REVISAR O DIAGRAMA DE CLASSES DA ANÁLISE NO PROJETO DE SISTEMA

DIAGRAMA DE CLASSES DE PROJETO

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.

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).

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,

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: swap_horiz Arraste para os lados.

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] [{propriedades}]

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: Clique na barra para ver as informações.

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

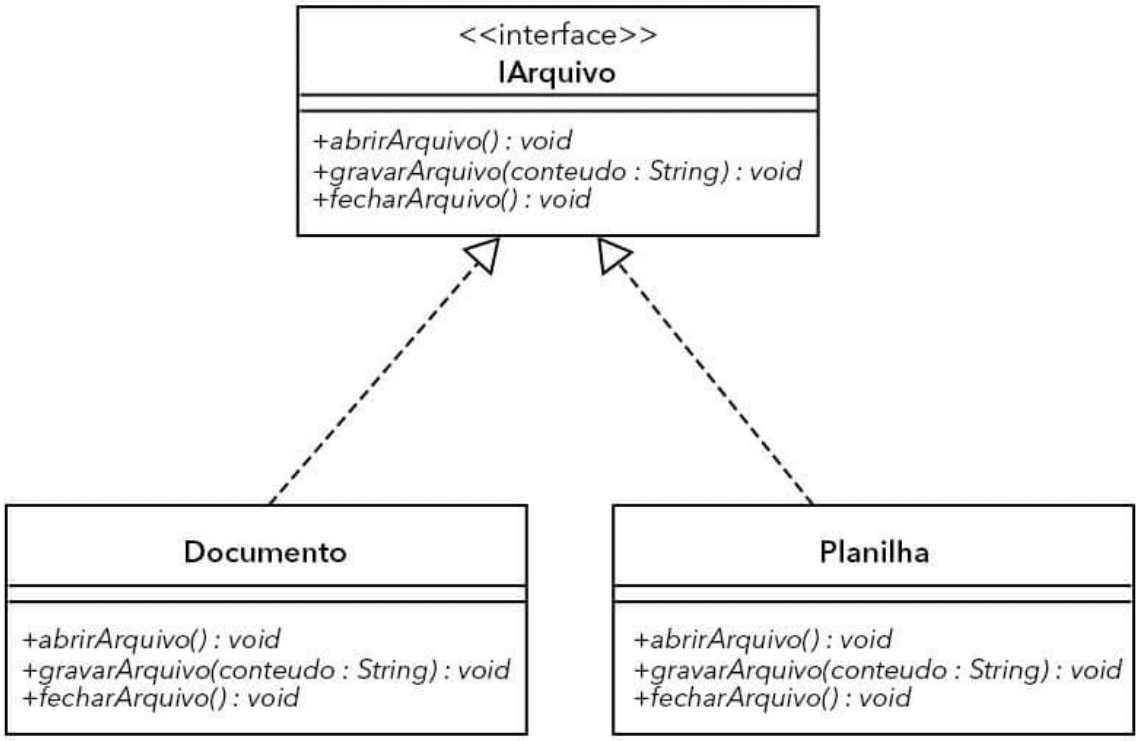
Representa relacionamentos todo-parte onde a existência da classe do lado parte depende da existência da classe do lado todo.

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.



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.

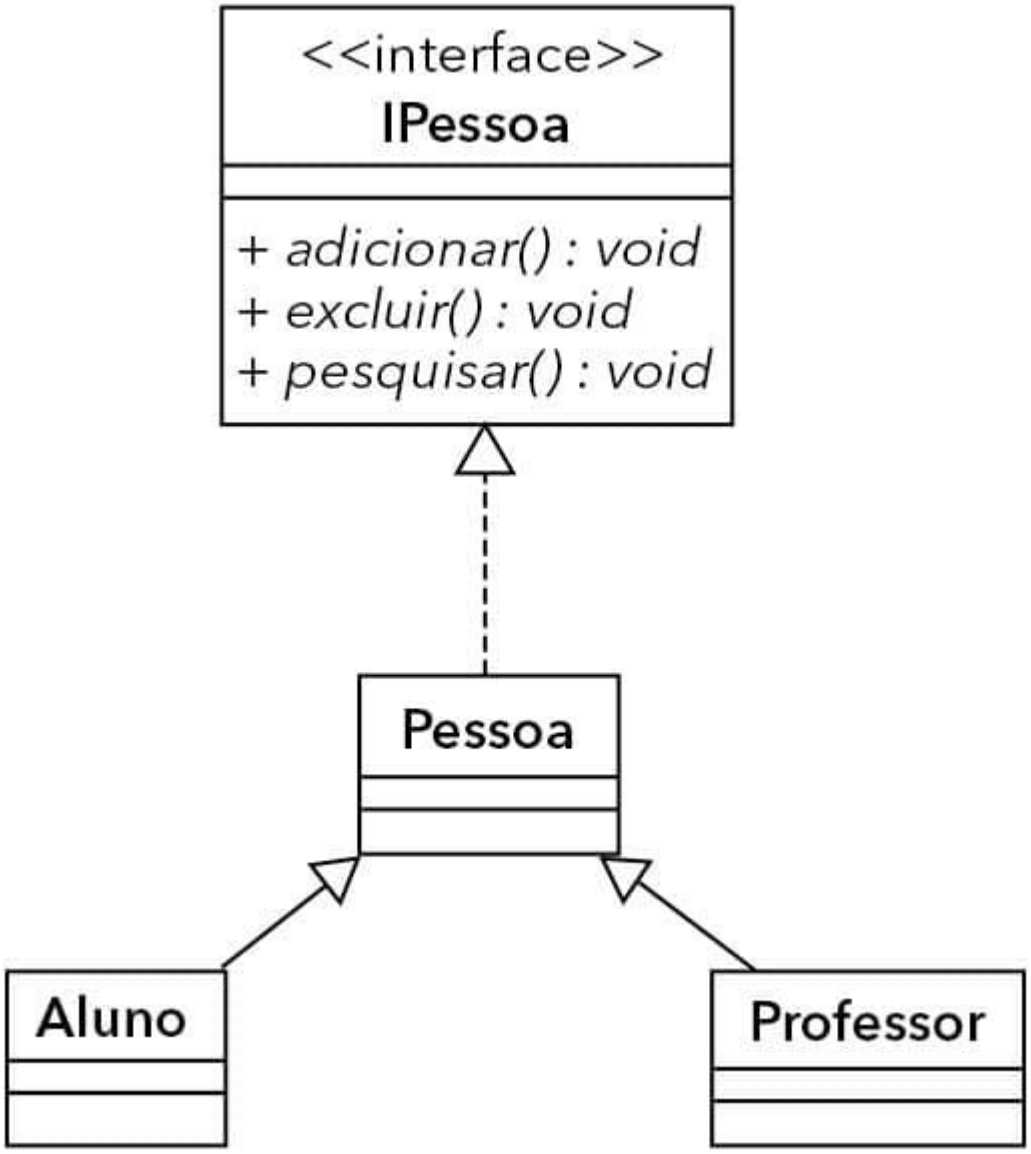
Não persistentes (transientes)

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.



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 navegabilidades 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.

Exemplo de navegabilidade. Elaborado na ferramenta Astah UML.