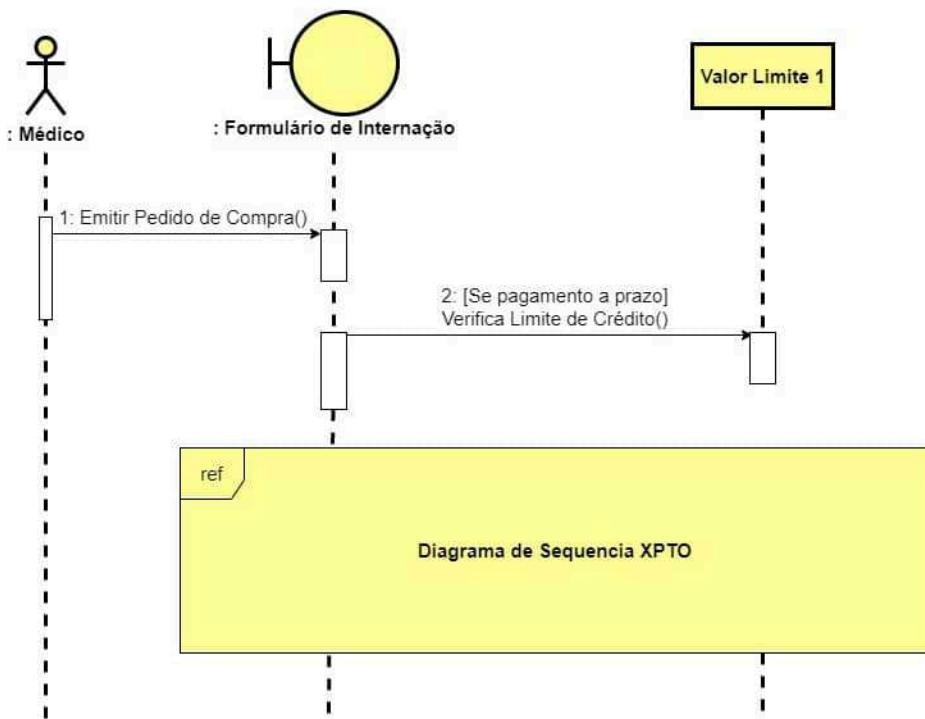


Imagen: Claudia Cappelli, adaptado por Isaac Barbosa

Exemplo de ocorrência de interação. Elaborado na ferramenta Astah UML.

## ***OPT***

São elementos que permitem tornar um trecho da interação opcional.

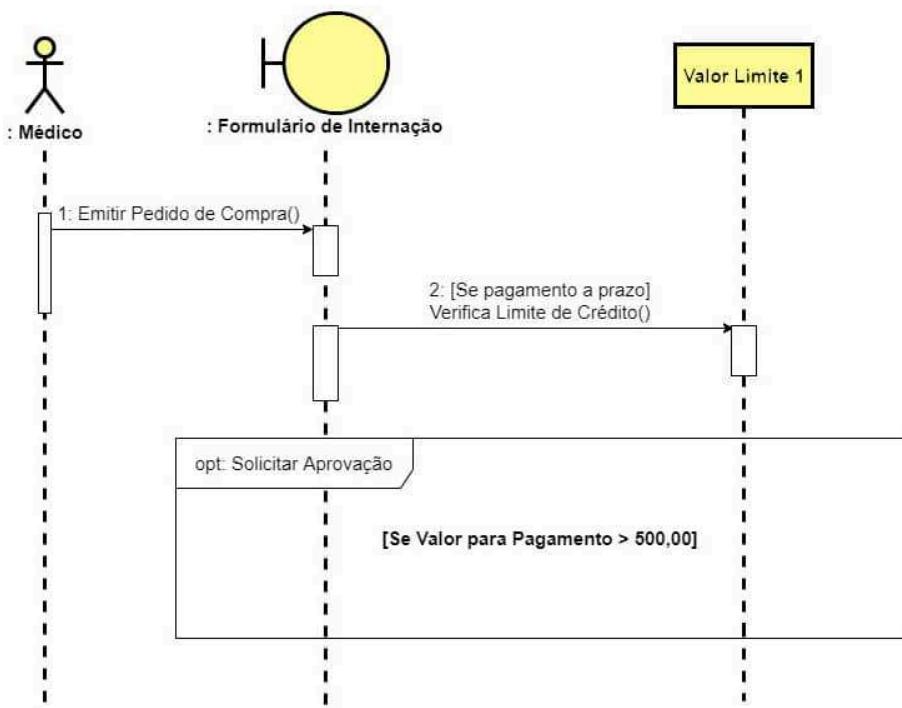


Imagen: Claudia Cappelli, adaptado por Isaac Barbosa

Exemplo de interação opcional “opt” . Elaborado na ferramenta Astah UML.

## ***LOOP***

São elementos que permitem repetir um trecho da interação até “n” vezes.

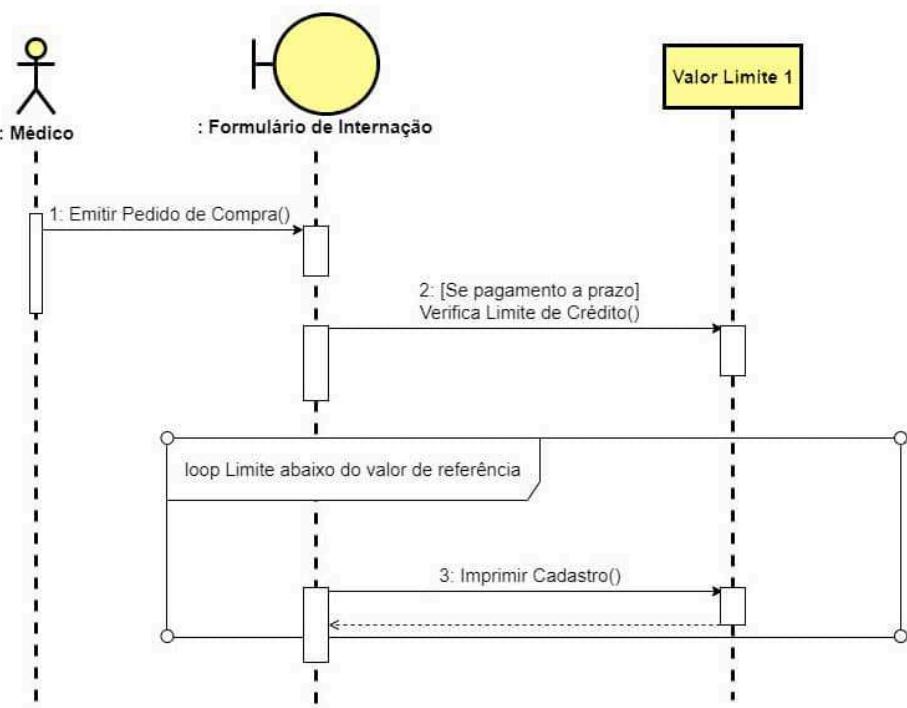


Imagen: Claudia Cappelli, adaptado por Isaac Barbosa

Exemplo de interação repetida “loop” . Elaborado na ferramenta Astah UML.

## **ALT**

São elementos que permitem que um trecho da interação seja alternativo.

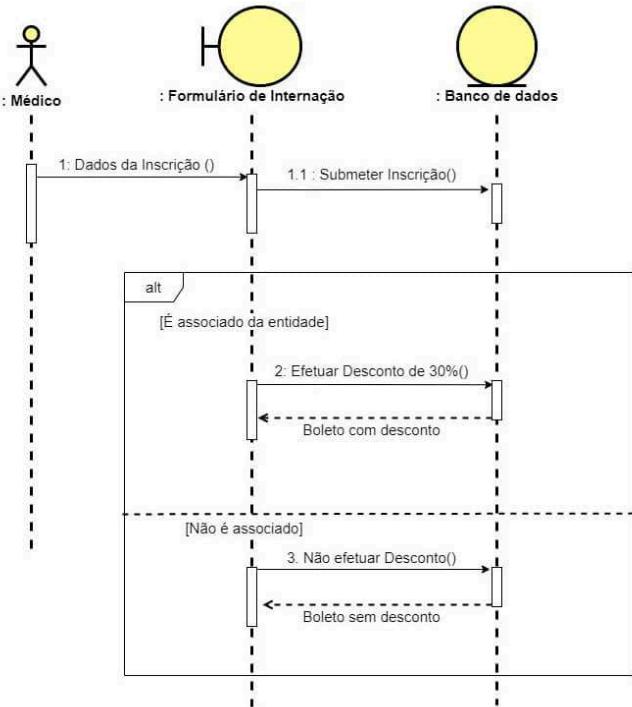


Imagen: Claudia Cappelli, adaptado por Isaac Barbosa

Exemplo de interação alternativa “alt” . Elaborado na ferramenta Astah UML.

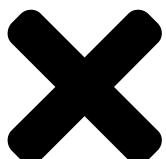
## DIAGRAMA DE COMUNICAÇÃO

Diagramas de comunicação servem para mostrar como os objetos interagem para executar o comportamento de um caso de uso ou de parte dele, para determinar interfaces e responsabilidades de classes e definir os papéis dos objetos que executam determinado fluxo de eventos. Mostram a comunicação, ou seja, o fluxo de mensagens que ocorre entre os objetos de um sistema de informação, assim como o diagrama de sequência. Os dois expressam informações semelhantes, mas numa forma de exibição diferente.

Então, qual diagrama usar?

Diagrama de Sequência

Se o objetivo for apresentar o fluxo de mensagens no decorrer do tempo, utilize o diagrama de sequência.



## Diagrama de comunicação

Se a intenção for de dar ênfase a como os objetos estão vinculados e quais mensagens trocam entre si, use o diagrama de comunicação.

No diagrama de comunicação, um vínculo é uma associação que identifica a ligação entre dois objetos envolvidos em um processo e é caracterizado pelo envio ou recebimento de uma mensagem ou de ambos. As mensagens enviadas de um objeto para outro são representadas por segmentos de retas com uma seta em uma das extremidades, indicando sua direção.

Mensagens devem ter nome, parâmetros (opcional) e a sequência que é expressa por números, assim como no diagrama de sequência.

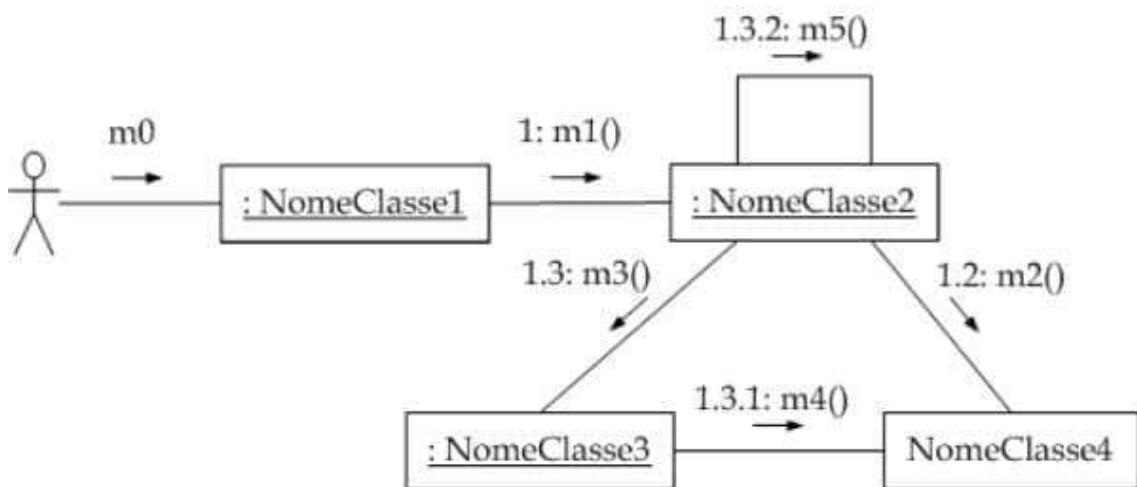


Imagen: BEZERRA, 2015. p. 153. Adaptado por Isaac Barbosa

Exemplo esquemático de diagrama de comunicação correspondente ao diagrama de sequência, anteriormente apresentado.

O exemplo esquemático acima ilustra a correspondência que existe entre o diagrama de sequência e o diagrama de comunicação. Os mesmos objetos são envolvidos, trocando as mesmas mensagens. O fluxo de execução das mensagens, que no diagrama de sequência é determinado pela ordem temporal de cima para baixo, aqui no diagrama de comunicação é determinado pela numeração ordenada das mensagens.

# COMO CONSTRUIR UM DIAGRAMA DE INTERAÇÃO

Para criação de um diagrama de interação (sequência ou comunicação), você pode seguir os seguintes passos (em geral):

## **1. DETERMINE O CASO DE USO QUE SERÁ MODELADO E IDENTIFIQUE SUAS OPERAÇÕES**



## **2. PARA CADA OPERAÇÃO, IDENTIFIQUE OS OBJETOS QUE FARÃO PARTE DE SUA INTERAÇÃO.**



## **3. IDENTIFIQUE AS CONEXÕES E OS RELACIONAMENTOS ENTRE ELES E, EM SEGUIDA, CATEGORIZE-OS.**



## **4. IDENTIFIQUE A SEQUÊNCIA DE FLUXOS DE MENSAGENS NA INTERAÇÃO ENTRE OS OBJETOS.**



**5. IDENTIFIQUE OS TIPOS DE MENSAGENS, BEM COMO SUA SEQUÊNCIA.**



**6. EM SEU SOFTWARE DE DIAGRAMAS UML, SELECIONE AS FORMAS APROPRIADAS PARA CRIAR O DIAGRAMA.**



**7. INSIRA UM RÓTULO PARA TODAS AS FORMAS PARA REPRESENTAR CADA EVENTO, INTERAÇÃO E MENSAGEM EM SEU SISTEMA.**



**8. DÊ NOME AO SEU DIAGRAMA USANDO ALGO QUE REALMENTE O IDENTIFIQUE. COM O VOLUME, PODE**

# FICAR COMPLEXA A BUSCA PELO DIAGRAMA, CASO USE NOMES SEM MUITA SEMÂNTICA.

## DICA

Não se recomenda incluir o ator nem o objeto de fronteira primeiramente, porque isso dificulta o entendimento. É mais recomendado iniciar a modelagem da interação com a representação do recebimento da mensagem para ativação da operação de sistema, no caso de uso. Essa recomendação ajuda a separar a lógica do domínio e a lógica da interface com o usuário.

As classes conceituais identificadas, que participam em cada caso de uso, correspondem às entidades do mundo real envolvidas na tarefa do caso do uso, como se este fosse executado manualmente. Durante a modelagem de interações, objetos dessas classes participam da realização de um ou mais casos de uso. Por outro lado, durante a modelagem de interações, o projetista pode ter a necessidade de adicionar classes de software (ou seja, classes da aplicação que não têm correspondência no mundo real) que ajudem a organizar as tarefas a serem executadas. Essas classes de software normalmente são necessárias para manter a coesão e o acoplamento das demais classes em um nível adequado.

Durante a aplicação do procedimento descrito acima, o projetista deve procurar construir diagramas de interação o mais inteligíveis possível. Por exemplo, é possível utilizar notas explicativas para esclarecer algumas partes do diagrama de interação. Essas notas podem conter texto livre ou pseudocódigo para orientar a programação. Outra estratégia que ajuda a construir um modelo de interações mais inteligível é utilizar os recursos de modularização (*alt*, *loop*, *opt* ).

## ATENÇÃO

Sempre que for adequado, o projetista deve fazer, com base nos princípios de projeto, com que as classes de domínio enviem mensagens entre si. Dessa forma, o controlador envia uma mensagem para um objeto do domínio e este, por sua vez, dispara o envio de mensagens para outros objetos do domínio, o que evita que o controlador precise ter conhecimento desses últimos.

# **EM QUE MOMENTO CONSTRUIR O MÓDELO DE INTERAÇÕES**

Alguns textos sobre modelagem de sistemas orientados a objetos indicam o início da modelagem de interações já na atividade de análise. Outros defendem o início de sua construção somente na etapa de projeto. O fato é que a distinção entre essas duas fases não é tão nítida na modelagem orientada a objetos e a modelagem de interações pode ser realizada em ambas as fases.

Na análise, podemos começar a construir o modelo de interações logo depois que uma primeira versão do modelo de casos de uso estiver pronta. Inicialmente, o modelo de interações pode ser utilizado para representar apenas os objetos participantes em cada caso de uso e com mensagens exibindo somente o nome da operação. Posteriormente (na etapa de projeto), esse modelo deve ser refinado, incluindo criação e destruição de objetos, detalhes sobre o tipo e assinatura completa de cada mensagem etc.

No caso do processo incremental e interativo de desenvolvimento, onde os modelos evoluem em conjunto durante o desenvolvimento do sistema, embora esses modelos representem visões distintas do sistema, eles são interdependentes. Os itens a seguir demonstram como o modelo de interações se relaciona aos de casos de uso e de classes.

## **MODELO DE CASOS DE USO → MODELO DE INTERAÇÕES**

Utilizamos os cenários extraídos do modelo de casos de uso como fonte de identificação de mensagens na modelagem de interações. É adequado que se verifique se cada cenário relevante para todos os casos de uso foi considerado na modelagem de interações.

## **MODELO DE CLASSES DE ANÁLISE → MODELO DE INTERAÇÕES**

Utilizamos informações provenientes do modelo de classes para construir modelos de interações. Em particular, algumas responsabilidades já podem ter sido definidas durante a modelagem de classes de análise. A notação do diagrama de interações não precisa ser totalmente utilizada na fase de análise. Em particular, podemos definir diagramas de interação que apresentam as mensagens apenas com os seus nomes. Essa definição inicial deve ser refinada posteriormente com a utilização da sintaxe mais avançada.

# **MODELO DE INTERAÇÕES → MODELO DE CASOS DE USO**

A partir do conhecimento adquirido com a construção do modelo de interações, podemos aperfeiçoar e validar os cenários do modelo de casos de uso.

## **MODELO DE INTERAÇÕES → MODELO DE CLASSESS**

É pela construção de modelos de interações que informações necessárias para a alocação e o detalhamento de operações (métodos) para classes surgem. Durante a construção de um diagrama de interação, pode ser que algumas informações necessárias em certa classe de análise ainda não existam, em virtude de não terem sido identificadas anteriormente. Portanto, além da identificação de operações, é provável que novos atributos sejam identificados durante a modelagem de interações. Outro elemento do modelo de classes que é comumente identificado ou validado pela construção do modelo de interações são as associações: uma mensagem entre objetos implica a existência de uma associação entre as classes envolvidas. Também é comum a situação em que identificamos novas classes durante a modelagem de interações, principalmente aquelas relacionadas ao domínio da solução do problema.

## **VERIFICANDO O APRENDIZADO**

### **1. MARQUE A OPÇÃO QUE NÃO APRESENTA UMA CARACTERÍSTICA DO DIAGRAMA DE SEQUÊNCIA:**

- A)** Documentam casos de uso.
- B)** Validam se as operações das classes foram declaradas
- C)** Mostram as mensagens entre objetos.
- D)** São necessários em todos os casos de uso do sistema.
- E)** Descrevam ao longo de uma linha do tempo a sequência de comunicações

## **2. ASSINALE A ALTERNATIVA QUE INDICA UMA ATIVIDADE DO PROCESSO DE CONSTRUÇÃO DOS DIAGRAMAS DE INTERAÇÃO:.**

- A)** Identificar a sequência de fluxos de mensagens na interação entre os objetos.
- B)** Identificar os diagramas de classe.
- C)** Listar cada atributo e método do diagrama de classes.
- D)** Identificar componentes do software.
- E)** Listar interfaces do software.

---

## **GABARITO**

### **1. Marque a opção que não apresenta uma característica do diagrama de sequência:**

A alternativa "**D**" está correta.

Não há necessidade de construção de diagramas de sequência para todos os casos de uso de um sistema, mas apenas para os casos de uso primários, ou seja, aqueles que representam atividades típicas do negócio da aplicação.

### **2. Assinale a alternativa que indica uma atividade do processo de construção dos diagramas de interação:.**

A alternativa "**A**" está correta.

A sequência de fluxos de mensagens é uma atividade importante do processo de construção dos diagramas de interação, seja de sequência ou de comunicação, pois é através das mensagens que ocorre a interação entre os objetos do sistema.

---

# **MÓDULO 2**

- 
- **Objetivo:** Revisar o diagrama de classes da análise no projeto de sistema

# DIAGRAMA DE CLASSES DE PROJETO

## VISÃO GERAL SOBRE DIAGRAMAS DE CLASSE DE PROJETO

No vídeo a seguir são apresentados os componentes do diagrama de classes de projeto.

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



Vamos tratar agora do diagrama de classes de projeto. Esse diagrama é um refinamento do diagrama de classes construído na fase de análise. Nesse refinamento, algumas classes sofrem alterações de suas propriedades com o objetivo de transformar o modelo de classes de análise no modelo de classes de projeto, assim como suas notações adicionais, transformações sobre atributos, operações e associações.

Além disso, vamos abordar também os relacionamentos entre classes, apresentando outros elementos de notação e princípios de modelagem que não são usados na análise, mas que são importantes na construção do modelo de classes de projeto.

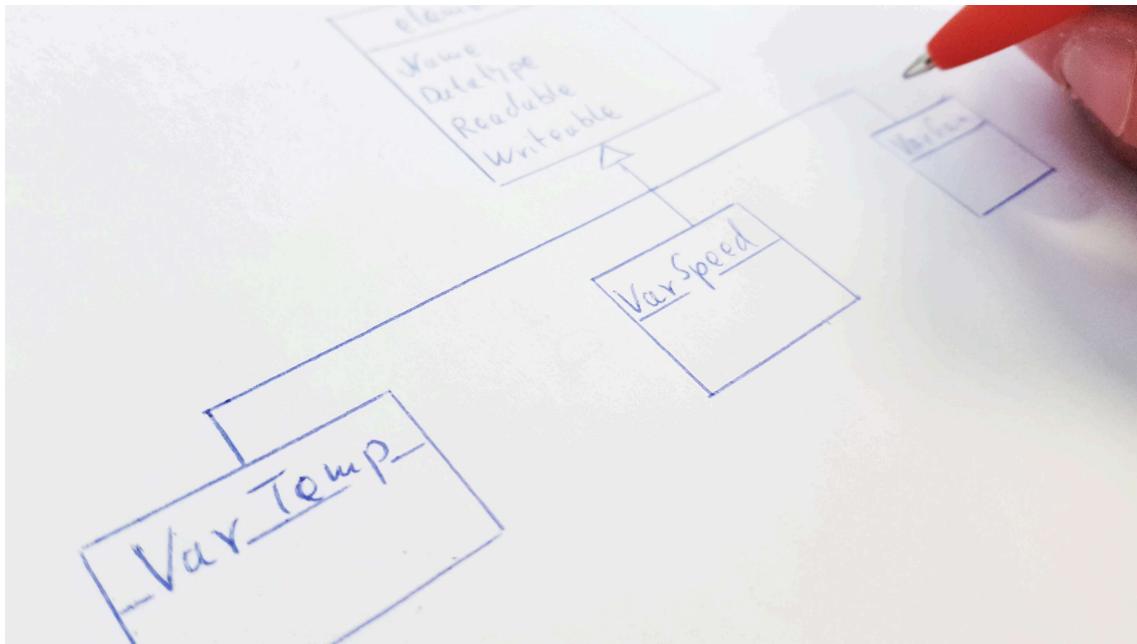


Imagen: Shutterstock.com

## DEFINIÇÃO

O diagrama de classes de projeto apresenta os dados que serão necessários para a construção do sistema de informação. Esse é o diagrama que possui o maior número de símbolos para sua diagramação.

O seu principal objetivo é permitir a visualização dos componentes das bases de dados do sistema, ou seja, as classes e seus objetos, relações entre elas, suas operações, seus métodos, suas interfaces e seus pacotes.

Também é papel desse diagrama atender às demandas dos casos de uso, armazenando as estruturas de dados projetadas para o sistema.

Na fase de análise, o diagrama de classes pode ser usado para produzir o modelo conceitual por meio do qual representamos as classes e seus atributos e não levamos em consideração aspectos relacionados ao software. Apenas estamos interessados no domínio do problema, ou seja, na representação das informações necessárias ao sistema e nos conceitos e características observados no ambiente. Já na fase de projeto, definimos a solução do problema, ou seja, damos importância ao software por meio do qual as classes serão programadas, evidenciamos detalhes da implementação, como, por exemplo, os tipos de dados dos atributos (*varchar, integer, date etc.* ) e suas operações e métodos.

# ESPECIFICAÇÃO DE ATRIBUTOS

No modelo de classes de análise, os atributos são definidos somente pelo nome. No entanto, a sintaxe completa da UML para definição de um atributo é bem mais detalhada. Essa sintaxe, usada na fase de especificação do modelo de classes de projeto, é a seguinte:

**<ATRIBUTO> ::= [/] [VISIBILIDADE] NOME[: TIPO] [= VALOR\_INICIAL]**

A visibilidade de um atributo indica seu nível de acesso. Ou seja, ela permite definir quais atributos de um objeto são acessíveis por outros objetos. A visibilidade de um atributo pode ser pública, protegida, privativa ou de pacote. A primeira é o que chamamos de **default** (se nenhuma visibilidade for especificada, essa é assumida).

## DEFAULT

O termo default diz respeito a um parâmetro configurado de forma automática sem intervenção do usuário, ou seja, um parâmetro utilizado como padrão pelo sistema.

A propriedade de visibilidade serve para implementar o encapsulamento da estrutura interna da classe. Somente as propriedades realmente necessárias ao exterior da classe devem ser definidas com visibilidade pública. Todo o resto deve ser definido através das visibilidades protegida e privativa. Abaixo temos a lista de tipos de visibilidades, seus símbolos e significados:

## PÚBLICA (+)

Qualquer objeto externo pode obter acesso ao elemento, desde que tenha uma referência para a classe em que o atributo está definido.

## **PROTEGIDA (#)**

O elemento protegido é visível para subclasses da classe em que este foi definido.

## **PRIVATIVA (-)**

O elemento privativo é invisível externamente para a classe em que este está definido.

## **PACOTE (~)**

O elemento é visível a qualquer classe que pertence ao mesmo pacote no qual está definida a classe.

O elemento nome na sintaxe do atributo corresponde ao nome do atributo. Na verdade, somente esse elemento é obrigatório na sintaxe de declaração de um atributo, visto que os demais são opcionais.

O elemento tipo especifica o tipo do atributo. Esse elemento é dependente da linguagem de programação na qual a classe deve ser implementada. O tipo de um atributo pode ser definido pela utilização de um tipo primitivo da linguagem de programação a ser utilizada na implementação.

## **★ EXEMPLO**

Se estamos utilizando a linguagem Java, temos, dentre outros, os seguintes tipos primitivos:  
*int , float e boolean .*

Um valor inicial também pode ser declarado para o atributo. Dessa forma, sempre que um objeto de uma classe é instanciado, o valor inicial declarado é automaticamente definido para o atributo. Assim como o tipo, o valor inicial de um atributo também é dependente da linguagem de programação.

Pode-se definir um atributo derivado em uma classe. Um atributo é derivado quando o seu valor pode ser obtido a partir do valor de outro(s) atributo(s). Por exemplo, o valor da idade de uma pessoa pode ser obtido a partir de sua data de nascimento. Um atributo derivado é representado com uma barra inclinada à esquerda (/). Atributos derivados normalmente são definidos por questões de desempenho.

Os atributos também têm multiplicidade. Esta determina o número mínimo e máximo de valores que um atributo pode conter. Abaixo podemos ver as multiplicidades e seus significados, que são similares às multiplicidades dos relacionamentos entre classes:

## **0..1**

Indica que o atributo tem no mínimo zero (é opcional) e no máximo um valor.

## **1..1**

Indica que o atributo tem no mínimo um (é obrigatório) e no máximo um valor.

## **0..\***

Indica que o atributo tem no mínimo zero (é opcional) e no máximo muitos valores.

**\***

Indica que o atributo tem muitos valores e equivale a 0..\* (é opcional).

## **1..\***

Indica que o atributo tem no mínimo um (é obrigatório) e no máximo muitos valores.

## **2..6**

Indica que o atributo tem no mínimo dois e no máximo seis valores.

# ESPECIFICAÇÃO DE OPERAÇÕES/MÉTODOS

Na descrição do modelo de interações, declaramos que as operações de uma classe são identificadas em consequência da identificação de mensagens enviadas de um objeto a outro. No detalhamento dessas operações, devemos considerar diversos aspectos: seu nome, lista de parâmetros, tipo de cada parâmetro, tipo de retorno. As operações de uma classe correspondem a algum processamento realizado por essa classe. Em termos de implementação, uma operação é uma rotina associada a uma classe. A sintaxe definida na UML para uma operação é a seguinte:

**[VISIBILIDADE] NOME([PARÂMETROS]) [: TIPO-RETORNO] [{PROPRIADES}]**

A visibilidade segue a mesma simbologia das visibilidades para atributos.

O elemento nome corresponde ao nome dado à operação. Se uma operação for pública, ela pode ser ativada com a passagem de uma mensagem para o objeto. Se for protegida, ela só é visível para a classe e para seus descendentes. Se for privativa, somente objetos da própria classe podem executá-la.

Os parâmetros de uma operação correspondem às informações que ela recebe quando é executada. Normalmente, essas informações são fornecidas pelo objeto remetente da mensagem que pede a execução da operação no objeto receptor. Uma operação pode ter zero ou mais parâmetros, que são separados por vírgulas. Cada parâmetro da lista tem a seguinte sintaxe:

**[DIREÇÃO] NOME-PARÂMETRO: TIPO-PARÂMETRO**

O elemento **direção** serve para definir se o parâmetro pode ou não ser modificado pela operação. O modelador pode definir se o parâmetro é de entrada (*IN* ), saída (*OUT* ) ou ambos (*INOUT* ). Se a direção for omitida, o valor assumido pelo parâmetro será IN, de entrada. As possíveis direções na definição de um parâmetro pela UML são::

## ***IN***

Parâmetro de entrada: não pode ser modificado pela operação. Serve somente como informação para o objeto receptor.

## ***OUT***

Parâmetro de saída: pode ser modificado pela operação para fornecer alguma informação ao objeto remetente.

## ***INOUT***

Parâmetro de entrada que pode ser modificado.

O elemento **nome-parâmetro** corresponde ao nome do parâmetro. Cada parâmetro deve ter um nome único dentro da assinatura da operação.

O elemento **tipo-parâmetro** corresponde ao tipo do parâmetro e é dependente da linguagem de programação.

O elemento **tipo-retorno** representa o tipo de dados do valor retornado por uma operação. Esse tipo depende da linguagem de programação.

# **ESPECIFICAÇÃO DE ASSOCIAÇÕES**

No modelo de classes de análise, relacionamentos entre objetos são normalmente definidos apenas com o uso da associação (ou como um de seus casos especiais, a generalização, a agregação ou a composição). As associações são os mecanismos que permitem aos objetos se comunicarem. Elas descrevem a conexão entre diferentes classes. Podem ter uma regra que especifica o propósito da associação e podem ser unidirecionais ou bidirecionais. Cada ponta da associação também possui um valor de multiplicidade que indica como os objetos de

um lado se relacionam com os do outro lado. Existem diversos tipos de associação, que são modelados desde a etapa de análise:

## **UNÁRIAS, REFLEXIVAS OU RECURSIVAS (AUTOASSOCIAÇÃO)**

Representam relacionamentos que ocorrem entre objetos da mesma classe.

## **BINÁRIAS**

Representam relacionamentos que ocorrem entre objetos de duas classes.

## **TERNÁRIAS OU N-ÁRIAS**

Representam relacionamentos que ocorrem entre objetos de mais de duas classes.

## **AGREGAÇÃO**

Representa relacionamentos todo-parte onde a classe do lado parte pode existir independentemente da classe do lado todo.

## **COMPOSIÇÃO**

Representa relacionamentos todo-parte onde a existência da classe do lado parte depende da existência da classe do lado todo.

## **GENERALIZAÇÃO/ ESPECIALIZAÇÃO**

Representa relacionamentos entre um elemento genérico e um ou mais elementos específicos, em que o mais específico possui todas as características do elemento genérico e contém ainda particularidades não encontradas no genérico. Essa característica também é chamada de Herança.

## **CLASSE ASSOCIATIVA**

Representa relacionamentos entre classes cuja multiplicidade é muitos para muitos e possuem propriedades próprias do relacionamento e não das classes associadas.

Na etapa de projeto, uma espécie de associação relevante é a dependência, que representa relacionamentos entre classes, onde uma é dependente da outra. Qualquer modificação na classe independente afetará diretamente objetos da classe dependente.

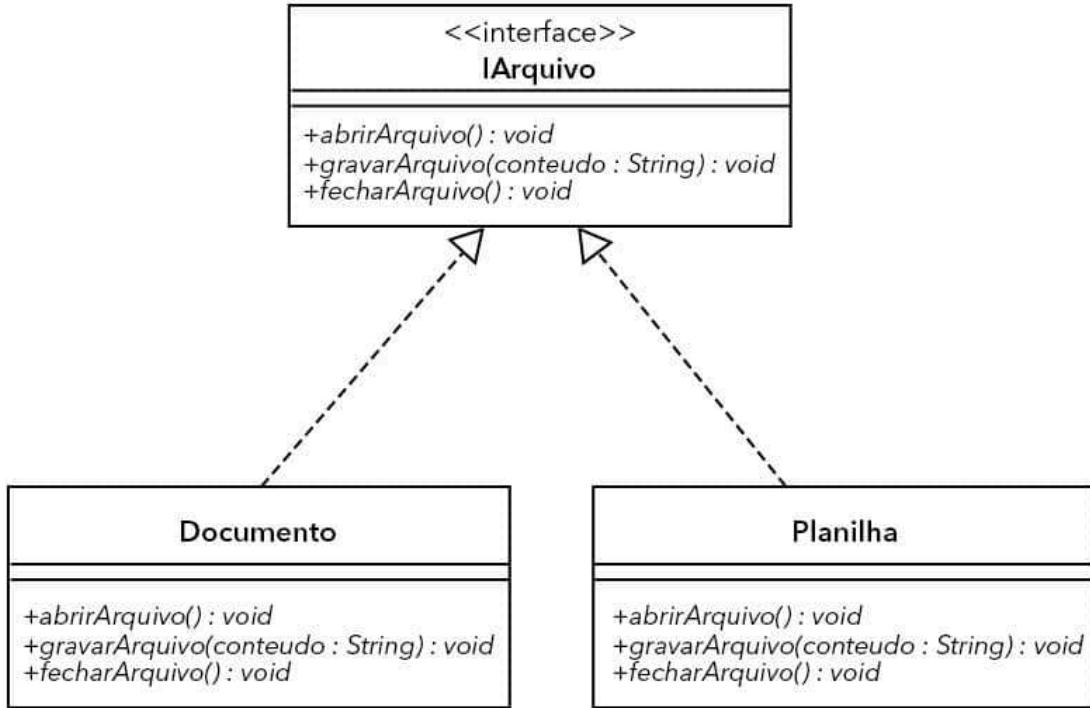


Imagen: Claudia Cappelli, adaptado por Isaac Barbosa

Exemplo de associação de dependência. Elaborado na ferramenta Astah UML.

Uma dependência entre classes indica que uma classe depende dos serviços fornecidos pela outra. No modelo de análise, é suficiente para o modelador identificar a existência de associações entre classes, que é uma forma de dependência. Mas, na fase de especificação do modelo de classes, essa dependência precisa ser mais bem definida pelo projetista, uma vez que ela tem influência na forma utilizada para implementar as classes envolvidas. Vamos descrever, então, detalhes das possíveis formas de dependência. Para isso, considere duas classes, A e B, onde A depende de B.

Vários tipos de dependência entre essas duas classes podem existir. Esses tipos são descritos a seguir:

## DEPENDÊNCIA POR ATRIBUTO

A possui um atributo cujo tipo é B. A dependência por atributo é também chamada de dependência estrutural.

## DEPENDÊNCIA POR VARIÁVEL GLOBAL

A utiliza uma variável global cujo tipo é B.

## **DEPENDÊNCIA POR VARIÁVEL LOCAL**

A possui alguma operação cuja implementação utiliza uma variável local de tipo B.

## **DEPENDÊNCIA POR PARÂMETRO**

A possui pelo menos uma operação, que possui pelo menos um parâmetro, cujo tipo é B.

A notação da UML para representar uma dependência no diagrama de classes é de uma seta tracejada ligando as classes envolvidas. A seta sai da classe dependente e chega na classe da qual depende. Ainda com respeito à notação, as dependências podem ser estereotipadas para indicar o tipo de dependência existente entre duas classes. Os estereótipos da UML para rotular relacionamentos de dependência são:

### **<<GLOBAL>>**

Para variável global

### **<<LOCAL>>**

Para variável local

### **<<PARAMETER>>**

Para parâmetro

# TRANSFORMAÇÃO DE ASSOCIAÇÕES EM DEPENDÊNCIAS

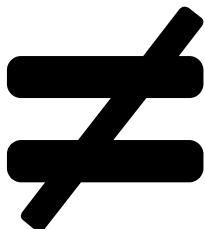
No modelo de classes de análise, o modelador especifica uma ou mais associações entre uma classe e as demais apresentadas. Na passagem do modelo de classes de análise para o de projeto, o modelador deve estudar cada associação para identificar se ela pode ser transformada em dependências. A razão para essa transformação é definir melhor o encapsulamento. Quanto menos dependências estruturais houver no modelo de classes, maior será a qualidade do projeto (do ponto de vista do encapsulamento e do acoplamento das classes constituintes). De um modo geral, as associações que ligam classes de entidade permanecem como associações no modelo de projeto.

## CLASSES PERSISTENTES E TRANSIENTES

As classes são divididas em:

Persistentes

Presumem que seus objetos precisam, de alguma maneira, ser preservados mesmo após o encerramento da utilização do sistema.



Não persistentes (transientes)

Terão seus objetos destruídos durante ou na finalização do uso do sistema. Estas são diferenciadas por estereótipos <<transient>>.

## CLASSES DE INTERFACE

Uma classe de interface é uma coleção de operações com um nome e é utilizada para especificar um tipo de serviço sem especificar como será sua implementação. Essas classes não têm métodos concretos, apenas o declaram para que outra classe possa implementá-lo. Elas possibilitam que objetos externos ao sistema possam colaborar com uma ou mais classes do sistema.

São exemplos as interfaces gráficas, as de interações entre os usuários e as telas do sistema. Há um estereótipo para ela (<<interface>>) e sua implementação utiliza uma reta tracejada oriunda da classe de implementação, contendo a seta vazia na extremidade que aponta para a classe de interface, como ilustra o exemplo a seguir.

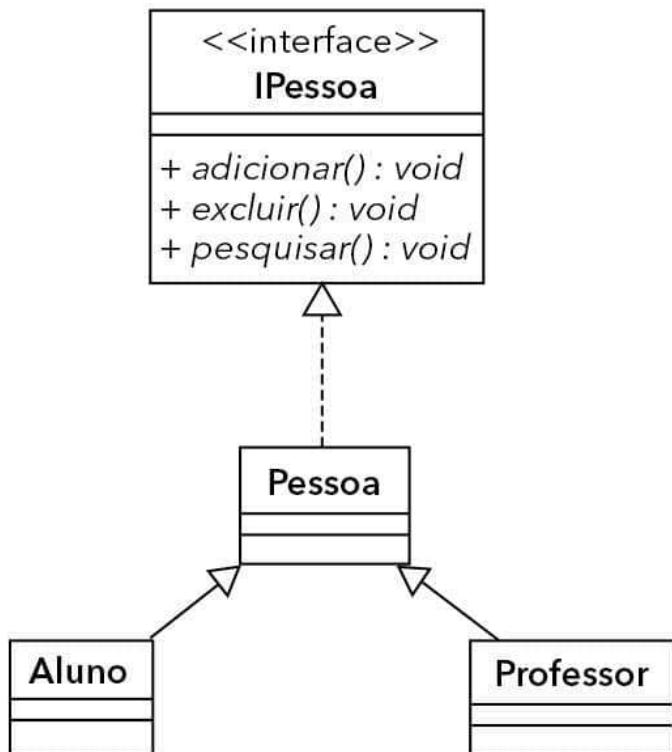


Imagen: Claudia Cappelli, adaptado por Isaac Barbosa

Exemplo de classe de interface. Elaborado na ferramenta Astah UML.

## NAVEGABILIDADE DE ASSOCIAÇÕES

As associações podem ser classificadas em bidirecionais e unidirecionais. Uma associação bidirecional indica que há um conhecimento mútuo entre os objetos associados. Ou seja, se um diagrama de classes exibe uma associação entre duas classes C1 e C2, então as duas assertivas a seguir são verdadeiras:

- CADA OBJETO DE C1 CONHECE TODOS OS OBJETOS DE C2 AOS QUAIS ESTÁ ASSOCIADO.
- CADA OBJETO DE C2 CONHECE TODOS OS OBJETOS DE C1 AOS QUAIS ESTÁ ASSOCIADO.

Graficamente, uma associação unidirecional é representada adicionando-se um sentido à associação. A classe para a qual o sentido aponta é aquela cujos objetos não possuem a visibilidade dos objetos da outra classe.

Durante a construção do modelo de classes de análise, associações são normalmente consideradas “navegáveis” em ambos os sentidos, ou seja, as associações são bidirecionais. No modelo de classes de projeto, o modelador deve refinar a navegabilidade de todas as associações.

## ATENÇÃO

Pode ser que algumas associações precisem permanecer bidirecionais. No entanto, se não houver essa necessidade, recomenda-se transformar a associação em unidirecional. Isso porque uma associação bidirecional é mais difícil de implementar e de manter do que uma associação unidirecional correspondente.

Para entender isso, basta ver que o acoplamento entre as classes envolvidas na associação é maior quando a navegabilidade é bidirecional, prejudicando a modularização. Portanto, um dos objetivos a alcançar durante o projeto é identificar quais navegaibilidades são realmente necessárias.

A definição da navegabilidade se dá em função da troca de mensagens ocorridas nas interações entre objetos do sistema. Mais especificamente, devemos analisar os fluxos das mensagens entre objetos no diagrama de interações. Dados dois objetos associados, A e B, se há pelo menos uma mensagem de A para B em alguma interação, então a associação deve ser navegável de A para B. Da mesma forma, se existir pelo menos uma mensagem de B para A, então a associação deve ser navegável de B para A.

A definição do sentido da navegabilidade é feita em função das mensagens identificadas na modelagem de interação. Em particular, se um objeto do tipo A envia uma mensagem para

outro do tipo B, então deve haver uma navegabilidade no sentido de A para B no diagrama de classes.

Mesmo em situações em que a navegabilidade de uma associação precise ser bidirecional, é possível implementar apenas um de seus sentidos.

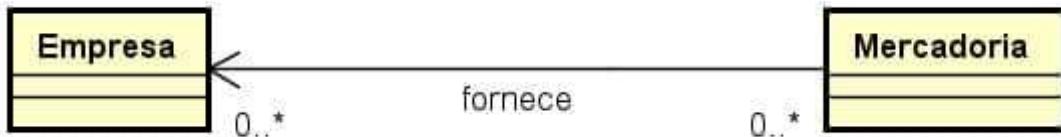


Imagen: Claudia Cappelli

Exemplo de navegabilidade. Elaborado na ferramenta Astah UML.

## VERIFICANDO O APRENDIZADO

### 1. NO DIAGRAMA DE CLASSES, UMA ASSOCIAÇÃO REFLEXIVA OU AUTOASSOCIAÇÃO:

- A) Permite apenas relacionamentos 1:1
- B) Envolve uma relação bilateral entre duas classes.
- C) Representa relacionamentos que ocorrem entre objetos da mesma classe.
- D) Indica que há dependência direta entre duas classes.
- E) Representa relacionamentos entre classes diferentes.

### 2. TEMOS QUATRO TIPOS DE ESPECIFICAÇÃO DE VISIBILIDADE DE ATRIBUTOS: PÚBLICO, PROTEGIDO, PRIVATIVO E PACOTE. ABAIXO LISTAMOS DEFINIÇÕES RELACIONADAS À VISIBILIDADE DE UM ATRIBUTO. ASSINALE A ALTERNATIVA QUE CONTENHA A DEFINIÇÃO CORRETA DO ATRIBUTO PÚBLICO.

- A)** Qualquer atributo visível para classes da subclasse em que foi definido
- B)** Qualquer objeto externo pode obter acesso a qualquer tipo de elemento da classe.
- C)** Qualquer objeto externo pode obter acesso ao atributo, desde que tenha uma referência para a classe externa ao pacote.
- D)** Qualquer objeto externo pode obter acesso ao atributo, desde que tenha uma referência para a classe em que o atributo está definido.
- E)** Qualquer objeto pode obter acesso ao elemento, desde que tenha uma referência para a classe externa deste elemento.

---

## GABARITO

### **1. No diagrama de classes, uma associação reflexiva ou autoassociação:**

A alternativa "**C**" está correta.

Uma associação reflexiva ou recursiva representa relacionamentos que ocorrem entre objetos da mesma classe. Esses relacionamentos recursivos (também chamados de autorrelacionamentos ou autoassociações) são casos especiais onde uma entidade se relaciona com si própria. Os autorrelacionamentos podem ser do tipo 1:1 (um-para-um), 1:N (um-para-muitos) ou N:M (muitos-para-muitos).

### **2. Temos quatro tipos de especificação de visibilidade de atributos: público, protegido, privativo e pacote. Abaixo listamos definições relacionadas à visibilidade de um atributo.**

**Assinale a alternativa que contenha a definição correta do atributo público.**

A alternativa "**D**" está correta.

A visibilidade (+) de um atributo público indica que as propriedades desse atributo são visíveis por qualquer objeto externo, isto é, pertencente a outra classe, contanto que tenha uma referência para a classe em que o atributo está definido.

## MÓDULO 3

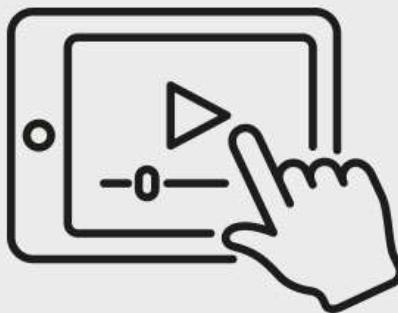
- **Objetivo:** Empregar os diagramas de estado e de atividades no projeto de sistema

## TRANSIÇÃO ENTRE ESTADOS

### EMPREGO DOS DIAGRAMAS DE ESTADOS E DE ATIVIDADES NA ETAPA DE PROJETO

No vídeo a seguir, os componentes do diagrama de estados e do diagrama de atividades são apresentados.

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



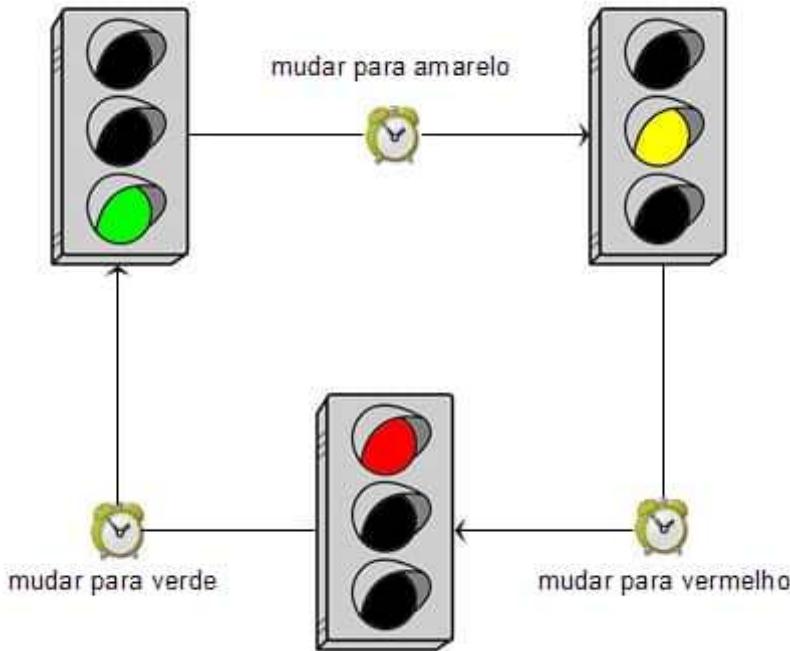


Imagen: Smartkids / Wikimedia Commons / Domínio público

### Exemplo de transição de estado.

Objetos de um sistema de software orientado a objetos se encontram em um estado particular a cada momento. Um objeto muda de estado quando acontece algum evento interno ou externo ao sistema. Quando um objeto muda de um estado para outro, diz-se que ele realizou uma **transição entre estados**. Os estados e as transições de estado de um objeto constituem o seu ciclo de vida.

No momento de sua transição de um estado para outro, um objeto normalmente realiza determinadas ações dentro do sistema. Cada objeto pode passar por um número finito de estados durante a sua vida. Quando um objeto transita de um estado para outro, significa que o sistema no qual ele está inserido também está mudando de estado.

Pela análise das transições entre estados dos objetos de um sistema de software, é possível prever todas as possíveis operações realizadas, em função de eventos que podem ocorrer.

**O DIAGRAMA DA UML UTILIZADO PARA REALIZAR  
ESSA ANÁLISE É O DIAGRAMA DE ESTADOS,  
TAMBÉM CONHECIDO COMO DIAGRAMA DE MÁQUINA  
DE ESTADO OU DE TRANSIÇÃO DE ESTADO (DTE).**

Esse diagrama permite descrever o ciclo de vida de objetos de uma classe, os eventos que causam a transição de um estado para outro e a realização de operações resultantes.

## ► ATENÇÃO

Assim como diagramas de interação não são construídos para todos os casos de uso, os diagramas de transição de estado não devem ser construídos para todas as classes de um sistema, mas apenas para aquelas que possuem um número definido de estados conhecidos, e quando o comportamento das classes de objetos é afetado e modificado pelos diferentes estados. Nessas classes, um DTE pode ser utilizado para enfatizar os eventos que resultam em mudanças de estado.

## DIAGRAMA DE TRANSIÇÃO DE ESTADOS

A UML tem um conjunto rico de notações para desenhar um DTE. Os elementos básicos de um diagrama de transição de estados são os estados e as transições. Associados a estas últimas estão os conceitos de **evento**, **ação** e **atividade**. Um DTE pode também conter elementos menos utilizados, mas às vezes úteis, como **transições internas**, **estados aninhados** e **estados concorrentes**. A figura a seguir mostra um diagrama de estados para o objeto oferta de disciplina, que pode ter quatro estados: **aberta**, **cancelada**, **lotada** e **fechada**.

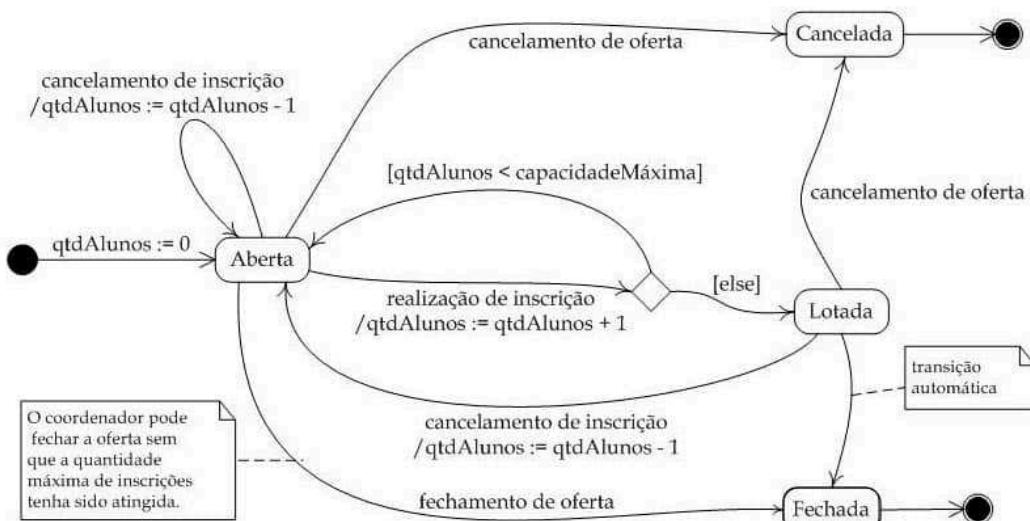


Imagen: BEZERRA, 2015. p. 225. Adaptado por Isaac Barbosa

Exemplo de diagrama de estados.

Um estado é uma situação na vida de um objeto durante a qual ele satisfaz alguma condição ou realiza alguma atividade. Cada estado de um objeto é normalmente determinado pelos valores dos seus atributos e (ou) pelas suas ligações com outros objetos.

A UML utiliza as seguintes notações para representar os estados:



Imagen: Isaac Barbosa

## ESTADO

O retângulo com as bordas arredondadas representa um estado. Em sua forma mais simples, o retângulo de um estado possui um único compartimento.



Imagen: Isaac Barbosa

## ESTADO INICIAL

O círculo preenchido indica o estado de um objeto quando ele é criado. Só pode haver um estado inicial em um DTE, restrição que serve para definir a partir de que ponto um DTE deve começar a ser lido.



Imagen: Isaac Barbosa

## ESTADO FINAL

Representado por um círculo parcialmente preenchido. Diferentemente do estado inicial, pode haver vários estados finais, dependendo dos estados terminais do objeto modelado.

## TRANSIÇÕES

Os estados estão associados a outros pelas transições. Uma transição é mostrada como uma linha conectando estados, com uma seta apontando do estado origem para o estado destino. Quando ocorre uma transição entre estados, diz-se que a transição foi disparada. Note que, em uma transição, o estado subsequente pode ser igual ao estado original. Uma transição pode ser rotulada com uma expressão:

## EVENTO (LISTA-PARÂMETROS) [GUARDA] / [AÇÃO]

A seguir, vamos entender os elementos envolvidos nas transições entre estados:

### EVENTO

Um evento é algo que acontece em algum ponto no tempo e que pode modificar o estado de um objeto, provocando a transição de estado. Um evento pode conter uma lista de parâmetros que são passados para fornecer informações úteis ao objeto receptor de evento. A lista de

parâmetros é especificada entre parênteses. Na figura **exemplo de diagrama de estados**, *realização de inscrição / qtdAlunos := qtdAlunos + 1* é um exemplo de evento.

Os eventos relevantes a uma classe de objetos podem ser classificados em quatro tipos:

**Evento de chamada:** recebimento de uma mensagem de outro objeto. Pode-se pensar nesse tipo de evento como uma solicitação de serviço de um objeto a outro.

**Evento de sinal:** recebimento de um sinal. Este é um tipo especial do evento anterior. Nesse evento, o objeto recebe um sinal de outro objeto que pode fazê-lo mudar de estado. A diferença básica entre o evento de sinal e o evento de chamada é que neste último o objeto que envia a mensagem fica esperando a execução dela. No evento de sinal, o objeto remetente continua o seu processamento após ter enviado o sinal. O evento de sinal raramente é utilizado.

**Evento temporal:** passagem de um intervalo de tempo predefinido. Em vez de receber uma mensagem específica, um objeto pode interpretar a passagem de um certo intervalo de tempo como sendo um evento.

**Evento de mudança:** uma condição que se torna verdadeira. O fato de determinada condição se tornar verdadeira também pode ser visto como um evento.

## CONDIÇÃO DE GUARDA

Uma transição pode ter também uma condição de guarda. Uma condição de guarda, ou simplesmente guarda, é uma expressão de valor lógico, da mesma forma que a utilizada para diagramas de interação. A condição de guarda de uma transição pode ser definida utilizando-se parâmetros passados no evento e atributos e referências a ligações da classe em questão. Além disso, uma condição também pode testar o valor de um estado. Uma transição na qual foi definida uma condição de guarda é disparada somente se o evento associado ocorre e a condição de guarda é verdadeira. Se uma transição não tiver uma condição de guarda, ela sempre será disparada quando o evento ocorrer. É importante não confundir a condição de guarda de uma transição com um evento de mudança (que também é definido com uma expressão de valor lógico). A expressão condicional de uma condição de guarda é sempre apresentada entre colchetes, como em *[qtdAlunos < capacidadeMáxima]* e *[else]* , visto no **exemplo de diagrama de estados**.

# AÇÕES

Ao transitar de um estado para outro, um objeto pode realizar uma ou mais ações. Uma ação é uma expressão que pode ser definida em termo dos atributos, das operações ou das associações da classe. Os parâmetros do evento também podem ser utilizados. Uma ação pode corresponder igualmente à execução de uma operação. A ação é representada na linha da transição e deve ser precedida por uma barra inclinada para a direita (símbolo “/”). Note que a ação associada a uma transição é executada somente se a transição for disparada. No **exemplo de diagrama de estados**, são exemplos de ações: `/ qtdAlunos := qtdAlunos + 1` e `/ qtdAlunos := qtdAlunos - 1`.

# ATIVIDADES

Semelhante a uma ação, uma atividade é algo que é executado pelo objeto. No entanto, uma atividade pode ser interrompida (considera-se que o tempo de execução de uma ação é tão insignificante que ela não pode ser interrompida). Por exemplo, enquanto a atividade estiver em execução, pode acontecer um evento que a interrompa e cause uma mudança de estado. Outra diferença entre ação e atividade é que uma atividade sempre está associada a um estado (ao contrário, uma ação está associada a uma transição).

# PONTO DE JUNÇÃO

Em algumas situações, o próximo estado de um objeto varia de acordo com o valor da condição de guarda. Se o valor da condição de guarda for verdadeiro, o objeto vai para um estado E1; se for falso, o objeto vai para outro estado E2. É como se a transição tivesse bifurcações e cada transição de saída da bifurcação tivesse uma condição de guarda. Essa situação pode ser representada em um DTE por um ponto de junção. Um ponto de junção é desenhado como um losango em que chegam uma ou mais transições (provenientes de estados diferentes) e de onde partem uma ou mais transições. A cada transição de saída do ponto de junção está associada uma condição de guarda. A transição que o objeto segue é aquela para a qual a condição de guarda é verdadeira. Note que há um ponto de junção no **exemplo de diagrama de estados**.

# ESTADOS ANINHADOS

Podem existir estados aninhados dentro de outro estado. Um estado que contém diversos outros é dito composto. Todos os estados dentro de um estado composto herdam qualquer transição deste último. Isso significa que o uso de estados compostos geralmente torna um DTE mais legível.

# ESTADOS CONCORRENTES

Um estado concorrente é um tipo especial de estado composto. Um objeto em um estado concorrente pode, na verdade, se encontrar em dois ou mais estados independentes.

## IDENTIFICAÇÃO DOS ELEMENTOS DE UM DIAGRAMA DE ESTADOS

Os estados podem ser vistos como uma abstração dos atributos e das associações de um objeto. A seguir temos alguns exemplos (os nomes em **negrito** são possíveis estados).

Um professor está **licenciado** quando não está ministrando curso algum durante o semestre.

Uma oferta de disciplina está **aberta** quando a capacidade máxima não for alcançada; caso contrário, a oferta estará **lotada**.

Um professor está **licenciado** quando não está ministrando curso algum durante o semestre.

Um pedido está **atendido** quando todos os seus itens estão entregues.

Um bom ponto de partida para identificar estados de um objeto é analisar os possíveis valores de seus atributos e as ligações que ele pode realizar com outros objetos. No entanto, a existência de atributos ou ligações não é suficiente para justificar a criação de um DTE para uma classe. O comportamento de objetos dessa classe deve depender de tais atributos ou ligações. Em relação a transições, devemos identificar os eventos que podem gerá-las. Além disso, é preciso examinar também se há algum fator que condicione o disparo da transição. Se existir, esse fator deve ser modelado como uma condição de guarda da transição. Já que as transições dependem de eventos para ocorrer, devem-se identificar esses eventos primeiramente.

Outro bom ponto de partida é a descrição dos casos de uso. Os eventos encontrados na descrição dos casos de uso são externos ao sistema. Contudo, uma transição pode também ser disparada por um evento interno ao sistema. Para identificar os eventos internos relevantes a um objeto, analise os **diagramas de interação**. Esses diagramas contêm mensagens sendo trocadas entre objetos e mensagens, que nada mais são do que solicitações de um objeto a outro. Essas solicitações podem ser vistas como eventos. De uma forma geral, cada operação com visibilidade pública de uma classe pode ser vista como um evento em potencial.

## DICA

Outra fonte para identificação de eventos relevantes é analisar as regras de negócio definidas para o sistema. Normalmente, essas regras possuem informações sobre condições limites que permitem identificar estados e transições.

# CONSTRUÇÃO DE DIAGRAMAS DE TRANSIÇÕES DE ESTADOS

Para sistemas bastante simples, a definição dos estados de todos os objetos não é tão trabalhosa. No entanto, a quantidade de estados possíveis de todos os objetos de um sistema complexo é grande. Isso torna a construção de um DTE para o sistema inteiro uma tarefa bastante complexa.

Para resolver o problema da explosão exponencial de estados, os diagramas de estados são desenhados por classe.

Geralmente, cada classe é simples o suficiente para que o diagrama de estados correspondente seja compreensível. Note, contudo, que essa solução de dividir a modelagem de estados por classes do sistema tem a desvantagem de dificultar a visualização do estado do sistema como um todo. Essa desvantagem é parcialmente compensada pela construção de diagramas de interação.

Nem todas as classes de um sistema precisam de um DTE. Um diagrama de estados é desenhado somente para classes que exibem um comportamento dinâmico relevante. A relevância do comportamento de uma classe depende muito de cada situação em particular. No entanto, objetos cujo histórico precisa ser rastreado pelo sistema são típicos para se construir um diagrama de estados. Uma vez selecionadas as classes para cada uma das quais se deve construir um diagrama de estados, acompanhe essa sequência de passos:

1

Identifique os estados relevantes para a classe.

Identifique os eventos relevantes aos objetos de uma classe. Para cada evento, identifique qual transição que ele ocasiona.

---

**2**

---

**3**

---

Para cada estado, identifique as transições possíveis quando um evento relevante ocorre.

---

Para cada estado, identifique os eventos internos e as ações correspondentes relevantes.

---

**4**

---

**5**

---

Para cada transição, verifique se há fatores que influenciam o seu disparo. Se esse for o caso, uma condição de guarda deve ser definida.

---

Verifique também se alguma ação deve ser executada quando uma transição é disparada (ações).

---

**6**

---

## 7

Para cada condição de guarda e para cada ação, identifique os atributos e as ligações que estão envolvidos. Pode ser que esses atributos ou ligações ainda não existam. Nesse caso, eles devem ser adicionados ao modelo de classes.

---

Defina o estado inicial e os eventuais estados finais.

## 8

---

## 9

Desenhe o diagrama de estados. Procure posicionar 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.

## ATENÇÃO

A construção de diagramas de estados de um sistema frequentemente leva à descoberta de novos atributos para uma classe, principalmente atributos que servem de abstrações para estados. Além disso, esse processo de construção permite identificar novas operações na classe, pois os objetos precisam reagir aos eventos que eles recebem, e essas reações são realizadas mediante operações.

# DIAGRAMA DE ATIVIDADES

Um diagrama de atividade pode ser considerado um tipo especial de diagrama de estados, em que são representados os estados de uma atividade em vez dos estados de um objeto. Ao

contrário dos diagramas de estados, que são orientados a eventos, diagramas de atividade são orientados a fluxos de controle.

Muitos autores dizem que o diagrama de atividades é um fluxograma. Na verdade, o diagrama de atividade pode ser visto como uma extensão dos fluxogramas. Além de possuir toda a semântica existente em um fluxograma (com notação ligeiramente diferente), o diagrama de atividade possui notação para representar ações concorrentes (paralelas), juntamente com a sua sincronização.

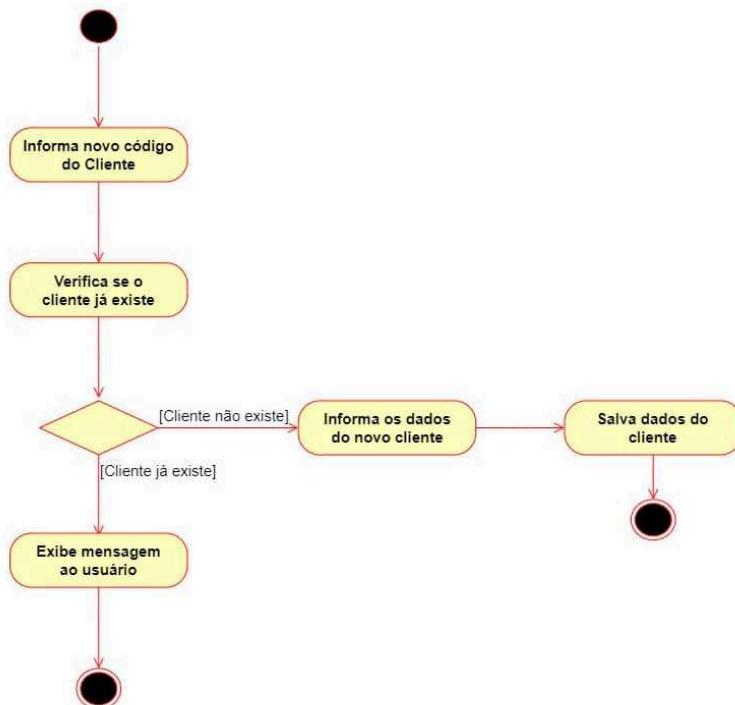
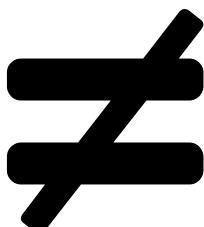


Imagen: Claudia Cappelli, adaptado por Isaac Barbosa

Exemplo de diagrama de atividades. Elaborado na ferramenta Astah UML.

Os elementos de um diagrama de atividade podem ser divididos em dois grupos:

Elementos utilizados para representar fluxos de controle sequenciais.



Elementos utilizados para representar fluxos de controle paralelos.

Veremos mais detalhes de cada tipo a seguir.

# FLUXO DE CONTROLE SEQUENCIAL

Os elementos de um diagrama de atividade usados no controle sequencial são:

## ESTADO AÇÃO E ESTADO ATIVIDADE

Um diagrama de atividades exibe os passos de uma computação. Cada estado corresponde a um dos passos da computação, em que o sistema está realizando algo. Um estado em um diagrama de atividade pode ser um estado atividade ou um estado ação. O primeiro leva um certo tempo para ser finalizado. Já o segundo é realizado instantaneamente.

## ESTADOS INICIAL E FINAL E CONDIÇÃO DE GUARDA

Assim como no diagrama de estados, um diagrama de atividade deve ter um estado inicial; ele pode ter também vários estados finais e guardas associados a transições. Um diagrama de atividade pode não ter estado final, o que significa que o processo ou procedimento sendo modelado é cíclico.

## TRANSIÇÃO DE TÉRMINO

Uma transição de término liga um estado a outro. Essa transição significa o término de um passo e o consequente início do outro. Observe que, em vez de ser disparada pela ocorrência de um evento (como nos diagramas de estados), essa transição é disparada pelo término de um estado de ação. Pode haver uma transição rotulada com a condição de guarda especial `[else]`, o que significa que, se todas as demais condições de guarda foram avaliadas como falsas, a transição associada a essa guarda especial é disparada.

## PONTOS DE RAMIFICAÇÃO

Um ponto de ramificação possui uma única transição de entrada e várias de saída. Para cada transição de saída, há uma condição de guarda associada. Quando o fluxo de controle chega a um ponto de ramificação, uma e somente uma das condições de guarda deve ser verdadeira.

## PONTO DE UNIÃO

Um ponto de união reúne diversas transições que, direta ou indiretamente, têm um ponto de ramificação em comum.

# FLUXO DE CONTROLE PARALELO

Um diagrama de atividade pode conter fluxos de controle paralelos. Isso significa que é possível haver dois ou mais fluxos de controle sendo executados simultaneamente em um diagrama de atividades. Para sincronizar dois ou mais fluxos paralelos, as barras de sincronização são utilizadas. Há dois tipos de barra de sincronização:

## BARRA DE BIFURCAÇÃO (*FORK* )

Uma barra de bifurcação recebe uma transição de entrada e cria dois ou mais fluxos de controle paralelos. A partir desse momento, cada um dos fluxos criados é executado independentemente e em paralelo com os demais.

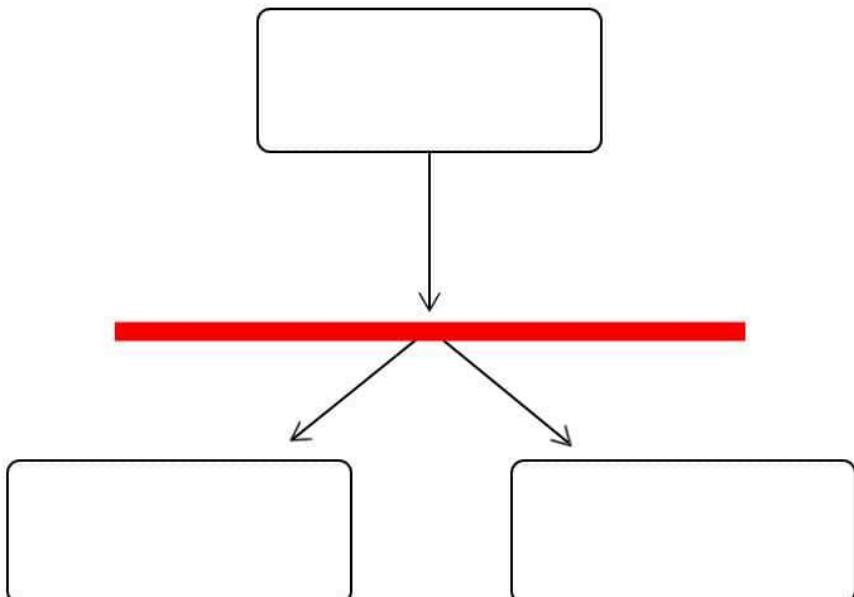
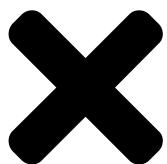


Imagen: Isaac Barbosa



## BARRA DE JUNÇÃO (*JOIN* )

Uma barra de junção recebe duas ou mais transições de entrada e une os fluxos de controle em um único fluxo. Essa barra tem o objetivo de sincronizar fluxos de controle paralelos criados

anteriormente no diagrama. As transições de saída da barra de junção somente são disparadas quando todas as transições de entrada tiverem sido disparadas.

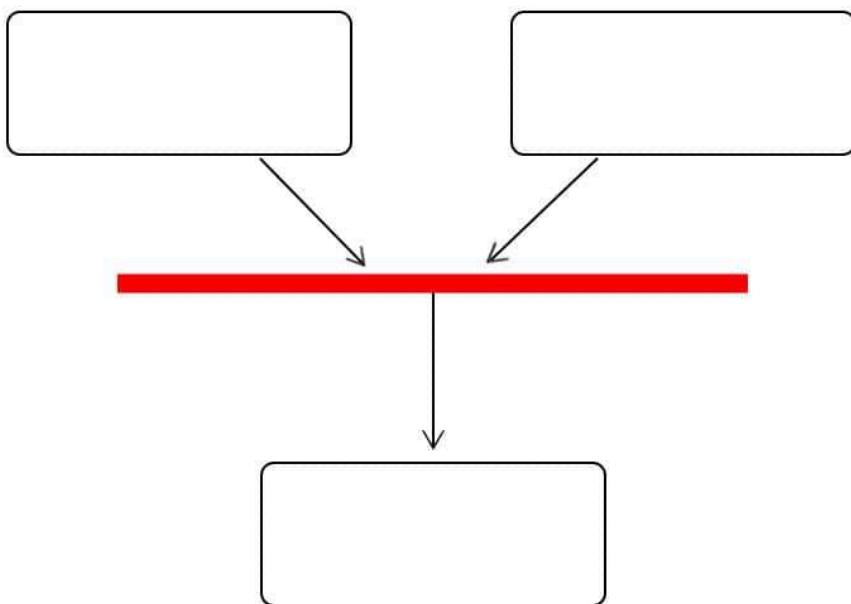


Imagen: Isaac Barbosa

## RAIAS

Algumas vezes, as atividades de um processo podem ser distribuídas por vários agentes que o executarão. Isso normalmente ocorre em processos de negócio de uma organização, onde as tarefas são executadas por pessoas ou departamentos diversos. Nesses casos, o processo pode ser representado em um diagrama de atividades com o uso de raias de natação (Tradução para swimlanes) . As raias de natação dividem o diagrama de atividades em compartimentos. Cada compartimento contém atividades que são realizadas por um agente específico. Além disso, as entidades de cada compartimento podem estar realizando atividades em paralelo.

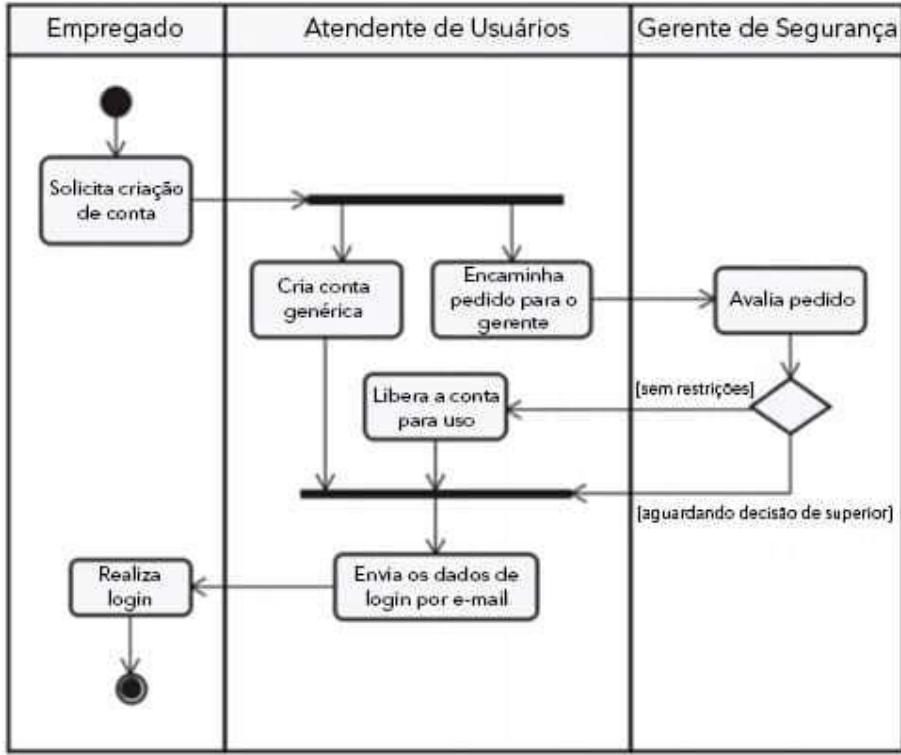


Imagen: Claudia Cappelli, adaptado por Isaac Barbosa

Exemplo de diagrama de atividades com raias e *fork*. Elaborado na ferramenta Astah UML.

## MODELAGEM DA LÓGICA DE UMA OPERAÇÃO COMPLEXA

Quando um sistema de software é adequadamente decomposto em seus objetos constituintes e as responsabilidades de cada objeto estão bem definidas, a maioria das operações é bastante simples. Essas operações não necessitam de modelagem gráfica para serem entendidas. No entanto, em alguns casos, notadamente quando uma operação de uma classe de controle implementa uma regra de negócio, pode haver a necessidade de se descrever a lógica dessa operação ou da própria regra de negócio. Diagramas de atividade também podem ser utilizados com esse objetivo, nisso se assemelhando a um fluxograma.

## VERIFICANDO O APRENDIZADO

**1. UM DOS DIAGRAMAS APRESENTADOS É O DIAGRAMA DE TRANSIÇÃO DE ESTADOS. ASSINALE ABAIXO A ALTERNATIVA QUE DEFINE CORRETAMENTE O QUE É UM ESTADO:**

- A)** História de vida de um objeto.
- B)** Descrição das atividades realizadas por um objeto.
- C)** Situações e/ou condições na vida de um objeto.
- D)** Cenário onde está inserido um objeto.
- E)** Elemento que define o objeto.

**2. NESTE MÓDULO, APRESENTAMOS O DIAGRAMA DE ATIVIDADE. ESSE DIAGRAMA UTILIZA BARRAS DE SINCRONIZAÇÃO. ASSINALE A ALTERNATIVA QUE CONTENHA OS DOIS TIPOS DE SINCRONIZAÇÃO EXISTENTES:**

- A)** Bifurcação e junção.
- B)** Transição e criação.
- C)** Bifurcação e criação.
- D)** Junção e criação.
- E)** Transição e junção.

---

**GABARITO**

**1. Um dos diagramas apresentados é o diagrama de transição de estados. Assinale abaixo a alternativa que define corretamente o que é um estado:**

A alternativa "**C**" está correta.

Um objeto pode mudar de estado ao longo de seu ciclo de vida, o qual é caracterizado por situações ou condições resultantes da ocorrência de eventos.

**2. Neste módulo, apresentamos o diagrama de atividade. Esse diagrama utiliza barras de sincronização. Assinale a alternativa que contenha os dois tipos de sincronização existentes:**

A alternativa "A" está correta.

No diagrama de atividades, podem ocorrer fluxos paralelos que requerem sincronização, representada por uma barra. Esta pode ser de dois tipos: bifurcação (fork), quando uma transição de entrada cria dois ou mais fluxos de saída paralelos, e junção (join) quando duas ou mais transições de entrada são unidas resultando num único fluxo de saída.

## MÓDULO 4

---

● **Objetivo:** Empregar os diagramas de componentes e de implantação

## DIAGRAMA DE COMPONENTES

## VISÃO GERAL DOS DIAGRAMAS DE COMPONENTES E DE IMPLANTAÇÃO

Os componentes do diagrama de componentes e do diagrama de implantação são apresentados no vídeo a seguir.

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



O diagrama de componentes apresenta o relacionamento entre os diferentes componentes de um sistema de software. Na UML, o termo "componente" refere-se a um módulo que representa sistemas ou subsistemas com capacidade de interagir. Para isso, existe uma abordagem de desenvolvimento em torno de componentes.

Nesse diagrama, são representados os diferentes componentes de software necessários para que o sistema funcione corretamente.

Em uma abordagem de programação orientada a objetos, o desenvolvedor usa o diagrama de componentes para agrupar classes com base em um objetivo comum. Assim, é função do diagrama de componentes documentar como os componentes estão estruturados, organizados e como se relacionam, permitindo uma melhor compreensão e facilitando a sua reutilização. Também serve para modelar arquivos de processos de negócio, descrevendo as partes que participam dele e suas relações.

Um diagrama de componentes suporta tanto a especificação dos componentes lógicos, como, por exemplo, componentes de negócio, de processo, entre outros. Os principais benefícios na construção desse diagrama são:

Permitir visualizar a estrutura lógica e física do sistema.

Identificar os componentes do sistema e como eles se relacionam.

Perceber o comportamento do serviço quanto à interface.

## COMPONENTE

Um componente nada mais é do que uma caixa preta (artefato do sistema) que possui vida autônoma e que, somente por meio de suas interfaces, oferece ou requer serviços de outras caixas pretas, sem que para isso tenha que conhecer seus conteúdos. Um componente é

modelado ao longo do ciclo de vida de desenvolvimento, incluindo a concepção da funcionalidade do sistema por meio de casos de uso, definição das classes, entre outros.

Na UML, os componentes também fazem uso de estereótipos. Os mais utilizados são arquivos, tabela, banco de dados, executável e biblioteca. Assim sendo, podemos dizer que um componente é qualquer arquivo que seja necessário à execução do sistema de informação.



Imagen: Isaac Barbosa

#### ▢ Representação de um componente

Os componentes são elementos de alto nível que permitem o agrupamento de várias interfaces sem considerá-las um pacote. Cada componente é representado por um retângulo contendo um nome que deve traduzir sua função. Se necessário, pode conter um estereótipo. No canto superior do retângulo, há um ícone que é também um retângulo com dois menores sobrepostos.

Os componentes também podem ser descritos por meio da visão externa usando símbolos de interface colados na caixa do componente. Nesse caso, é interessante anexar uma máquina de estados para interfaces, portas e para o próprio componente para permitir uma descrição mais precisa.

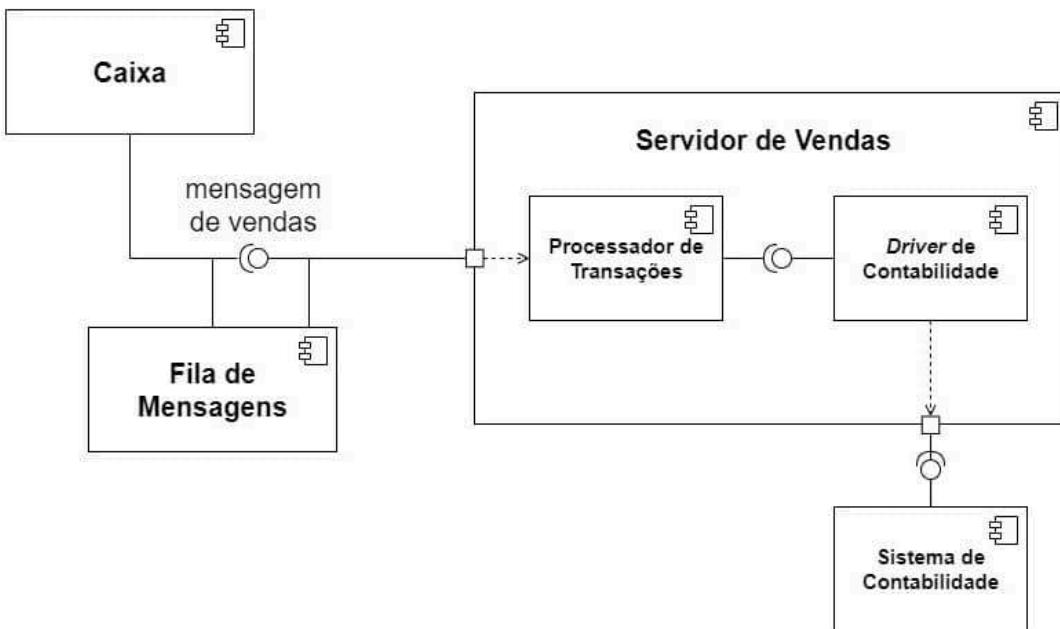


Imagen: Claudia Cappelli, adaptado por Isaac Barbosa

Exemplo de diagrama de componentes. Elaborado na ferramenta Astah UML.

Quando mais detalhes são necessários para a perfeita especificação de um componente, podemos definir uma estrutura interna de partes e conectores, usando o que podemos chamar de visão interna ou “caixa branca”. Nessa estrutura, geralmente estão contidos outros componentes. Esses componentes podem ser implementados por diferentes tipos de arquivo. No diagrama de componentes, esses tipos podem ser destacados por meio de estereótipos, representados por rótulos acima do nome do componente. Nas diretrizes da UML definidas pela OMG (Object Management Group), um artefato é dito como sendo definido pelo usuário e representa um elemento concreto do mundo físico. Ele pode ter propriedades ou operações e pode ser instanciado ou associado com outros artefatos.

## INTERFACES

Interfaces são coleções de operações que especificam serviços de um componente. É por meio delas que os componentes se comunicam com o mundo externo, seja para oferecer ou receber serviços. Interfaces descrevem o comportamento visível externamente e não especificam qualquer estrutura (não podem incluir atributos), nem qualquer implementação.

As interfaces fazem as associações entre os componentes do software.

As interfaces podem ser chamadas quando requerem ou esperam serviços. Podem ser opcionalmente por meio de portas onde é possível haver informações adicionais, tais como

requisito de desempenho ou políticas que determinem que a interface seja realizada de maneira adequada.

## CONECTORES

São utilizados nos diagramas de componentes para incluir interfaces baseadas em restrições e notações. Existem dois tipos principais de conectores:

### CONECTOR ASSEMBLY

Conecta dois componentes para definir que um deles fornece os serviços que o outro requer.

### CONECTOR DELEGATE

Indica que um sinal que chega para um componente será transferido para o outro para tratamento ou manipulação. Isso significa que quando colocamos um conector de delegação, estamos dizendo que naquela instância do componente o trabalho não será realizado, mas sim por outro componente. Esse tipo de conector é bastante usado para modelar decomposição hierárquica em serviços.

## CAMADAS

A arquitetura de sistemas em camadas permite isolar a camada de apresentação das de regras de negócio e de armazenamento de dados. Isso permite aumentar o desempenho do sistema, além de facilitar a manutenção, atualização e substituição de componentes. Isso é muito interessante nos sistemas atuais que se caracterizam muitas vezes por atender grande número de usuários e áreas de negócio.

Um exemplo clássico de arquitetura em camadas é o padrão de projeto MVC (*Model – View – Controller*) , em que os componentes da aplicação consistem na interface (*View*) que invoca o controlador (*Controller*) , o qual, por sua vez, acessa o banco de dados (*Model*) .

Essa separação das camadas de software permite a modularização e o reuso dos componentes do sistema. Um exemplo é ilustrado na figura a seguir, em uma aplicação

hipotética (note que diversas outras formas de componentização poderiam ser adotadas, a critério do projetista).

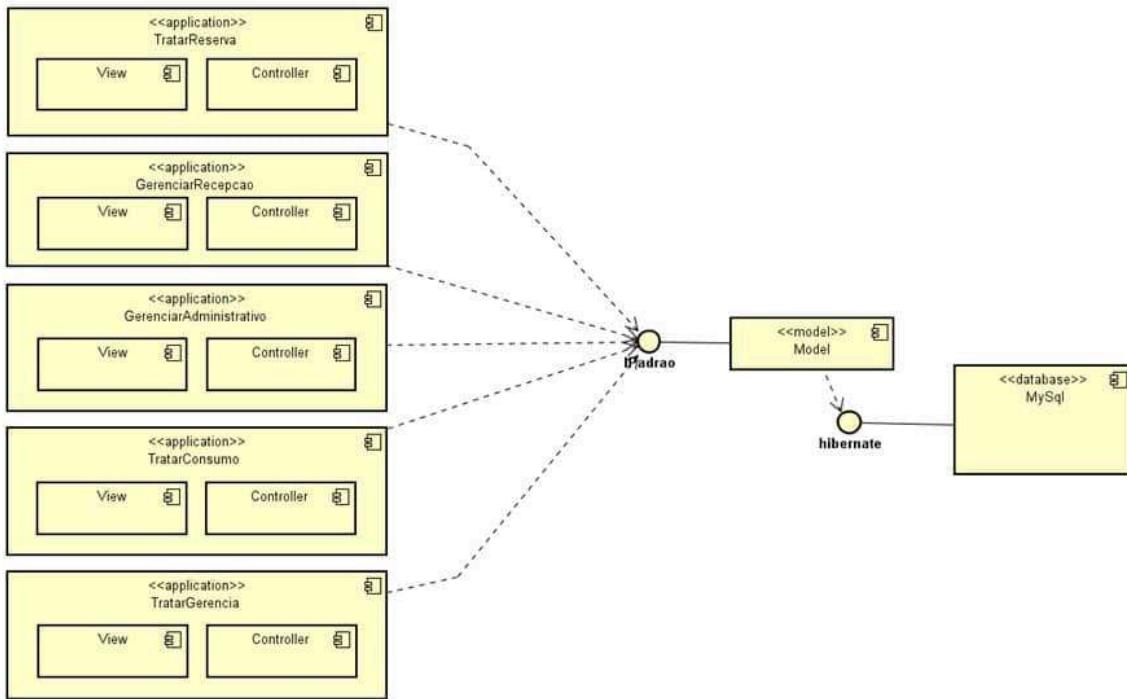


Imagen: Claudia Cappelli

Exemplo de diagrama de componentes usando MVC. Elaborado na ferramenta Astah UML.

## DIAGRAMA DE IMPLANTAÇÃO

O diagrama de implantação especifica um conjunto de artefatos de infraestrutura que podem ser utilizados para definir a arquitetura física de execução de sistemas de informação. Ele foca a organização da arquitetura sobre a qual o software irá ser implantado e executado em termos de hardware, software básico e redes de comunicação, ou seja, quais máquinas (computadores, servidores, switches etc.), quais programas de software básico (sistema operacional, sistema gerenciador de banco de dados, browser) serão necessários para suportar o sistema, bem como definir como essas máquinas serão conectadas, com qual velocidade de conexão e quais protocolos de comunicação são utilizados.

O diagrama de implantação descreve a implementação física de informações geradas pelo programa de software em componentes de hardware. Na UML, um diagrama de implantação representa a estrutura física do sistema em si, o que, assim como o diagrama de componentes, pode ser feito de várias formas.

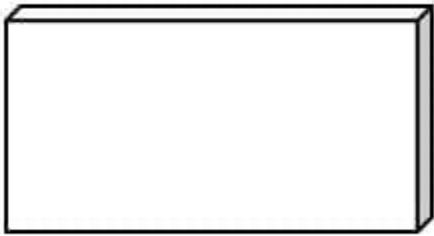


Imagen: Isaac Barbosa

Caixas tridimensionais representam os elementos básicos de software básico ou hardware, ou nós no sistema.

---

Imagen: Isaac Barbosa

As linhas de nós a nós indicam relacionamentos.



Imagen: Isaac Barbosa

As formas menores contidas dentro das caixas tridimensionais representam os artefatos de software contidos nos nós.

## NÓS DO DIAGRAMA DE IMPLANTAÇÃO

São definidos de maneira aninhada e representam dispositivos de hardware e ambientes de execução de software. São os elementos mais básicos do diagrama de implantação; podem representar computadores pessoais e/ou servidores de bancos de dados, de aplicação ou de internet, sistemas operacionais, browser, sistemas gerenciadores de bancos de dados etc. Eles

podem ser compostos de outros nós, sendo comum encontrar um nó que representa um item de hardware contendo outros nós que representam ambientes de execução.

O nó representa um elemento físico capaz de oferecer recursos computacionais e, geralmente, possui pelo menos memória e processador. Também serve para representar topologias de rede por meio de ligações entre instâncias de nós. Podem ser conectados para representar uma topologia de redes usando caminhos de comunicação que podem ser definidos como servidor de aplicação, servidor web, entre outros.

## ⚠ ATENÇÃO

A elaboração do diagrama de implantação depende do porte do sistema a ser desenvolvido. Por exemplo, não faz muito sentido para um sistema que será executado dentro de um único computador. Em outros casos, porém, ele pode ser muito útil, principalmente para ser utilizado como meio de comunicação entre a equipe de desenvolvimento e de infraestrutura, pois servirá de base para a equipe de infraestrutura instalar e configurar servidores, gerenciadores de bancos de dados, entre outros.

A figura a seguir ilustra um diagrama de implantação.

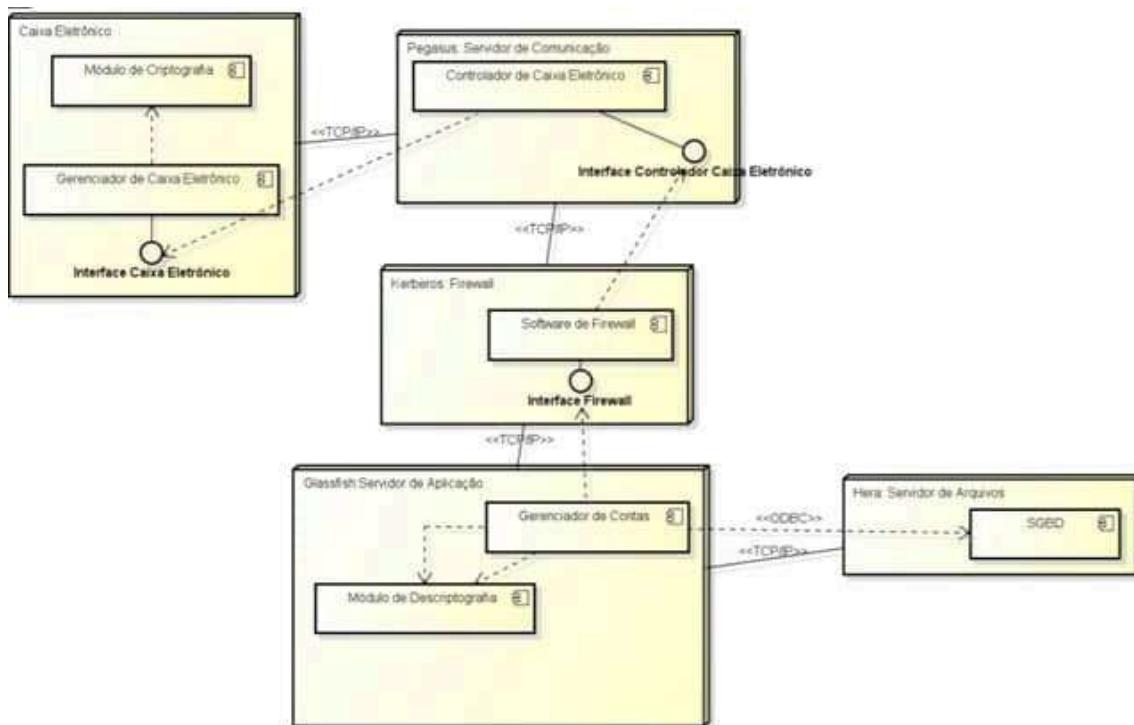


Imagen: Claudia Cappelli

Exemplo de diagrama de implantação. Elaborado na ferramenta Astah UML.

Podem ser usados estereótipos que são, como em outros diagramas da UML, uma maneira gráfica ou textual de diferenciar componentes dos diagramas (classes, casos de uso, associações etc.). Podem ser representados por meio de textos escritos entre dois sinais de maior e menor quando não modificam o modelo original do componente (<>) e, também, por meio de um novo componente visual, porém padronizado pela UML, como os nós que podem conter os seguintes estereótipos: <<device>>, <<artifact>>, <<execution environment>>.

## DEVICE

Representa um dispositivo computacional com capacidade de processamento em que artefatos podem ser implantados para execução. Os dispositivos podem ser complexos, isto é, podem ser constituídos por outros dispositivos. Para representar um device, utilizamos o estereótipo <<device>>.

## ARTEFATOS

Representam objetos concretos do mundo físico, usados e/ou resultantes de um processo de desenvolvimento. São exemplos de artefatos: programas fontes e executáveis, uma tabela de bancos de dados, scripts, um documento do sistema, uma mensagem de correio eletrônico etc. Para representar um artefato, é utilizado o estereótipo <<artifact>>. Artefatos são sempre implantados em nós.

## AMBIENTE DE EXECUÇÃO

Representam uma subclasse de um nó. São exemplos de ambiente de execução: sistemas operacionais, sistemas gerenciadores de bancos de dados etc. Para representar um ambiente de execução utilizamos o estereótipo <<execution environment>>.

## ASSOCIAÇÃO

Os nós são conectados por meio de caminhos de comunicação para criar sistemas em rede, ou seja, possuem ligações entre si de forma que possam se comunicar e trocar informações. Essas ligações são chamadas de associações e são representadas por segmentos de reta ligando um nó a outro. Uma associação pode conter estereótipos utilizados para determinar, por exemplo, o tipo de protocolo de comunicação usado entre os nós.

## VERIFICANDO O APRENDIZADO

**1. APRENDEMOS QUE O DIAGRAMA DE COMPONENTES DOCUMENTA ARQUIVOS FÍSICOS ARMAZENADOS EM MEIO DIGITAL, REPRESENTANDO DIVERSOS COMPONENTES DE SOFTWARE DO SISTEMA. ASSINALE A ALTERNATIVA QUE INDICA DE FORMA MAIS COMPLETA O QUE É EXPLICITADO NESSE DIAGRAMA.**

- A) Reuso e relações de arquivos.**
- B) Uso e definição dos arquivos nos processos de negócio.**
- C) Estrutura, organização e relacionamentos de arquivos.**
- D) Protótipos de arquivos e suas estruturas.**
- E) Comunicabilidade e organização de arquivos.**

**2. NO DIAGRAMA DE IMPLANTAÇÃO, TEMOS O CONCEITO DE NÓ E COMO ESTE É UTILIZADO. É CORRETO AFIRMAR QUE OS NÓS REPRESENTAM:**

- A) Qualquer dispositivo de hardware ou ambiente de execução de software.**
- B) Classes de componentes ou dispositivos de hardware.**
- C) Memória e processador.**
- D) Servidores onde acontece a execução dos componentes do software.**
- E) Os componentes de um software.**

---

## **GABARITO**

**1. Aprendemos que o diagrama de componentes documenta arquivos físicos armazenados em meio digital, representando diversos componentes de software do sistema. Assinale a alternativa que indica de forma mais completa o que é explicitado nesse diagrama.**

A alternativa "C" está correta.

O diagrama de componentes representa os arquivos do sistema que contêm programas e dados, separando-os de acordo com um padrão de projeto, e mostrando sua organização, estrutura e seus relacionamentos.

**2. No diagrama de implantação, temos o conceito de nó e como este é utilizado. É correto afirmar que os nós representam:**

A alternativa "A" está correta.

Os nós não representam um único tipo de artefato de infraestrutura do sistema. São tipicamente definidos de maneira aninhada e representam tanto dispositivos de hardware, quanto ambientes de execução de software básico, como sistemas operacionais, sistemas gerenciadores de banco de dados, navegadores de web etc.

## CONCLUSÃO

## CONSIDERAÇÕES FINAIS

Neste conteúdo, aprendemos a desenvolver diagramas usualmente empregados na fase de projeto de software. No primeiro módulo, vimos os diagramas de interação (sequência e comunicação). No diagrama de sequência, tivemos uma visão geral e sua relação com diagrama de classes e especificações de casos de uso. Também entendemos quais são os elementos que formam esse diagrama (básicos, decisões, repetições, criação e destruição de objetos). Falamos sobre as mensagens síncronas e assíncronas. Sobre o diagrama de comunicação, aprendemos sobre que elementos compõem esse diagrama e como ele pode ser utilizado em conjunto com o diagrama de sequência.

No módulo seguinte, tratamos do diagrama de classes de projeto. Esse é um dos diagramas mais importantes da fase de modelagem do software, desde a fase de análise, passando pelo projeto até a implementação. Tratamos das responsabilidades das classes, do papel dos relacionamentos, do detalhamento de métodos e dos atributos. Aprendemos sobre naveabilidade, classes de interface, de controle e pacotes.

Em seguida, falamos sobre diagrama de atividades e diagrama de estados. Em ambos, tratamos dos conceitos básicos e de suas respectivas finalidades. Cuidamos também de falar sobre seus elementos e o passo a passo para sua construção. Por fim, aprendemos também outros dois diagramas mais físicos, mas não menos importantes, que são o diagrama de componentes e o de implantação. Em ambos, vimos finalidades e conceitos.

Para ouvir um *podcast* sobre o assunto, acesse a versão online deste conteúdo.



## REFERÊNCIAS

BEZERRA, E. **Princípios de análise e projeto de sistemas com UML**. Rio de Janeiro, Brasil: Campus, 2015.

OMG. **Unified Modeling Language - Versão 2.5**. OBJECT MANAGEMENT GROUP, 2015.

## EXPLORE+

Para aprofundar mais seu conhecimento sobre o conteúdo apresentado:

## VEJA

O canal do YOUTUBE ***Estudo na Web*** , que tem como principal objetivo desmistificar e tornar mais fácil o aprendizado com exemplos simples e exercícios práticos, diversos materiais sobre UML, inclusive alguns específicos sobre os diagramas apresentados aqui.

## LEIA

O artigo ***Os principais diagramas da UML*** – Resumo rápido apresentado pela revista Profissionais TI, da organização de mesmo nome, que traz um conjunto de diversas ferramentas que podem ser utilizadas para modelagem dos diagramas apresentados neste conteúdo.

---

## CONTEUDISTA

Claudia Cappelli

## 🔗 CURRÍCULO LATTES