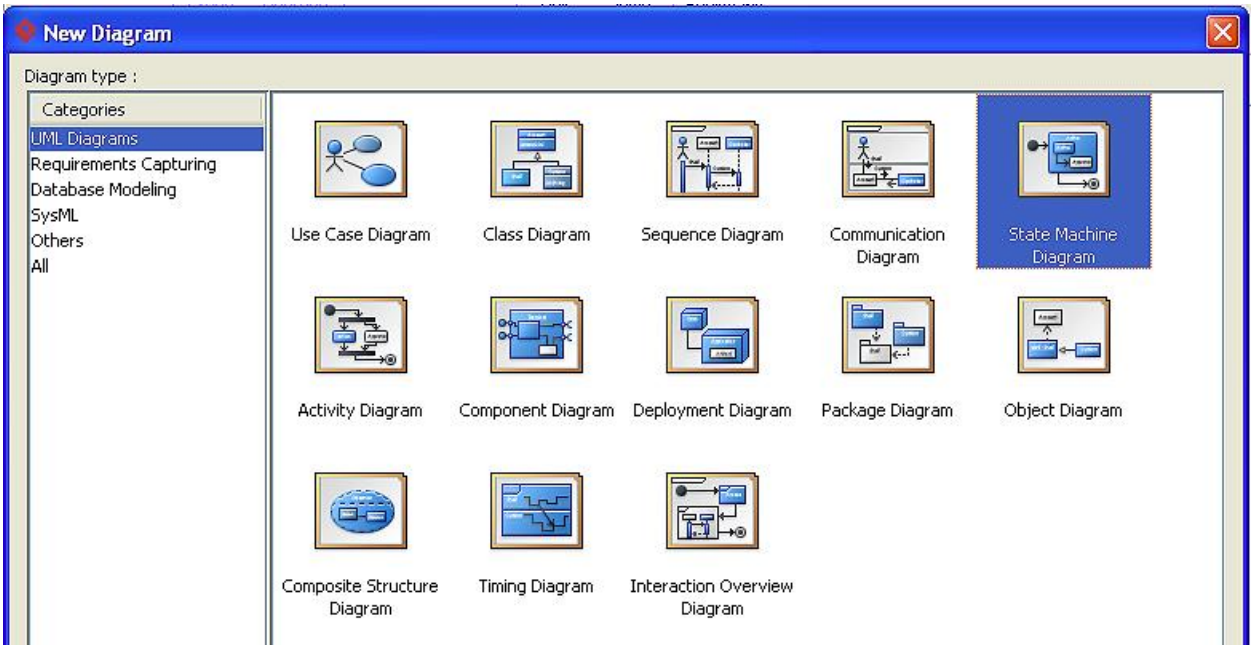
UNIDADE 3 – APROFUNDANDO NA UML
MÓDULO 1 – DIAGRAMA DE MÁQUINA DE ESTADOS

01

1 - MODELANDO UM DIAGRAMA DE MÁQUINA DE ESTADOS

Olá, seja bem-vindo a mais uma etapa no nosso estudo! Trataremos agora dos diagramas de máquina de estados.



Tela de diagramas UML do Visual Paradigm, em destaque, o diagrama de máquina de estados

Anteriormente estudamos a visão dinâmica representada pelos diagramas de sequência e de comunicação. Ambos os diagramas modelam as interações entre objetos. Neste módulo, estudaremos como o diagrama de máquina de estados modela o efeito que essas interações têm sobre a composição interna de cada objeto, ou seja, o que acontece com o objeto quando ele recebe uma mensagem.

As mensagens modeladas em diagramas de interação são o lugar onde os eventos externos lançam demandas sobre os objetos (eles invocam os objetos). As respostas internas dos objetos a esses eventos causam alterações aos estados dos objetos. Normalmente, os diagramas de estados são criados a partir das interações em um diagrama de sequência, tomando por base a relação entre as mensagens nos diagramas de interação e os eventos que desencadeiam alterações em um objeto.

02

A relação entre as interações e eventos que causam as mudanças de estado também pode nos ajudar a identificar os objetos que justifiquem um diagrama de máquina de estados. Na vida prática, é muito raro ter que modelar este nível de detalhamento. A grande maioria dos objetos é simplesmente criada,

referenciada, usada e mais tarde excluída. Mas, em muitas aplicações, existem alguns objetos fundamentais que são o foco da aplicação.

Por exemplo, num hipotético sistema de gerenciamento de um cinema, um objeto que contenha a informação sobre os preços dos ingressos provavelmente tem o seguinte ciclo de vida:

Todas essas mudanças são bastante simples, não requerendo um diagrama que explique ou detalhe o ciclo de vida deste objeto. De outro modo, os objetos que controlam os assentos em uma sala de cinema são **criados** quando uma sessão de cinema é definida, **disponibilizados** quando a sessão de cinema é posta à venda, reservados durante o processo de compra de ingressos e **vendidos** logo após a conclusão da venda de ingressos; um ingresso pode ainda ser cancelado/devolvido e o assento voltará ao estado de **disponível**.

A cada mudança de estado, as regras daquilo que pode acontecer a cada assento mudam. Se um assento é reservado por um cliente durante uma compra de ingresso, esse mesmo assento não pode ser vendido a outro cliente. Se um assento já foi vendido, ele não pode nem ser selecionado por outro cliente.

03

O objeto que rege o preço do ingresso não requer um diagrama de máquina de estados. Por outro lado, o **objeto que regulamenta o ciclo de vida de um assento** justifica a criação de um diagrama de máquina de estados.

Quando você revir os diagramas de sequência que tratam da venda de assentos você não conseguirá identificar facilmente naqueles diagramas o estado de cada objeto relacionado aos assentos do cinema. Ainda, é possível que cada estado de um determinado objeto possa estar documentado em diagramas de sequência separados, ou seja, um diagrama para cada cenário. É nesse momento que um diagrama de máquina de estados pode ajudar. Nos diagramas de sequência pode ser provável que você identifique mensagens chegando ao objeto, mas qual mudança comportamental cada mensagem efetua no objeto? A resposta a esta pergunta deve ser modelada em um diagrama de máquina de estados.

Exemplo hipotético: suponha que você tenha um objeto com 5 métodos, denominados: “a(), b(), c(), d(), e()”. Todos os métodos são públicos, mas há uma regra de negócio que define que o método “b()” só pode ser executado depois que o método “a()” for chamado, e que o método “c()” só pode ser chamado depois do método “b()”. Muito embora os três métodos estejam públicos e disponíveis, é necessário criar um controle (via programação) que regulamente esta regra. Esse controle deve impedir a execução dos métodos em outra ordem que não seja a pré-estabelecida. E para isso, o diagrama de máquina de estados será o ponto inicial para modelarmos essa questão.

Estado X → método a() → estado Y → método b() → estado W → método c() → estado Z.

Uma regra de negócio rege a sequência (ciclo de vida) de um objeto.

Ao analisar diagramas de sequência, procure por objetos que recebem muitas mensagens. Esses objetos são sérios candidatos a necessitarem de diagramas de estado para modelar as respectivas regras de negócio (ciclo de vida e sequenciamento).

04

Cada diagrama de máquina de estados descreve o clico de vida de um objeto em termos de eventos que provocam mudanças no estado do objeto. O diagrama de máquina de estados identifica os eventos externos e eventos internos que podem alterar o estado do objeto.

‘**Mas o que significa mudar o estado de um objeto?**

Na prática, a mudança de estado é controlada por um atributo do objeto que tem seu valor alterado ao longo da sua utilização pelo programa. O estado de um objeto é simplesmente sua condição atual. Esta condição reflete nos valores dos atributos que descrevem o objeto. Comportamentos no sistema alteram esses valores, redefinindo ou mudando o estado do objeto.

Ao ler documentos de caso de uso, é muito comum identificarmos os estados dos objetos por meio dos adjetivos e verbos que são atribuídos aos nomes dos objetos. Exemplos: cadastrado, configurado, definido, reservado, recebido, enviado, emitido, orçamento feito, venda concluída, dados cadastrados, etc. Procure por palavras como essas para lhe ajudar a identificar os estados dos objetos.

Os diagramas de estado modelam os eventos que disparam uma transição (mudança) de um estado para outro estado do objeto.

Um evento corresponde a uma mensagem enviada ao objeto pedindo (ou mandando) que ele faça algo. Isso é chamado de “**ação**”. Ações alteram valores de atributos do objeto, redefinindo o estado do objeto.

Note que nem todas as mensagens implicam em uma ação que altere o estado do objeto. A mensagem como mostrarNome (), por exemplo, não faz nenhuma alteração no objeto para que ele interprete isso como uma transição. Os nomes dos métodos, infelizmente, às vezes nos enganam, por isso, analise sempre outros documentos (como o caso de uso) para determinar estados de objetos.

Enquanto um objeto está em um estado, também pode executar o trabalho associado a esse estado. Esse trabalho é chamado de “**atividade**” e não altera o estado do objeto. Por exemplo, enquanto uma sessão de cinema estiver aberta, ela pode manter o controle dos lugares vendidos.

05

1.1. Atributos do Estado

Para implementar o estado de um objeto, normalmente o meio mais fácil é criar um **atributo** que define seu estado. Esse atributo é baseado em uma enumeração onde cada item representa um estado.

Exemplo:

Suponha que você tenha a seguinte enumeração de estados de um objeto qualquer:

1 = criado;
2 = configurado;
3 = registrado;
4 = finalizado.

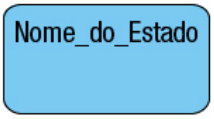
E que no seu objeto tenha um atributo de nome “Estado”. Quando você atribui o valor “1” ao atributo “Estado”, você está dizendo que o estado atual do objeto é “criado”, quando você muda esse atributo de 1 para 2, você está dizendo que o estado atual do objeto é “configurado”.

Ao ler o valor do atributo, é necessário conhecer a lista de enumeração de estados, pois só assim o programador saberá o que significa o valor numérico de 1 a 4 para o objeto. Posteriormente, é muito provável que exista no código uma cláusula do tipo “select case”, onde, para cada valor (1, 2, 3 e 4) o sistema irá executar um comportamento diferente.

06

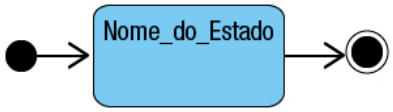
2 - Modelando estados e eventos de objeto

A base para o diagrama de máquina de estados é a relação entre estados e eventos. Um estado simples (ou estado regular) é modelado como um retângulo com cantos arredondados e com o nome do estado dentro.



Padrão de diagramação de um estado de um objeto

Além de estados simples, a UML define dois estados especiais: o **estado inicial** e o **estado final**. Cada um tem sua própria notação única. O estado inicial de um objeto é modelado como um ponto sólido com uma seta apontando para o primeiro estado. O estado final é representado por um círculo ao redor do ponto sólido. Setas indicam a sequência de estados, ou seja, o ciclo de vida do objeto. Importante saber que ao chegar ao estado final, não há como o objeto mudar de estado, pois ali encerra o seu ciclo de vida.



Notação representando os estados inicial e final

07

Uma determinada regra de negócio pode definir qual será o próximo estado do objeto. Também é possível que a regra defina uma entre várias opções de mudança de estados.

O diagrama abaixo representa uma situação hipotética válida, onde podemos observar mudanças que vêm e voltam e mais de uma opção de sequência. Observe que é possível:

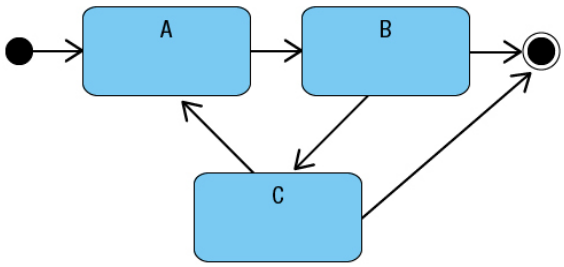
A mudar para B,

B mudar para fim,

B mudar para C,

C mudar para fim,

C mudar (voltar) para A.



Exemplo de mudanças de estado com mais de uma opção

08

Já citamos algumas regras que definem os estados de uma poltrona de um cinema:

- Quando uma sessão de cinema é criada, assentos também são **criados** para aquela sessão.
- Quando os ingressos de uma sessão são colocados à venda, os assentos tornam-se **disponíveis**.
- Quando um cliente seleciona um assento para a compra de um ingresso, enquanto a venda não é concluída o assento fica **reservado**.
- Se o cliente não concluir a compra do ingresso, o assento volta a ficar **disponível**.
- Quando o cliente conclui a compra do ingresso, o assento fica **vendido**.

- Um cliente pode cancelar a compra de ingressos, após concluir a devolução do pagamento, o assento volta a ficar **disponível**.

Observe que sublinhamos os verbos e adjetivos que definem os possíveis estados das poltronas. De posse dessas informações, já podemos desenhar o diagrama de máquina de estados dos assentos do cinema:

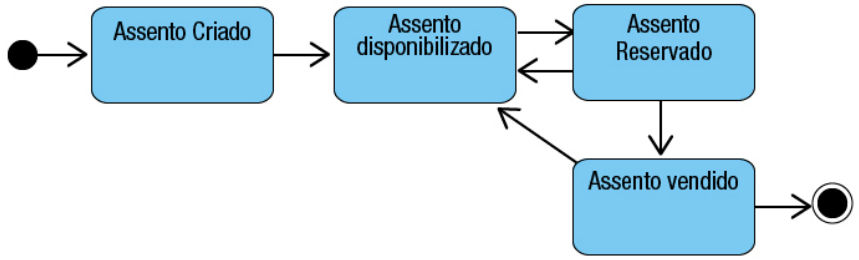


Diagrama de máquina de estados que representa o cenário de estados de uma poltrona de uma sessão de cinema

O estado final do objeto não significa que o objeto é destruído, mas sim que ele é congelado em uma situação que não pode ter seu estado final mudado. Seus atributos permanecem visíveis: podem ser lidos, mas não alterados.

Muito embora seja comum destruir objetos no seu ponto de término, também é comum vermos sistemas onde os objetos são armazenados em bancos de dados quando chegam no estado final. Ainda, é comum que dados sejam mantidos em sistemas para auditoria ou para possibilitar desfazer operações de exclusão. Nessa ideia, uma situação de “excluído” é apenas um estado a mais do objeto (modelado como mais um retângulo de cantos arredondados) e não o ato de realmente apagar a informação.

09

3 - Modelando eventos

Um evento sensibiliza um objeto fazendo com que um método seja executado. É muito comum existirem objetos instanciados em um sistema que ficam aguardando a sua respectiva execução. A ordem que diz “execute tal tarefa” pode ser um evento.

Tecnicamente falando, um evento é um gatilho que evoca uma resposta na forma de uma ou mais condutas. Um **evento** em um diagrama de máquina de estados corresponde a uma **mensagem** em um diagrama de sequência.

Quando, por exemplo, o objeto “assento” recebe uma mensagem “reserve o assento número 36F”, o ato de receber essa mensagem é um evento que desencadeia uma mudança no estado do objeto que referencia o assento “36F”, mudando-o de “disponível” para “reservado”.

Veja alguns exemplos do que um evento pode ser:

- A recepção de uma comunicação executada por outro objeto, como um objeto de compra de ingressos que dispara um evento de reserva/compra da poltrona da sessão de cinema.
- Uma regra de negócio baseada em **tempo decorrido**.
- Uma regra de negócio baseado em um teste lógico no sistema, como por exemplo “estoque abaixo do mínimo”, ou “saldo negativo”, ou “usuário efetuou logoff”.
- Uma regra de negócio baseado em um evento do computador, como por exemplo a chegada de um e-mail, a inserção de um CD no drive de CD, um sistema que é carregado, uma informação que chega por uma porta lógica do computador etc.

Tempo decorrido
Regra baseada em tempo corrido pode ser:

- Um evento que ocorra todo dia à zero hora;
- Ou um evento que ocorra todo primeiro dia útil do mês;
- Ou um evento que ocorra a cada 30 minutos;
- Ou um evento que ocorra passados 5 minutos sem a intervenção do usuário no sistema (geralmente ocorre o logoff do sistema);

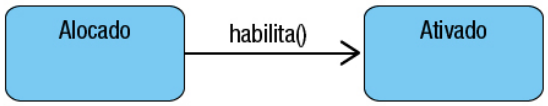
Ou passados 5 minutos de reserva de uma poltrona e o usuário não concluiu a venda (nesse caso ele perderia a reserva e a poltrona voltaria ao estado de disponível).

10

3.1 - Modelagem de eventos e transições

Uma notação de evento de estado é feita por meio de um texto em uma seta que liga um estado a outro, usando uma linha sólida com uma ponta de seta. A seta representa a transição associada com o acontecimento. Os termos “eventos” e “transições” são muitas vezes utilizados como sinônimos no diagrama de máquina de estados, visto que ambos sempre aparecem juntos.

A direção da seta indica a direção da mudança. A base da seta identifica o estado em que o objeto estava quando o evento é gerado, e a ponta da seta identifica o estado em que as alterações afetam o objeto (a causa do evento). A figura abaixo apresenta o evento “habilita()” que causa a transição do estado “alocado” para o estado “ativado”.



Exemplo de diagrama de estado contendo o nome da transição.

A maioria dos eventos em um diagrama estados vem de outros objetos. Mas o diagrama estados não lhe diz de qual objeto os eventos vieram. Isso acontece na medida em que o diagrama de máquina de estados detalha apenas o efeito que cada evento gera. Os diagramas de interação modelam as origens e destinos das mensagens. Dentro de um diagrama de máquina de estados, um objeto não precisa saber quem enviou o evento. Um objeto só é responsável pela forma como ele responde ao evento. Por isso, é importante lembrar que a base da seta identifica o estado em que o objeto está em quando recebe o acontecimento e não diz nada sobre o local de onde o evento veio.

Veja um **exemplo** bem simples.

Exemplo
Suponha que você tenha uma TV e que ela possua um controle remoto. Suponha também que você tenha comprado um controle remoto daqueles universais (que operam qualquer aparelho). Para ligar a TV você pode optar por pressionar o botão de ligar na TV, no controle remoto da TV ou no controle remoto universal. Não importam quem está enviando a mensagem de ligar a TV, o que importa é que ela estava em um estado de "desligado", alguém enviou uma mensagem de "ligar()", e que agora ela executará os procedimentos internos para ligar (mostrando a imagem e o som do canal atual) até ficar no estado de "ligado". Para a TV, pouco importa quem manda a mensagem, o que importa é como ela reage à mensagem de "ligar()".

11

3.2 - Mensagens e Estados

O estado no qual o objeto se encontra pode afetar como ele responde à **mensagem** recebida, ou seja, as alterações na condição de um objeto podem ditar mudanças no seu comportamento. Por exemplo, quando a atual condição (estado) de uma conta corrente está negativa, a conta corrente responde de uma forma diferente do que quando a conta corrente tem um saldo positivo de crédito, ou seja, os cheques são rejeitados (em vez de pagar), saques são bloqueados, transferências bancárias são impossibilitadas (ou o banco pode permitir a operação e cobrar uma taxa de juros).

Veja outro exemplo: se você pressionar o acelerador em seu carro enquanto o carro está ligado, o motor acelera. Se você pressionar o acelerador quando o carro está desligado, o motor não faz nada. De forma oposta, uma mensagem idêntica pode executar operações distintas em um objeto, dependendo do estado em que ele se encontre. Imagine uma TV com um botão de "power", pressionar o "power" quando a TV está ligada, desliga a TV; pressionar o "power" quanto a TV está desligada, liga a TV.

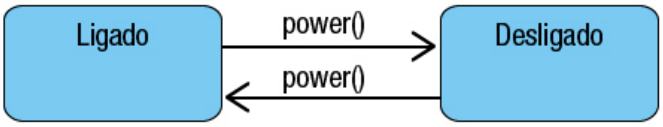


Diagrama de estado de um interruptor de TV

Levando em consideração que um mesmo evento faz com que ocorram dois comportamentos diferentes em um objeto, então a resposta que o objeto executa é a relação entre a mensagem e o estado do objeto.

Resposta = Estado + Evento

12

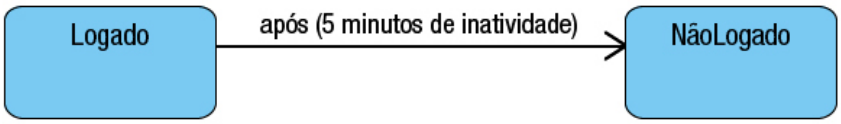
3.3 - Modelando eventos com tempo decorrido

Muitas vezes uma mudança de estado é acionada devido à **passagem do tempo**. Por exemplo, em sistemas web é muito comum que se o usuário não executar uma operação dentro de um determinado tempo, ele é desconectado do sistema e um novo login é necessário.

Em sistemas de compra, após você selecionar um produto para comprar, ele fica reservado para você concluir a compra; entretanto, passado determinado tempo, se você não concluir a comprar, sua reserva é cancelada e o produto fica disponível para outro cliente.

Para modelar o tempo como uma condição de mudança de um estado de um objeto é comum utilizarmos expressões que nos remete à passagem do tempo, como as expressões “após”, “depois de”, “passados” etc. Após a expressão de condição de tempo, entre parêntesis, adicionamos a unidade de tempo necessária para acontecer a condição.

Exemplo: para modelar uma mudança de estado que ocorre após 5 minutos de inatividade do sistema, poderíamos utilizar a expressão “após (5 minutos de inatividade)”. Dessa forma, teríamos o seguinte diagrama hipotético:



Exemplo de evento com tempo decorrido

13

3.4 - Modelando eventos de mudança em regra de negócio

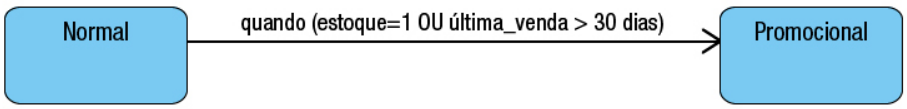
Outro tipo de evento muito comum em sistema é aquele que afeta uma determinada **regra de negócio**. Por exemplo, em um sistema hipotético de controle de estoque, quando o estoque de um determinado produto ficasse em apenas uma unidade, o estado dele poderia ser considerado “estoque crítico”. Em

outro contexto, quando numa loja de sapatos um determinado sapato só tivesse um único par daquele modelo, ele poderia mudar para o estado “item promocional”.

Esse tipo de regra de negócio sofre uma avaliação lógica booleana. Ou seja, a resposta à condição deve ser verdadeira ou falsa. Quando a condição for verdadeira, o evento ocorre. Por isso, muita atenção deve ser dada na modelagem das regras de negócio, especialmente quando forem regras compostas (que utilizam operadores “e” e “ou”).

As palavras-chave que utilizamos para modelar esse tipo de evento geralmente são “quando”, “se”, “enquanto que”.

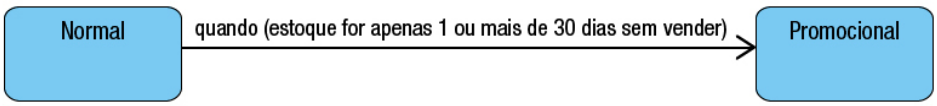
Por exemplo, para modelar a regra de negócio “uma peça deve entrar em estado promocional se o estoque for de apenas 1 ou se ela estiver a mais de 30 dias sem ter sido vendida”, teríamos o diagrama abaixo:



Exemplo de evento ligado a regra de negócio

14

Também é possível utilizar um texto descritivo ao invés de uma expressão lógica para representar o evento, como no exemplo a seguir:



Exemplo de evento ligado a regra de negócio

Ambos os exemplos anteriores representam a mesma regra de negócio, o produto entrará em promoção **se estiver apenas um item no estoque** ou **se passar mais de 30 dias** sem nenhuma venda deste tipo de produto.

Durante a modelagem é possível identificar outros estados necessários para o correto funcionamento do sistema. Neste caso, deixe clara a regra que cria os estados intermediários. A modelagem revela problemas de design de uma forma que a documentação textual dificilmente mostra.

15

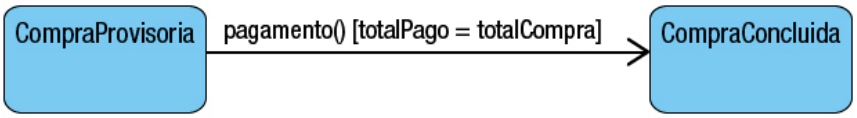
3.5 - Modelando uma condição de guarda

Normalmente, um evento é recebido e respondido de forma incondicional. Às vezes, porém, o recebimento do evento é **condicional**. Por exemplo, quando uma ordem de compra de bilhete é inserida no sistema de cinema, ela é considerada provisória até que o pagamento tenha sido recebido.

O problema é que não é qualquer pagamento que resolve a questão, é necessário testar para ter certeza de que o pagamento realmente cobre o custo bilhete. Este teste é chamado de uma **condição de guarda**.

Muitos sistemas incluem esse tipo de teste para permitir que o cliente pague com mais de um tipo de forma de pagamento, por exemplo, que ele pague metade em dinheiro e metade no cartão de crédito.

A condição de guarda é modelada colocando a expressão lógica dentro de colchetes logo após o nome do evento. Veja, por exemplo, o diagrama abaixo onde a compra só é considerada concluída quando o total pago for igual ao valor total da compra:



Exemplo de diagrama com condição de guarda

16

4 - Onde encontrar os estados dos objetos

Durante a modelagem, o melhor lugar para se encontrar os estados dos objetos é analisando os diagramas de sequência, cada mensagem que lá aparece é um indicativo de que ali podemos estar criando um novo estado para o objeto.

Lembre-se todo objeto tem um estado inicial quando é instanciado e outro quando recebe a primeira mensagem. Também tem um estado sempre que não é mais necessário e outro quando é destruído.

17

Resumo

Neste módulo, aprendemos que:

a) O diagrama de estado modela a via de um objeto único. Os elementos fundamentais do diagrama de máquina de estados são os estados e os eventos.
b) Um estado representa uma condição de um objeto. Um estado é registrado pelos valores de um ou mais atributos de um objeto.
c) Um evento dispara uma mudança de um estado, ou seja, uma transição.

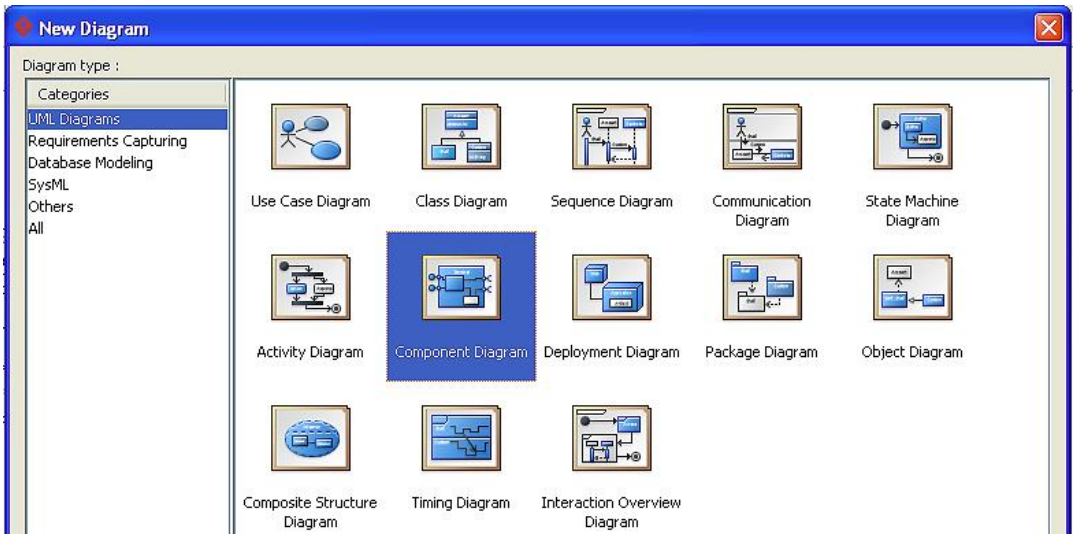
d) Uma expressão de uma ação é um comportamento disparado por um evento. Esse comportamento faz mudanças nos valores dos atributos e redefinem o estado do objeto.
e) Quando todas as ações associadas a uma transição possuem o mesmo comportamento, você pode remodelar em uma única ação.
f) Um mesmo evento pode mudar o estado de um objeto baseando-se no estado atual (exemplo da TV que liga e desliga sempre que pressionamos o botão “power”).
g) Um mesmo evento pode não mudar o estado do objeto baseando-se no estado atual (exemplo da TV desligada e pressionar o botão “desligar”).
h) Eventos podem ocorrer baseando-se na chegada de uma mensagem, em tempo decorrido, teste lógico, ou uma regra de negócio.
i) Estados intermediários podem ser criados durante a modelagem para facilitar o entendimento de regras de negócio complexas.
j) Condições de guarda são regras de negócio que disparam eventos quando seu resultado lógico é verdadeiro.

UNIDADE 3 – APROFUNDANDO NA UML
MÓDULO 2 – DIAGRAMA DE COMPONENTES

01

1 - O DIAGRAMA DE COMPONENTES

Olá, seja bem-vindo a mais um módulo de estudos! Neste momento trataremos dos diagramas de componentes.



Tela de diagramas UML do Visual Paradigm, em destaque, o diagrama de componentes.

02

Uma vez que o projeto lógico é concluído, utilizando os casos de uso, diagramas de classe e interação, o próximo passo é definir a **implementação física do seu *design***. A implementação física deve considerar três aspectos:

a) A descrição física do software.	b) O layout do *hardware*.	c) A integração do *software* e do *hardware*.
• Os modelos de diagrama de componentes da implementação física do software, especificado pelos requisitos lógicos no diagrama de classe.	• O diagrama de Implantação e modelos da arquitetura física do *hardware*.	• Os diagramas combinados de implantação e de componentes e o modelo de integração e distribuição do *software* em todo o *hardware*.

As **classes** descrevem a organização lógica e a intenção de seu projeto de *software*. Os **componentes** descrevem as implementações físicas de seu projeto de *software*.

A finalidade do diagrama de componentes é definir módulos de *software* e as relações entre eles. Isso fornece um meio de definir o *software* como um conjunto de unidades modulares e intercambiáveis, que podem ser reunidas para criar sucessivamente novas unidades modulares melhores e mais complexas. Os componentes podem representar qualquer coisa desde uma única classe até sistemas completos.

03

Um diagrama de componentes pode ser usado para ilustrar a relação entre as classes que **especificam** os requisitos para o componente, e os artefatos que **implementam** o componente. Os artefatos representam qualquer tipo de código que pode residir na memória do computador, arquivos binários, scripts, arquivos executáveis, bancos de dados ou aplicações.

A relação entre um componente e um artefato pode parecer confusa. O componente é a definição de um **tipo** de execução física. Um artefato é a implementação real. A mesma notação de componente pode ser usada para representar os artefatos também.

A Arquitetura Orientada a Modelo (Model Driven Architecture - MDA) tenta descrever um sistema independente de sua implementação. Isto permite que o mesmo conjunto de exigências (que é o mesmo tipo de componente) possa ser implementado em diferentes tecnologias, isto é, em vários artefatos físicos diferentes.

Cada componente tem que definir uma ou mais interfaces que permita com que outros componentes possam comunicar com ele. Os requisitos para a implementação interna do componente são especificados por classes. A implementação real é encapsulada no artefato que o implementa, por exemplo, classes Java, .NET, PHP, HTML e XML.

04

Os **artefatos** são agrupados em três categorias gerais:

Componentes de implantação	Componentes de trabalho	Componentes de execução
São necessários para executar o sistema. Exemplos incluem sistemas operacionais, a Máquina Virtual Java (JVM), e os sistemas gerenciadores de bando de dados (SGBD).	Incluem os modelos, código fonte e arquivos de dados usados para criar componentes de implantação. Exemplos: diagramas UML, classes em Java, arquivos java .JAR, bibliotecas de vínculo dinâmico (DLL), e tabelas de banco de dados.	São componentes criados durante a execução do aplicativo. Exemplos incluem Enterprise Java Beans, Servlets, HTML e documentos XML, componentes COM + e .NET, e componentes CORBA.

As **dependências** representam os **tipos de relações** que existem entre os componentes em um diagrama de componentes. Por exemplo:

a) Um componente depende das classes que residem dentro dele para especificar o comportamento do componente.

b) Um componente depende dos artefatos utilizados para implementá-lo.

c) Componentes dependem um do outro para a funcionalidade poder ser executada e para obter informações.

Um componente inclui componentes de diagramas, componentes de interface e relacionamentos para especificar os requisitos para a execução física do seu sistema em termos de módulos de *software* e as relações entre eles.

05

1.1 - Processos de Modelagem de Negócios

Os diagramas de componentes e implantação também podem ser usados para modelar sistemas de negócios, incluindo os processos manuais.

Quando usado para este fim, os componentes representam os **processos** de negócios (como um processo de pagamento ou de uma compra on-line) e os **produtos** desses processos (como pagamentos e notas fiscais).

Os nós do diagrama de implantação representam unidades de negócios (departamentos, equipes e subsidiárias, por exemplo) ou de recursos empresariais (como um armazém, uma unidade de fabricação, ou de um escritório em que os departamentos de negócio residem).

06

2 - Modelando o diagrama de componentes

Um componente é um tipo de contêiner. Como tal, não têm características próprias, mas contém as classes que definem suas características. Um componente encapsulado proporciona uma vista da funcionalidade definida pelas classes contidas. O propósito de tal representação é apoiar uma abordagem baseada em componentes para a construção de *software*.

Cada unidade de *software* deve ser capaz de funcionar isolada, ser reutilizável e substituível.

A chave para a capacidade de substituir os componentes é seguir as **regras de encapsulamento**, ou seja, limitar a definição do componente para uma explicação da sua **finalidade** e respectiva **interface**. Esta técnica é comparável à **encapsulação de métodos**.

Um componente não necessariamente expõe todas as interfaces das classes contidas (muitas classes podem ser protegidas ou privadas). Algumas das interfaces são usadas apenas para a comunicação entre as classes contidas (interfaces protegidas). O modo pelo qual o componente expõe as interfaces pode variar. O componente pode agregar interfaces a nível de classe. Por exemplo, as classes contidas podem oferecer cinco métodos diferentes para se comprar um ingresso no cinema, mas o componente pode oferecer uma única interface chamada, por exemplo, de "ComprarIngresso".

É importante notar que o diagrama de componentes é apenas uma especificação. Não há uma instância deste diagrama. A este respeito, é muito parecido com um diagrama de classes. Para modelar as instâncias de componentes que representam a execução do aplicativo, use os diagramas de componentes e de implantação juntos.

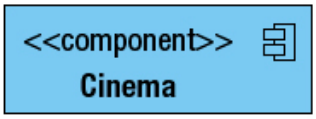
encapsulação de métodos
Enquanto um método fornece os meios para receber a entrada e gerar a saída, os mecanismos que ele usa para fazer isso estão escondidos por trás da interface objeto, ou seja, a assinatura de operação. As interfaces fornecidas pelo componente expõem uma ou mais das interfaces das classes contidas. A diferença é que outros componentes não têm acesso direto às interfaces contidas; entretanto, os desenvolvedores têm a liberdade de alterar o design interno, incluindo a substituição de classes internas, sem perturbar os outros componentes que dependem do componente alterado.

07

2.1 - Componentes de modelagem

A UML define um retângulo com ícone específico para representar um componente; o estereótipo “<<component>>” também aparece no símbolo.

O nome do componente é colocado dentro do ícone, como o exemplo a seguir:



Exemplo de diagramação de um componente

08

2.2 - Estereótipos de componentes

Os estereótipos dos componentes fornecem pistas visuais do papel que o componente desempenha na arquitetura. Os estereótipos de componentes referem-se a tipos de artefatos (código fonte, arquivos binários, bancos de dados, e assim por diante) usados para implementar os recursos do componente, e aos tipos de componentes EJB, CORBA, COM +, e .NET, por exemplo. Alguns dos estereótipos mais comuns de componentes gerais estão listados abaixo:

a) «Executable»: um componente que roda em um processador.

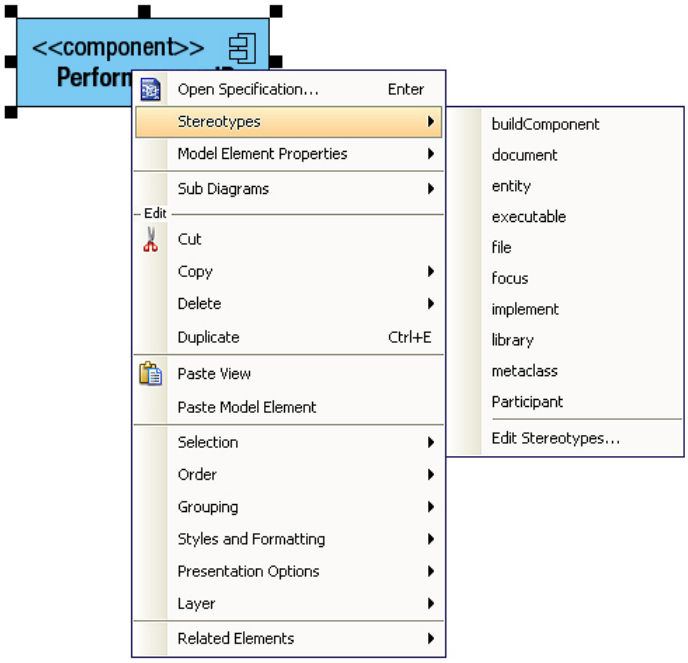
b) «Library»: um conjunto de recursos referenciados por um executável durante a execução.

c) «Table»: um componente de banco de dados acessado por um executável.

d) «File»: representa normalmente dados ou código-fonte.

e) «Document»: um documento como uma página Web.

A imagem abaixo apresenta os estereótipos padrões que podemos encontrar na ferramenta Visual Paradigm:



Estereótipos padrão da ferramenta Visual Paradigm

Executable
Um componente que roda em um processador.

Library
Um conjunto de recursos referenciados por um executável durante a execução.

Table
Um componente de banco de dados acessado por um executável.

File
Representa normalmente dados ou código-fonte.

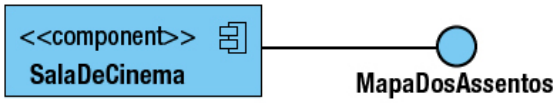
Document
Um documento como uma página Web.

09

2.3 - Modelando as interfaces dos componentes

Uma **interface** de um componente pode ser modelada pela representação de um círculo ligado ao componente por uma linha sólida. Veja o exemplo abaixo, em que o componente SalaDeCinema

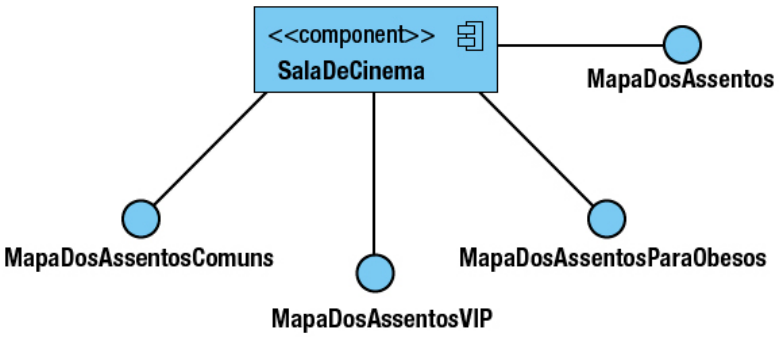
incorpora a interface para exibir um mapa com os assentos disponíveis e ocupados para uma determinada sessão de cinema:



Exemplo de interface que mostra um mapa dos assentos disponíveis e ocupados de uma sala de cinema.

A interface implementada por um componente é definida pelas classes dentro do componente. Assim, a interface já deveria ter sido definida em seus diagramas de classes. Além disso, um componente pode implementar muitas interfaces de uma vez, de acordo com os tipos e quantitativos descritos pelas classes implementadas pelo componente.

Veja o exemplo a seguir, onde o componente SalaDeCinema prevê vários tipos de interfaces diferentes, complementares (especializadas) a MapaDosAssentos:



Exemplo de componente com várias interfaces.

10

2.4 - Modelando uma dependência entre uma interface e um componente

Por vezes, um outro componente precisa acessar uma determinada interface para realizar uma regra de negócio. Para tal necessidade, podemos modelar um relacionamento de dependência entre o componente e a interface.

Exemplo: o componente “SalaDeCinema” provê a interface “MapaDosAssentos”, já a aplicação que vende os ingressos e que incorpora a interface com usuário possui o componente “AlocaçãoDePoltronas”. Para que essa aplicação possa acessar a interface “MapaDosAssentos”, é necessário criar uma dependência entre o componente “AlocaçãoDePoltronas” e a interface “MapaDosAssento”.

Essa relação de dependência é modelada por meio de uma seta pontilhada que sai do componente e aponta para a interface. O diagrama a seguir ilustra que o componente “AlocaçãoDePoltronas” não

funcionará ao menos que ele possa acessar o componente “SalaDeCinema” por meio da interface “MapaDosAssentos”.

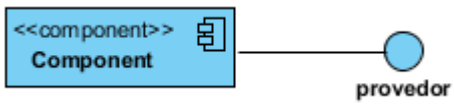
Veja o diagrama abaixo:

Exemplo de relação de dependência entre um componente e uma interface.

11

2.5 - Modelando tipos de interface e seus relacionamentos

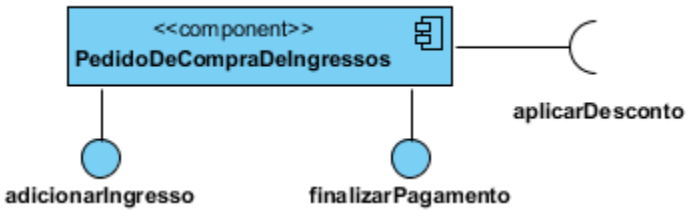
Uma funcionalidade essencial de um componente é sua habilidade de definir interfaces. O componente precisa expor algum(ns) meio(s) para que outros componentes se comuniquem com ele. As interfaces podem ser de dois tipos: provedor e consumidor.

Provedor	Consumidor
Quando um componente implementa uma interface que oferece serviços ele é denominado de provedor.	Quando um componente implementa uma interface que acessa um serviço de outro componente ele é um consumidor.
Exemplo: quando um componente “PedidoDeCompraDeIngresso” oferece funcionalidades de “adicionarIngresso” e de “finalizarPagamento”, percebemos que a implementação destas funcionalidades pertence a este componente, ou seja ele provê esse serviço, daí ele é um provedor. O ícone que representa um provedor é o círculo fechado.  **Diagramação de uma interface provedora**	**Exemplo**: quando o componente “PedidoDeCompraDeIngresso” precisa de aplicar descontos estudantis e de idosos na compra de um bilhete de cinema, ele precisa consultar o componente “Desconto” para receber as regras relacionadas ao desconto. Desta forma, o componente “PedidoDeCompraDeIngresso” precisa de uma interface que consome as informações de outro componente que atuará como provedor. O ícone que representa um provedor é uma meia-lua (semicírculo) que se acopla ao ícone de um provedor.  **Diagramação de uma interface consumidora**

Uma interface provedora fornece serviço para uma interface consumidora.

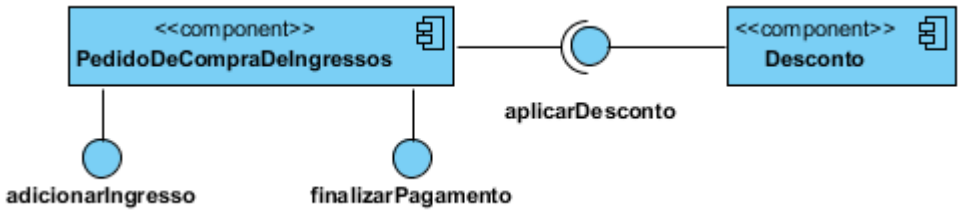
12

No exemplo abaixo vemos o componente “PedidoDeCompraDeIngresso” apresentando duas interfaces provedoras e uma interface consumidora:



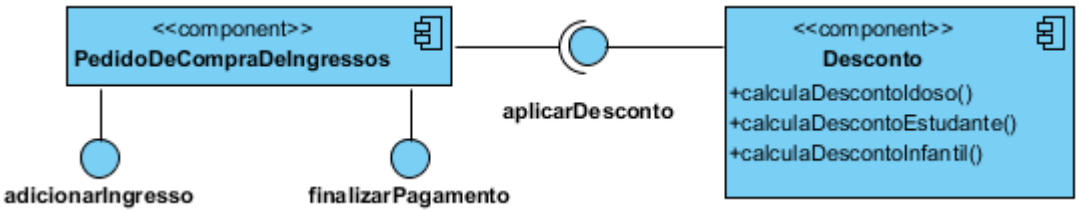
Exemplo de componente com interfaces consumidoras e provedoras.

Este próximo exemplo demostra como duas interfaces, uma consumidora e outra provedora são acopladas para comunicação entre elas:



Exemplo de duas interfaces, uma consumidora e outra provedora, conectadas entre si.

Por fim, temos ainda a possibilidade de expor os métodos de acesso às interfaces do componente. O exemplo abaixo apresenta três métodos de acesso à interface de descontos:



Mesmo exemplo anterior, agora incluindo os métodos de acesso ao componente

Finalmente, os componentes podem ser organizados em pacotes assim como outros diagramas e elementos do modelo. Isto pode ser muito útil na gestão de distribuição do aplicativo. O resultado é um diretório que contém todos os elementos de *software* necessários para a implementação do sistema ou subsistema representada pelo pacote.

RESUMO

Neste módulo, aprendemos que:

a) O propósito do diagrama de componentes é definir módulos de *software* e o relacionamento entre eles. Cada componente é um pedaço do código fonte que reside na memória do computador.

b) Os componentes podem ser de três tipos:

   a. Componentes de implantação, que são necessários para um sistema funcionar.

   b. Componentes de trabalho, que inclui arquivos de modelagem, código fonte, arquivos de dados e são utilizados para criar os componentes de implementação.

   c. Componentes de execução, que são os componentes criados enquanto a aplicação está funcionando (em tempo de execução).

c) A UML define que cada componente precisa possuir classes que permitam a sua execução. Essas classes especiais são chamadas de Interfaces e provêm mecanismo de troca de dados com outros componentes.

d) Dependências são um tipo de relacionamento que define a necessidade de existir um determinado componente para que outro funcione.

e) Um componente pode ser estereotipado por uma grande variedade de tipos, os mais comuns são: «executable», «library», «table», «file», e «document».

f) Um componente é modelado com um retângulo que incorpora o ícone de componente no seu desenho.

g) Uma interface pode ser de dois tipos, provedora e consumidora.

   a. As interfaces provedoras são representadas por um círculo e fornecem funcionalidades;

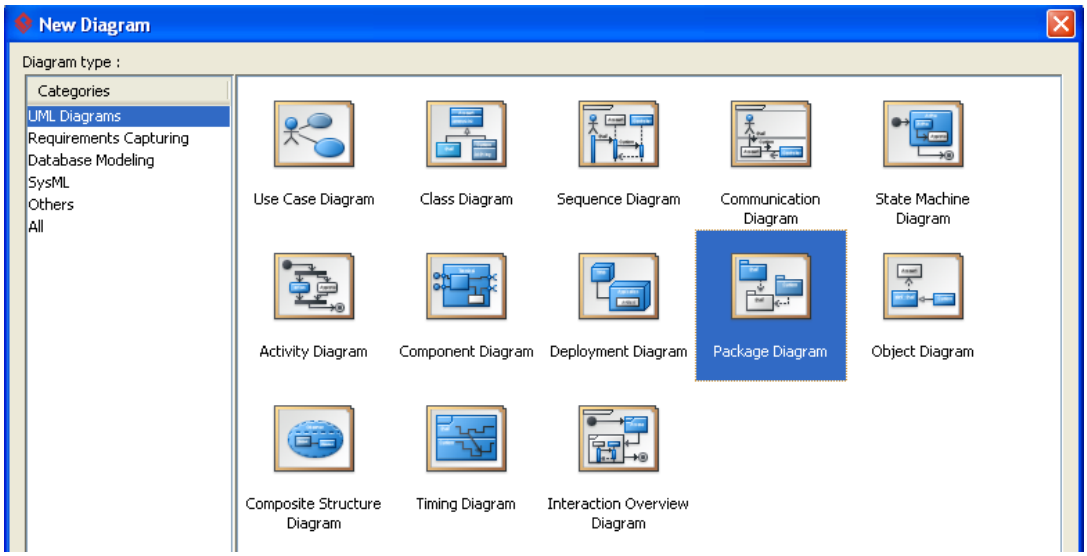
   b. As interfaces consumidoras são representadas por uma meia-lua (semicírculo) e elas solicitam informações de interfaces provedoras.

UNIDADE 3 – APROFUNDANDO NA UML
MÓDULO 3 – DIAGRAMA DE PACOTES

01

1 - MODELAGEM DE PACOTES

Olá, seja bem-vindo! Neste módulo trataremos dos diagramas de pacotes. Organizar o seu trabalho é uma das coisas mais importantes que você faz. Há uma série de dispositivos apresentados pela UML para ajudar a organizar pacotes, sistemas, subsistemas e modelos. Este módulo apresenta os dispositivos e fornece uma visão geral da abordagem do processo unificado (*Unified Process - UP*) para organizar o seu trabalho.



Tela de diagramas UML do Visual Paradigm, em destaque, o diagrama de pacotes

A construção de um **pacote** é a principal orientação da UML para o agrupamento dos elementos de um modelo. Um pacote oferece exatamente a mesma funcionalidade que uma pasta de arquivos no Windows. Ele permite que você organize elementos, agrupando-os e colocando-os em um local hierarquicamente organizado (os pacotes podem conter outros pacotes, assim você pode criar uma hierarquia de pacotes).

02

1.1 - Namespace

Namespaces (sem uma boa tradução para o idioma português, mas alguns autores traduzem para "espaço de nomes") é um nome que representa uma coleção de itens diversos a qual você cria um identificador único, ou seja, um nome exclusivo para o pacote.

Da mesma forma que utilizamos nosso sobrenome para identificar os membros da nossa família, podemos utilizar namespaces para agregar componentes semelhantes.

Um pacote UML também oferece um namespace para classificadores que você coloca no pacote. Isso significa que uma vez que você colocar um elemento em um pacote, o seu nome torna-se único para um elemento desse tipo nesse pacote. Você pode criar um elemento com o mesmo nome em um pacote diferente, e ele vai ter uma definição diferente (assim como seu conteúdo). Veja esta analogia em termos de Windows Explorer.

Assim como podemos criar quantas subpastas quisermos em uma hierarquia de arquivos e pastas, podemos criar quantos níveis hierárquicos desejarmos, representando cada subnível por um ponto (“.”) e o nome do subnível. Exemplo, suponha que você tenha uma estrutura hierárquica de pacotes que represente em primeiro nível o nome da empresa, depois o nome do sistema, depois o nome da funcionalidade, para esse exemplo, o padrão do namespace seria “empresa.sistema.funcionalidade.componente”.

Veja esta analogia
Você pode ter dois arquivos com mesmo nome em duas pastas diferentes e esses arquivos (apesar de possuírem o mesmo nome) poderão ter conteúdos completamente diferentes. Exemplo: suponha que seu pacote chama “SistemaX” e que dentro dele exista uma classe de nome “Cliente”, pelo fato de você ter colocado essa classe dentro do pacote, ela pode ser representada pelo namespace “SistemaX.Cliente”.

03

O padrão para nomear um namespace é “agrupador.item”, sendo que podemos ter namespaces que agrupam outros namespaces. Veja alguns exemplos de nomes válidos para namespaces:

- **SistemaCinema.Cliente** – namespace “SistemaCliente”;
- **SistemaCinema.ModuloPagamento.Cliente** – namespace “SistemaCinema” que agrupa outro namespace menor de nome “ModuloPagamento”;
- **Cinemark.SistemaCinema.ModuloPagamento.Cliente** – namespace da empresa “Cinemark” que agrupa/contém o sistema “SistemaCinema” que agrupa/contém o módulo “MóduloPagamento” que agrupa/contém a classe “Cliente”.

A Microsoft define um padrão para estruturar um namespace. Esse padrão é **NomeDaEmpresa.NomeDaTecnologia.Funcionalidade.Desenho**, sendo que “funcionalidade” e “desenho” são opcionais. Um exemplo de um namespace da própria Microsoft é o namespace “Microsoft.Media” que contempla todas as funcionalidades acerca da manipulação de arquivos multimídia.

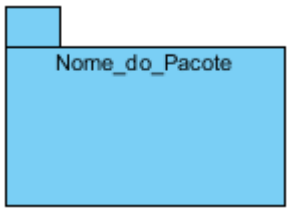
Fique Atento!

Criar namespaces padronizados ajuda a evitar namespaces com nomes duplicados. Por exemplo, suponha que exista um namespace de um serviço bancário público de nome BancoDoBrasil.ServicoBancario e que sua aplicação implemente outras funcionalidades especializadas de serviço bancário. Um nome padrão para você seria SuaEmpresa.ServicoBancario.

04

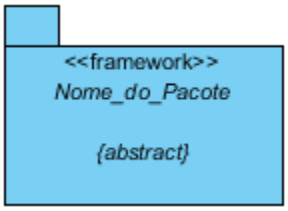
1.2 - Pacote

Um pacote é representado na UML pelo símbolo de uma pasta. O nome do pacote pode ser colocado no meio do símbolo, ou na aba do lado esquerdo superior, como mostrado abaixo. Ele normalmente é colocado na guia, se você está desenhando o conteúdo do pacote dentro do próprio símbolo pacote.



Layout padrão para um pacote

Um pacote pode ser **estereotipado**, tendo sua nomenclatura inserida acima do nome do pacote, usando o padrão "<<estereótipo>>". Caso o pacote seja do tipo **abstrato**, essa referência é feita pela palavra "{abstract}", incluindo as chaves, escrita abaixo do nome.



Exemplo de pacote com estereótipo "framework" e simbologia de abstrato

Um **estereótipo** é a generalização de um componente para definir características semelhantes. Exemplo

Abstração refere-se a identificar apenas o que é essencial para o componente funcionar, eliminando outras características desnecessárias.

Itens abstratos definem modelos (templates) para criar a estrutura inicial, mas incompleta, daquilo que se quer. Cada elemento criado a partir desse modelo implementará as funcionalidades desejáveis. Geralmente, um item abstrato define apenas o que ele é, e a sua implementação definirá como ele irá se comportar. Exemplo

Exemplo (estereótipo)
Exemplo: ao definir um equipamento como “modem”, criar um estereótipo significa identificar as características que são comuns a todos os Modens e juntá-las nesse componente.

Exemplo (abstração)
Exemplo, eu posso te um modelo abstrato do que é um modem, ali digo que ele se conecta com a linha telefônica, com o computador e com a energia. Depois, ao criar o meu modem em particular, vou agregar nele como eles funcionará.

05

1.2.1 - O que um pacote contém

Um pacote é simplesmente um mecanismo geral de agrupamento em que coloca classificadores UML, tais como definições de classes, definições de casos de Uso, definições de estado, as relações entre estes classificadores e os diagramas, etc.

Existem três tipos de elementos que você pode colocar em um pacote:

a) **Elementos de propriedade do próprio pacote** (criados especificamente para aquele pacote);
b) **Elementos se fundiram ou foram importados** de outros pacotes;
c) **E elementos acessados a partir de um outro pacote,** que basicamente visitam o pacote atual.

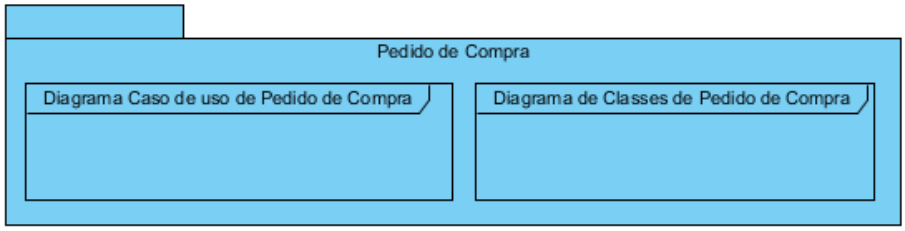
Podem existir outros pacotes dentro de um pacote, uma hierarquia infinita onde, cada subpacote, fornece um *namespace* único para todos os elementos dentro dele. Um pacote também pode conter vários diagramas UML de qualquer tipo.

06

1.2.2 - Notação para mostrar elementos de um pacote

A forma padrão para mostrar o conteúdo de um pacote é criar **hiperlink** a partir dele para outro diagrama, e mostrar o conteúdo do pacote neste diagrama.

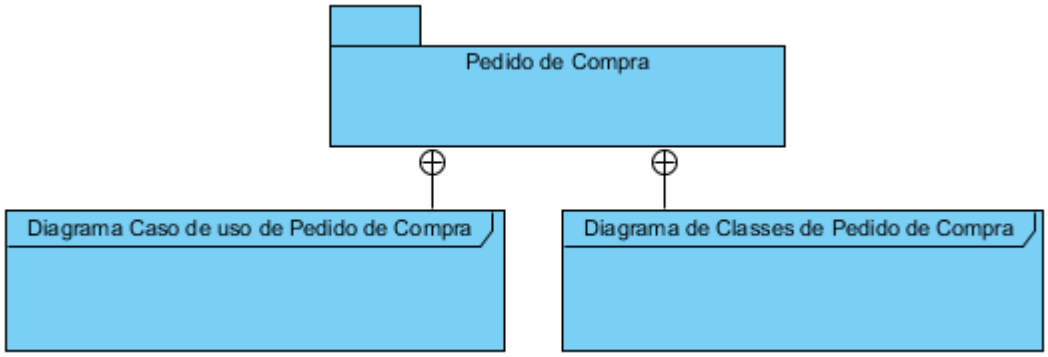
Entretanto, caso o conteúdo seja muito pequeno, você pode criar um ícone de pacote grande o suficiente para que o conteúdo seja diagramado dentro do próprio ícone do pacote (muito pouco usado na prática).



Representação do conteúdo de um pacote, em que o pacote "Pedido de Compra" possui dois diagramas, um de caso de uso e outro, de classes.

07

Outra forma de representar, também pouco usual, é desenhar os elementos fora o símbolo do pacote, e anexá-las a ele com linhas que são agrupadas no final do pacote por um círculo com um sinal de mais (+) para dentro, veja o exemplo a seguir:

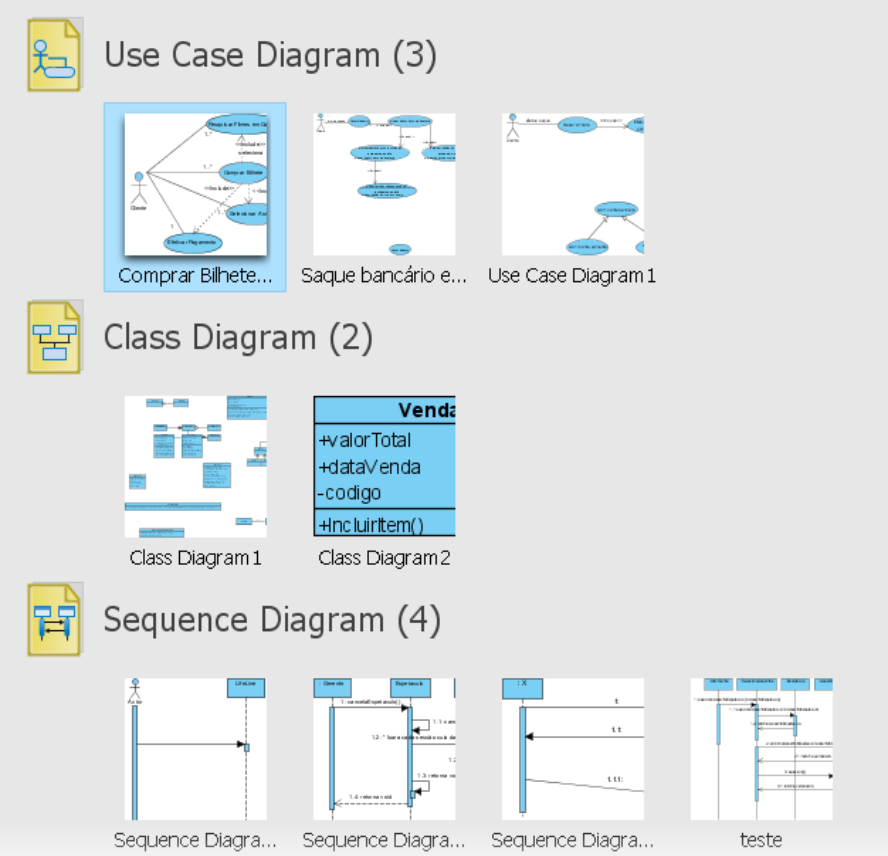


Representação do conteúdo de um pacote, em que o pacote "Pedido de Compra" possui dois diagramas, um de caso de uso e outro de classes.

É de se esperar que você utilize uma ferramenta de modelagem para criar os diagramas que pertencem a um pacote. Nesse caso, a própria ferramenta deve possuir mecanismos que apresentem o conteúdo de um pacote.

Normalmente é possível visualizar uma árvore hierárquica semelhante ao explorador de arquivos, onde você consegue identificar o que faz parte do pacote. Entretanto, na UML nem tudo é feito por meio de diagramas. Uma descrição de caso de uso, por exemplo, é normalmente um documento de texto. Algumas ferramentas possuem editores de texto internos que permitem gerenciar diagramas e documentos de texto, outras ferramentas não possuem essa integração com documentos de texto e você precisa mantê-las em outros mecanismos de controle e organização, como pastas de arquivos ou ferramentas de gerenciamento de arquivo.

Dessa forma, é necessário ter atenção especial quando for diagramar um pacote, pois pode ser necessário criar links ou referências a documentos que não estão sob controle do *software* de diagramação, mas sim armazenados em uma pasta de arquivos (por exemplo).



Exemplo de como uma ferramenta pode apresentar uma estrutura de diagramas UML

08

2 - Modelando a dependência de pacotes

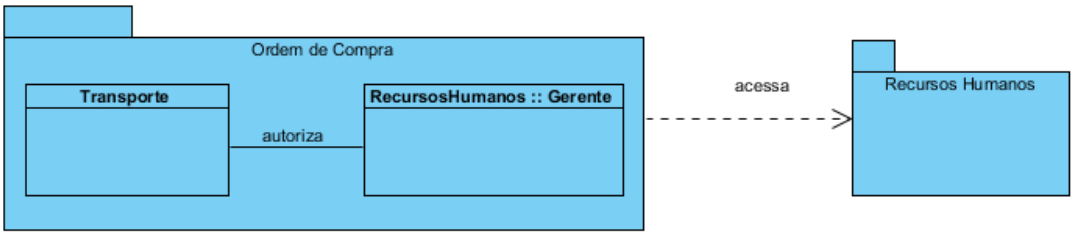
Quando você modela um sistema, é comum ser necessário criar um relacionamento entre um classificador de um pacote com outro classificador de outro pacote diferente. Para isso, é necessário que você importe o classificador do outro pacote no pacote que você está modelando. Há duas formas de fazer essa referência: por acesso ou por importação / mesclagem.

2.1 - Acesso

Quando você acessa um classificador de outro pacote, esse classificador permanece no seu pacote de origem, o que você faz é simplesmente construir uma relação com ele.

Para sinalizar que aquele componente vem de outro pacote é necessário nomear o componente referenciado com o padrão **NomeDoPacote::NomeDoClassificador**.

Veja o exemplo abaixo, onde o pacote “Ordem de Compra” acessa e utiliza o componente “Gerente” do pacote “Recursos Humanos”.



Exemplo de referência de acesso de um pacote em outro. O pacote "Ordem de Compra" acessa o componente "Gerente" que pertence ao pacote "Recursos Humanos"

Uma relação de acesso entre dois pacotes é desenhada como uma seta pontilhada apontando para o pacote que contém o componente acessado. A partir desse relacionamento, todos os componentes públicos do pacote referenciado podem ser acessados.

09

2.2 - Mesclagem e Importação

Na UML 1.x, um relacionamento do tipo "importação" significa que o seu pacote irá conter uma cópia do pacote importado. Entretanto, futuras alterações feitas no pacote importado não serão atualizadas automaticamente no seu pacote. Ou seja, mesmo parecendo idênticos, os itens do pacote original e do pacote que recebeu a importação são distintos. Não há como manter os pacotes sincronizados, qualquer modificação no pacote de origem não é automaticamente refletida no pacote de destino.

A UML 2.0 incorporou um comportamento diferente: ao invés de importar o pacote referenciado, ocorre uma **mesclagem**.

Mesclar significa juntar os dois pacotes em um só, sem distinção do pacote original e do pacote mesclado; ao mesclar, não é necessário trazer todo o conteúdo do pacote referenciado, você pode escolher aquilo que quer ser mesclado ao seu pacote.

Entretanto, a mesclagem exige que você resolva eventuais problemas de conflitos de nomenclatura dos itens relacionados. Pode ser que existam componentes com o mesmo nome em ambos os pacotes, esse conflito precisa ser resolvido de uma das maneiras que veremos a seguir.

10

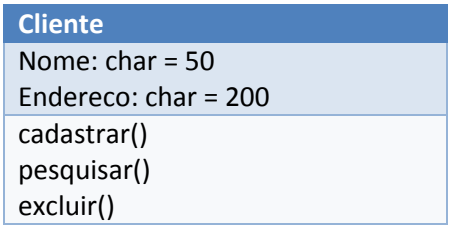
Solução 1

Caso você identifique que os itens conflitantes possuem a **mesma funcionalidade** (referem-se exatamente à mesma coisa), você tem a oportunidade de juntá-los em um único item.

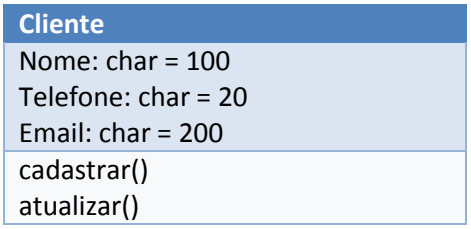
Exemplo: suponha que no seu modelo de origem possua uma classe de nome Cliente e que do pacote mesclado também exista uma classe Cliente, após a conclusão da mesclagem você poderá criar uma

única classe Cliente resultante da fusão das duas classes anteriores, ou seja, os métodos e atributos idênticos não serão duplicados, e os métodos e atributos exclusivos serão somados na classe resultante.

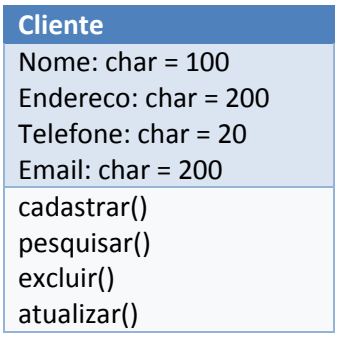
Assim, qualquer diagrama que referencie a classe Cliente está tratando exatamente da mesma informação. Eventuais conflitos de visibilidade, parâmetros etc., também devem ser ajustados. Quando este tipo de fusão ocorre, os itens recebidos e mesclados no seu modelo são considerados **especializações** dos itens do pacote original.



Classe de Origem



Casse do outro pacote



Casse final mesclada

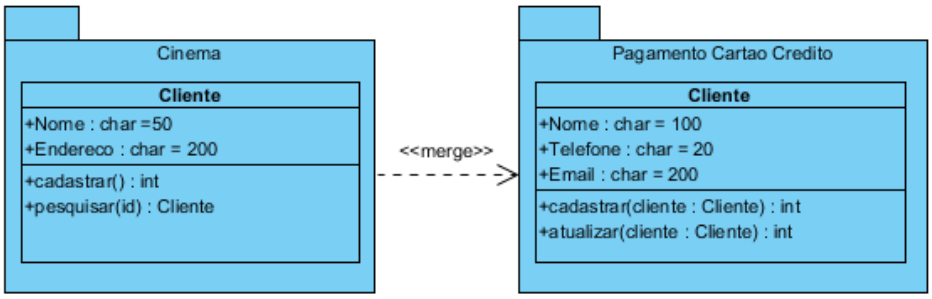
11

Solução 2

Caso você identifique que os itens conflitantes possuem **funcionalidades diferentes**, você tem a oportunidade de criar classificadores específicos para cada um e, assim, mantê-los independentes dentro do seu pacote.

Exemplo: suponha que exista um pacote chamado “cinema” que possua a classe Cliente e que foi mesclado com o pacote chamado “pagamento de cartão de crédito”, que também possui a classe Cliente, nesse caso poderiam ser criados os classificadores **Cinema::Cliente** e **PagamentoCartaoCredito::Cliente** para distinguir entre ambos.

A relação de mesclagem entre dois pacotes é desenhada como uma seta pontilhada, apontando para o pacote de origem. Sobre a seta insere-se o estereótipo «merge».



Diagramação de um merge

12

2.3 - Apagando um pacote

Quando você exclui um **pacote**, todo o seu conteúdo é excluído junto:

- diagramas,
- classes,
- fluxos,
- relacionamentos, etc.

Se o seu pacote foi referenciado em outro, os links e componentes utilizados em outros pacotes também serão excluídos.

Ao excluir um **diagrama**, os elementos que lá foram criados **não são excluídos** do pacote. Eles permanecem lá, podendo ser utilizados em outros diagramas.

13

3 - O DIAGRAMA DE PACOTE

Um pacote pode conter qualquer tipo e quantos diagramas UML forem desejados. Na UML 1.x, o diagrama de pacotes é elaborado como um diagrama estático qualquer, por exemplo, como um diagrama de classes. Já na UML 2.0, existe o diagrama de pacotes de modo específico, como um diagrama de estrutura. Se o seu diagrama contiver principalmente classes, você poderá chamá-lo de diagrama de classes, se o seu diagrama contiver principalmente objetos, você poderá chamá-lo de

diagrama de objetos; agora, se ele contiver pacotes, classes e objetos, você poderá chamá-lo do nome que você quiser.

Desenhar um diagrama de pacotes é uma atividade estritamente opcional e rara. Você não precisa criar diagramas de pacotes mostrando seus subpacotes, a menos que exista uma razão clara para esse nível de documentação.

Algumas ferramentas não possuem um diagrama específico para isso, visto que a hierarquia dos elementos e a relação “contém-está contido” é feita por meio de pastas de arquivos em que cada pasta representa um pacote. Muitas vezes, os diagramas de pacotes são usados para referenciar explicitamente o tipo de relação entre os pacotes (mesclagem, importação, acesso etc.).

Há um número infinito de maneiras de organizar o material dentro de pacotes. A especificação UML não fornece um método para classificar o seu trabalho. Há uma série de métodos publicados, você pode seguir como o Rational Unified Process (RUP), Open UP, Catalysis, Shlaer / Mellor, a Unified Modeling ICONIX Objeto Approach, e outros.

14

RESUMO

Neste módulo, aprendemos que:

a) O pacote é a principal componente da UML para o agrupamento de artefatos do modelo. Ele permite que você organize elementos, agrupando-os e colocando-os em um recipiente.

b) Um pacote oferece exatamente a mesma funcionalidade que uma pasta no Windows Explorer, ou seja, um lugar para guardar outros diagramas e eventualmente outras subpastas.

c) Um pacote fornece um espaço de nomes para classificadores que você coloca no pacote. Um pacote pode conter outros pacotes de uma hierarquia aninhada.

d) Um pacote pode conter vários diagramas de qualquer e todos os tipos de UML, além de outros pacotes menores (subpacotes).

e) Um pacote é representado pelo símbolo de uma pasta. O nome do pacote pode ser colocado no meio do símbolo, ou na aba superior esquerda da embalagem.

f) Há três formas de demonstrar que um elemento pertencente a um pacote: a) desenhar os elementos dentro de um grande ícone de pacote; b) desenhar os elementos do lado de fora do símbolo do pacote e ligá-los por meio de linhas que são agrupados no final do pacote por um círculo com a marca de + no seu interior; c) ou desenhar elementos em outro diagrama, e criar um hyperlink para anexado ao símbolo do pacote.

g) Classificadores permitem agrupar e rotular (nomear) itens. Também permitem controlar a visibilidade em relação a outros pacotes. A visibilidade pode ser pública ou privada.

h) Quando você constrói relacionamentos entre um classificador em um pacote (a origem) e um classificador em outro pacote (o destino), os pacotes criam um relacionamento entre eles.

i) Para modelar que você está acessando um classificador em outro pacote, desenhe o segundo classificador no pacote em que você está trabalhando, aponte ele por meio de uma seta tracejada e descreva o acesso no pacote de origem por meio do padrão "NomeDoPacote::NomeDoClassificador".

j) A relação de mesclagem entre um pacote (origem) e outro pacote (destino) permite importar uma cópia de os classificadores no pacote de destino para o pacote de origem. Será feita uma cópia do conteúdo desejado no pacote de destino, utilizando o namespace de origem como referência. Não haverá vínculo entre origem e destino, ou seja, futuras alterações na origem necessitam de atualizações do destino.

k) A relação de importação entre os pacotes especificados na UML 1.x é substituído pela relação mesclagem entre pacotes na UML 2.0.

l) Quando você exclui um pacote, você exclui todos os elementos contidos nele os quais é proprietário, incluindo pacotes aninhados e seus elementos de propriedade. Elementos que pertencem a outros pacotes e são referenciados tem apenas a referência (hyperlink) excluída, eles serão mantidos nos respectivos pacotes originais.

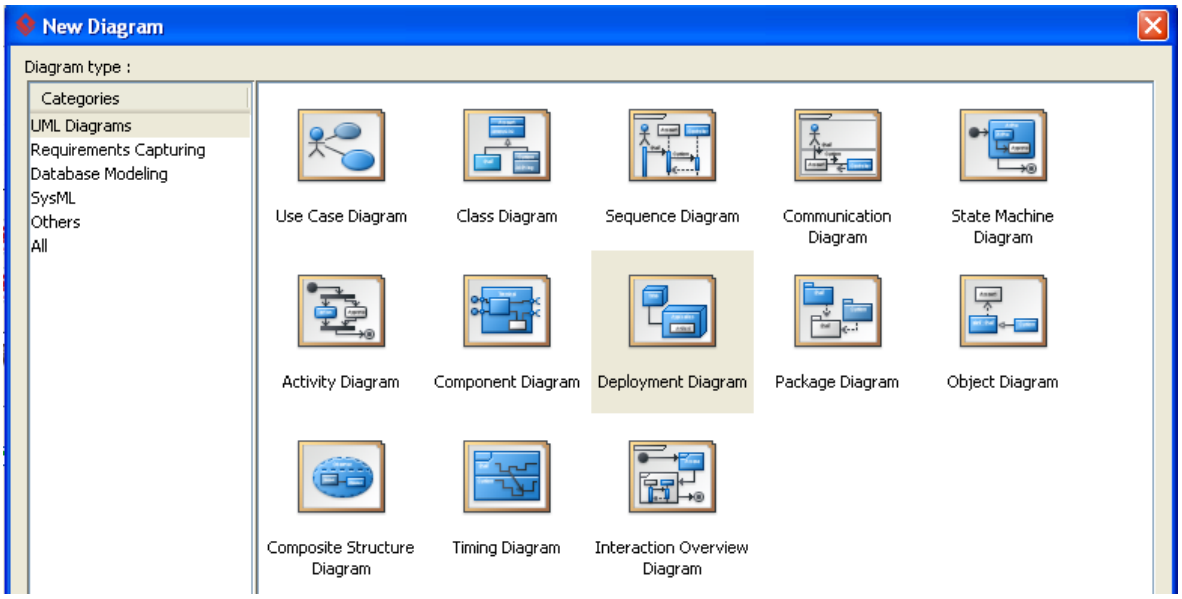
m) Um diagrama de pacotes é um diagrama do tipo estático, como o diagrama de classes, em que você desenha apenas os pacotes.

UNIDADE 3 – APROFUNDANDO NA UML
MÓDULO 4 – DIAGRAMA DE IMPLANTAÇÃO
01

1 - VISÃO GERAL SOBRE DIAGRAMAS DE IMPLANTAÇÃO

Olá, seja bem-vindo! Nesta unidade trataremos dos diagramas de implantação.

O diagrama de implantação descreve uma arquitetura de execução, que representa a configuração do *hardware* e do *software* onde o sistema será implantado, configurado e operado.



Tela de diagramas UML do Visual Paradigm, em destaque, o diagrama de implantação.

02

O propósito do diagrama de implantação é apresentar uma **visão estática** (uma foto) do ambiente de implementação. Uma visão completa da descrição do sistema irá necessitar vários diagramas de implantação, cada um focado em diferentes aspectos do gerenciamento do sistema. Para cada uma dessas visões, o diagrama de implantação será combinado com o diagrama de componentes para representar a relação entre o *hardware* e o *software*.

Exemplos:

- Para uma aplicação com arquitetura multicamadas, o diagrama de implantação deverá modelar a distribuição da aplicação em camadas, as conexões físicas e os caminhos lógicos de comunicação.
- Outro diagrama poderá focar em como os componentes de *software* estão distribuídos ao longo dos recursos da rede, como, por exemplo, o código fonte sai do ambiente de desenvolvimento e vai para os ambientes de teste, homologação e produção, tudo isso controlado pelo sistema de gerenciamento de configuração.
- Outro diagrama poderia modelar como um pacote é publicado em produção, controlando a interrupção temporária do sistema (se for o caso).

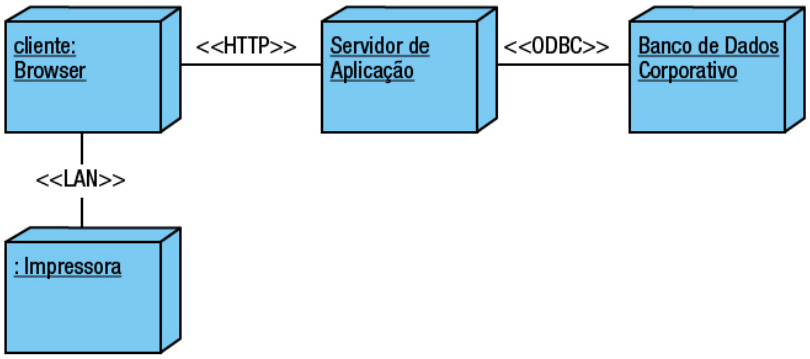
Um diagrama de implantação exibe a configuração dos nós de processamento em tempo de execução e os componentes que neles existem.

Um nó é um recurso computacional onde artefatos são colocados para execução. O diagrama representa a mesma informação que é normalmente modelada em um diagrama de redes, ou seja, os equipamentos e respectiva conectividade da arquitetura computacional.

03

A UML descreve o diagrama de implantação em dois contextos:

Um diagrama de implantação pode definir **tipos de nós**, algo semelhante a um diagrama de classes que define tipos de objetos.	Um diagrama de implantação pode **incluir instâncias de componentes** dentro das instâncias dos nós para ilustrar o ambiente de execução (seja ele desenvolvimento, teste, homologação ou produção).
Assim, temos o seguinte foco na diagramação: • Um nó representando um tipo de recurso de processamento, como uma CPU ou um trabalho humano (manual) realizando uma tarefa. • Uma associação definindo um tipo de relacionamento que suporte a comunicação entre os nós.	Dessa forma, a diagramação representa: • Uma instância de um nó como um recurso de processamento específico designado para realizar determinado tipo de tarefa. Exemplo: uma máquina específica onde funciona o banco de dados. • Um link entre os nós para especificar as conexões de um tipo pré-determinado. Exemplo: o tipo de conexão entre o servidor de banco de dados e o servidor da aplicação.



Exemplo de diagrama de implantação
Fonte: Bezerra, 2002

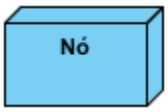
04

2 - Modelando nós e suas características

A informação mais simples que um diagrama de implantação pode fornecer é uma **modelagem do *hardware*** onde o sistema deverá funcionar. Sob essa ótica, um diagrama de implantação nada mais é do

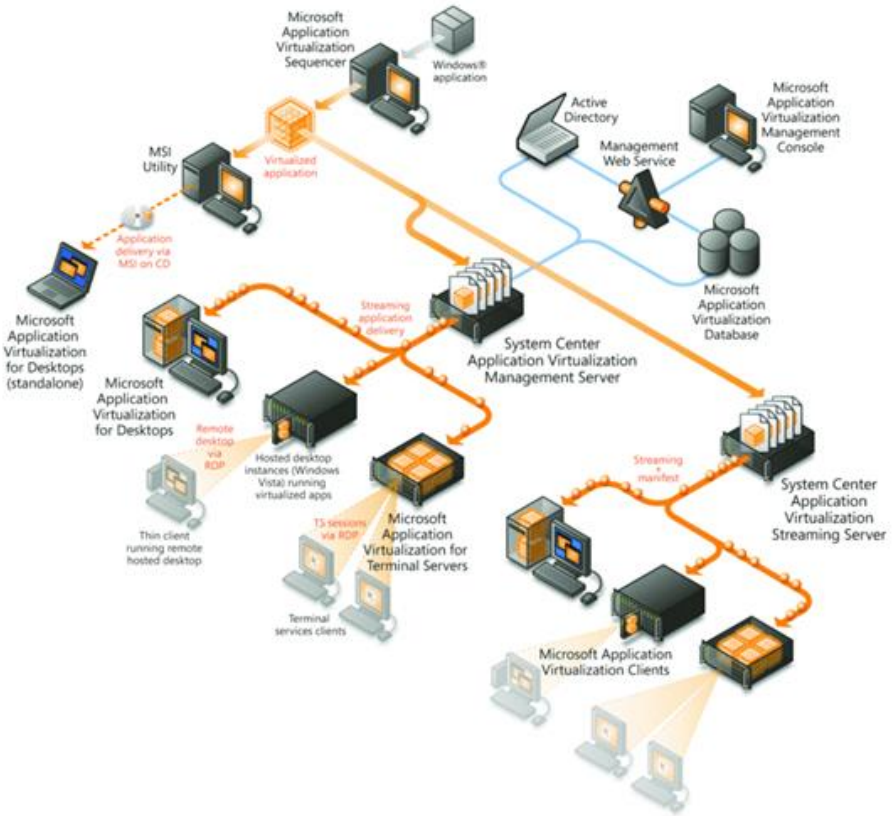
que um simples diagrama de redes, semelhante ao que é feito há anos por equipes de TI em outras ferramentas de diagramação (como o Ms Visio por exemplo).

O que mais difere são os ícones utilizados: enquanto na UML só temos ícones de caixas para representar um recurso computacional, nas outras ferramentas de diagramação temos vários desenhos diferentes que representam *racks*, *switches*, servidores, *storages*, *firewalls*, *modems* etc. Veja como um nó é representado na UML:

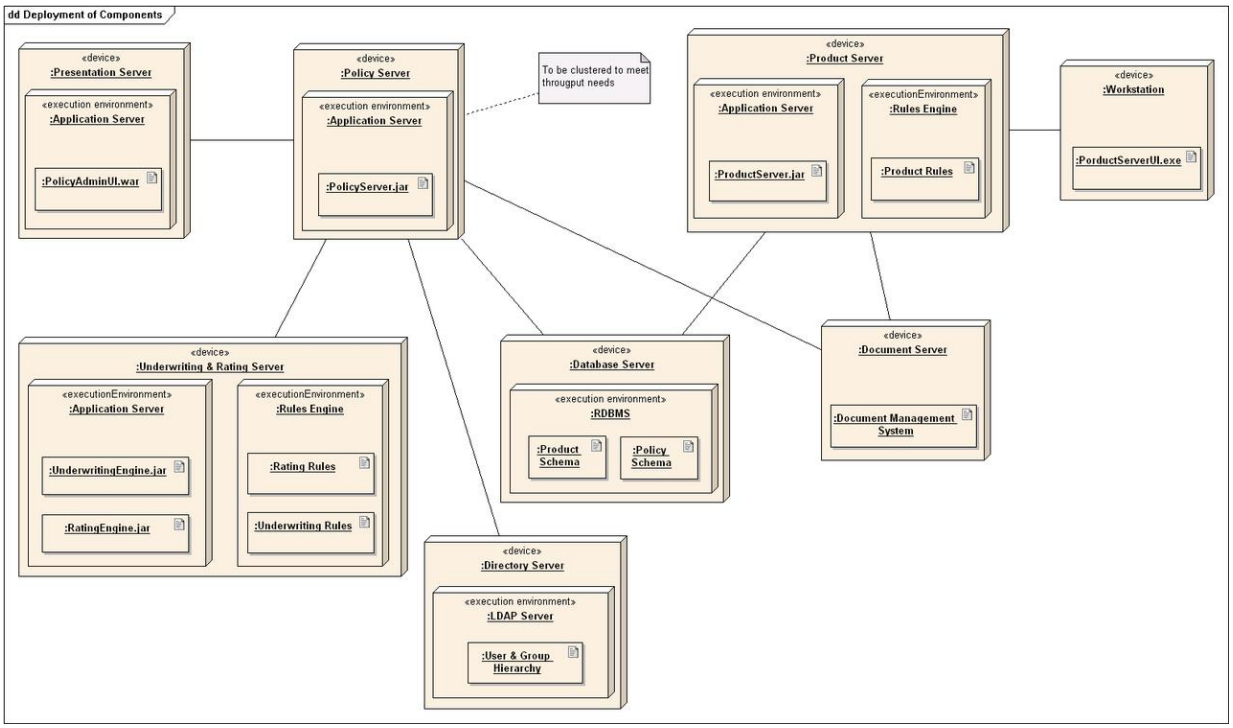


Layout de diagramação de um nó na UML

Esteticamente falando, essas outras ferramentas permitem diagramas desenhos muito mais "belos". Veja dois exemplos abaixo, um diagrama feito no Ms Visio e outro feito em ferramenta UML:



Exemplo de diagrama de arquitetura de *hardware* feita em Ms Visio



Exemplo de diagrama de implantação feita em UML

05

Para a UML, um nó representa qualquer equipamento, componente ou recurso que realize um determinado trabalho. Dependendo do sistema que você estiver modelando, um nó pode representar um servidor, uma memória, um processador, um *storage*, uma pessoa, enfim, qualquer coisa.

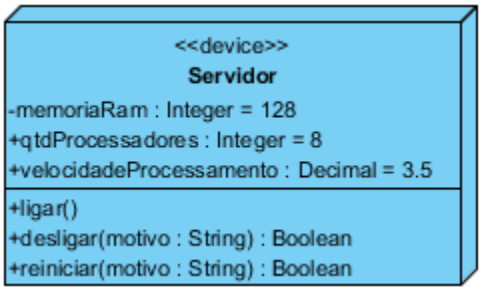
Um nó pode ser decomposto (ou detalhando) em subpartes menores, o que poderíamos chamar de "subnó" (mas esse nome não existe na UML). Ainda, podemos representar os componentes que compõem o nó.

Assim como as classes, um nó pode possuir atributos e operações, além de poder participar de associações. Isso significa que podemos definir **informações de um nó**, como, por exemplo:

- tamanho de memória do servidor,
- quantidade de processadores,
- velocidade de processamento etc.

Podemos também definir suas **funcionalidades**, como por exemplo: ligar (boot), desligar (*shutdown*), reiniciar etc. Esses detalhes podem ser extremamente importantes nos casos em que você poderá utilizar a documentação do projeto para apoiar a compra e a configuração de equipamentos de TI.

Veja o exemplo abaixo de um nó representando informações hipotéticas:



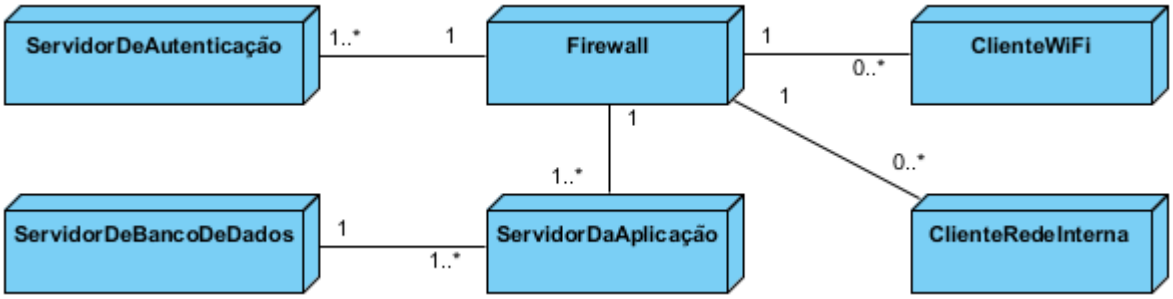
Exemplo de Nó com seus atributos e operações.

06

2.1 - Modelando associações

Assim como nos diagramas de classe, um diagrama de implantação permite modelar o relacionamento entre os nós. As características técnicas da UML para esta modelagem são as mesmas que já estudamos nos diagramas de classes, ou seja, podemos descrever regras de associação, multiplicidade etc.

Abaixo um diagrama simples para você relembrar como podemos descrever regras de multiplicidade:



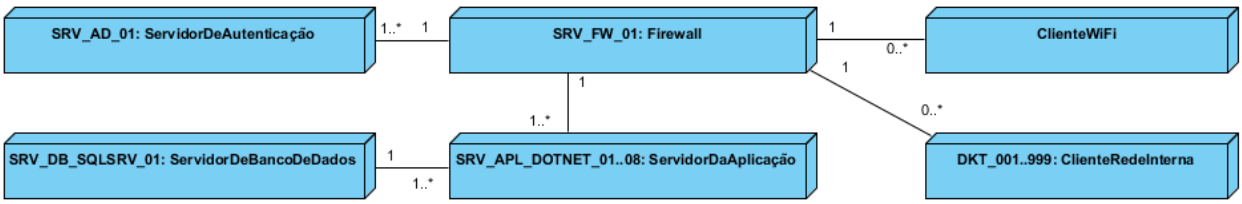
Exemplo de diagrama de implantação com associações e regras de multiplicidades

07

2.2 - Modelando instâncias

Da mesma maneira que o diagrama de objetos modela instâncias do diagrama de classes, podemos usar as mesmas regras para modelar uma instância do diagrama de implantação.

Veja o exemplo a seguir que representa, hipoteticamente, uma instância do diagrama do item anterior:

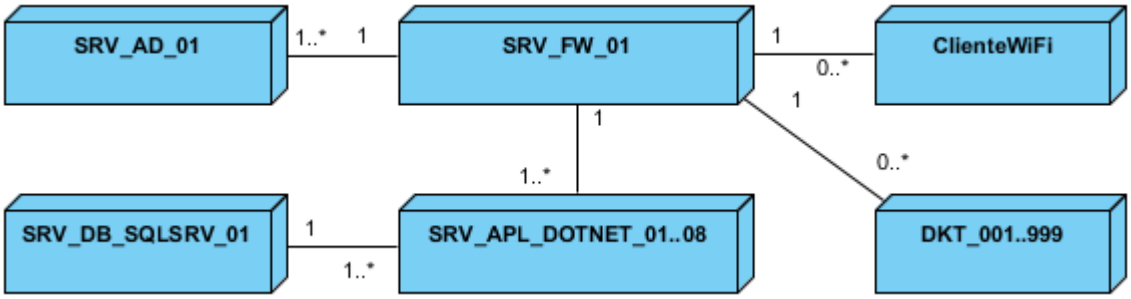


Exemplo de diagrama de implantação no nível de instância.

Lembre-se o mesmo padrão de nomenclatura utilizado no diagrama de objetos:

```
nome_do_nó : tipo_do_nó
```

Também podemos omitir o tipo do nó na nomenclatura da instância. Dessa forma, teríamos algo como o exemplo a seguir:



Exemplo de diagrama de implantação sem nome dos tipos do nó.

08

2.3 - Modelando estereótipos

Também podemos utilizar alguns estereótipos comuns às redes de dados, como <<Ethernet>>, <<TCP>>, <<WiFi>> etc.

Veja o exemplo a seguir:

Exemplo de relação com estereótipo

09

2.4 - Modelando componentes, comunicação e dependência

Quando um componente reside em um nó, ele pode ser modelado dentro do próprio nó. A comunicação entre os componentes é modelada por meio de uma linha contínua, já a dependência lógica entre os componentes é desenhada usando uma seta pontilhada, da mesma forma que você aprendeu quando estudamos os diagramas de componente.

Veja o exemplo a seguir:

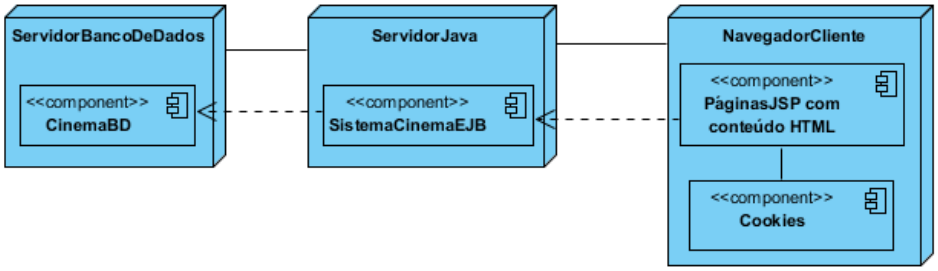


Diagrama apresentando dependência e comunicação entre os pacotes.

Os nós, componentes e dependências na figura acima representam os seguintes relacionamentos:

- O servidor de banco de dados contém o banco de dados “CinemaDB”.
- O servidor de aplicação executa o sistema “SistemaCinemaEJB”, que depende do banco de dados “CinemaDB”.
- O cliente rota páginas JSP, que dependem do sistema “SistemaCinemaEJB”.
- As páginas JSP e HTML comunicam-se com cookies, mas não há dependência, só comunicação.

10

2.5 - Modelando artefatos

Durante a transição da modelagem para a implementação, toda construção lógica do projeto deve ser mapeada para um elemento da implementação denominado **artefato**.

Cada artefato representa um pedaço do *software* que será endereçado em uma unidade de memória física (banco de dados, aplicação etc.). O artefato é uma manifestação de um ou mais elementos modelados.

Por exemplo, a classe `InterfaceUsuarioCompraIngressos` em um modelo pode se tornar uma página JSP/HTML definida para apresentar a tela de compra de ingressos do cinema.

Tecnicamente falando, um artefato é um **arquivo**, portanto, possui um nome. Além do nome, um artefato pode ser modelado como uma classe, pois possui outros atributos e mesmo operações.

Um artefato pode conter outros artefatos, por exemplo, o artefato que contém a interface do usuário para a compra de ingressos pode conter vários arquivos JSP, JPG, HTML, CSS e XML comprimidos dentro de um arquivo JAR.

Um artefato é modelado por um retângulo que incorpora o ícone de um artefato (semelhante a uma folha de papel), além do estereótipo `<<artefato>>`. Veja o exemplo abaixo:

Exemplo de modelagem de um artefato

11

Para descrever como um artefato é criado, a UML define um tipo especial de relacionamento chamado "manifestação". Esse relacionamento é modelado por meio de uma seta pontilhada com o estereótipo `<<manifest>>` entre o artefato e o elemento usado para sua construção. Veja o exemplo a seguir:

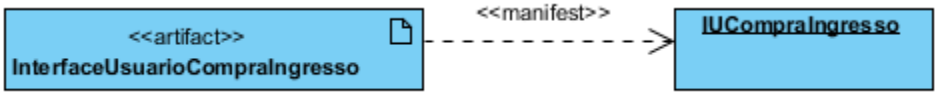


Diagrama representando a dependência da manifestação entre um artefato e o elemento da modelagem.

O exemplo acima expressa a seguinte modelagem: o artefato "InterfaceUsuarioCompraIngresso" é a manifestação (ou materialização física) da especificação contida na classe "IUCompraIngresso".

O interessante aqui é a separação entre o mundo lógico (a classe) do mundo físico (o artefato), enquanto o mundo lógico expressa características comportamentais, que são independentes de tecnologia, o mundo físico (o artefato) é o resultado da transformação da especificação sob uma tecnologia específica, ou seja, o artefato pode vir a ser um produto em Java, DotNet, ou outra qualquer, não importa, ele é a materialização da classe modelada. Isso permite facilmente mudar de uma tecnologia para outra.

Os estereótipos podem simplificar o trabalho na definição dos artefatos. Em um alto nível de abstração, artefatos podem ser distinguidos entre `<<source>>` (arquivo fonte) e `<<executable>>` (arquivo compilado/executável). Já em um nível mais especializado, os mesmos estereótipos podem ser ajustados para especificar claramente a tecnologia utilizada, como por exemplo: `<<html>>`, `<<JSP>>`, `<<EJB>>`, `<<Servlet>>`, `<<aspx>>`, `<<HTML>>`, etc.

12

2.6 - Modelando generalização

Da mesma forma que classes podem ser generalizadas em um tipo comum (superclasse), "nós" também permitem o mesmo tipo de modelagem. Modelar a generalização traz a vantagem de criar no nó "pai" os atributos e métodos comuns.

O exemplo abaixo apresenta um diagrama simples usando a generalização:

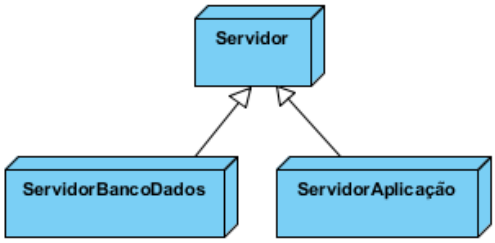


Diagrama de implantação representando generalização de "nós".

13

3 - IMPLANTANDO ARTEFATOS EM NÓS

Um artefato deve ser implantado para poder ser utilizado no mundo real. Ou seja, ele deve ser colocado em um local onde ele possa ser armazenado e executado. Esse local pode ser um nó (ou seja, um ambiente de execução ou um dispositivo), uma propriedade, ou uma especificação de uma instância. A relação entre o local e o artefato é denominada **relacionamento de dependência de implantação**.

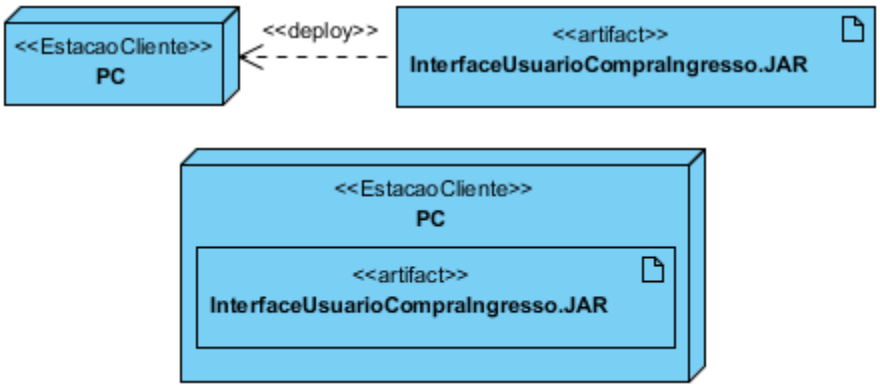
Uma implantação é modelada como uma dependência estereotipada. Essa dependência pode ser modelada de duas formas, conforme ilustrado abaixo.

No exemplo, o artefato "InterfaceUsuarioCompraIngresso" é implantado (transferido) para a "EstacaoCliente". As formas de modelagem são:

a) por meio de uma seta pontilhada com o estereótipo <<deploy>>;

b) com o desenho do artefato dentro do nó.

Veja a seguir:



Exemplo de modelagem de relação entre artefato e um nó

Um dos artefatos mais comuns que encontramos em nós refere-se à própria especificação de implantação, o que normalmente é feita por arquivos XML. Informações como modo de concorrência, modo de execução, modo de transação são exemplos de configurações muito frequentes.

RESUMO

Neste módulo aprendemos que:

a) O diagrama de implantação modela a arquitetura de *hardware* por meio da identificação dos recursos de processamento. Estes recursos são normalmente computadores ou equipamentos, mas também podem ser pessoas realizando trabalhos manuais.

b) Um componente define os requisitos para a implementação das funcionalidades especificadas em um diagrama de classes. Um componente realiza as funcionalidades das classes (classificadores).

c) Um artefato define a implementação de um componente em uma tecnologia específica. Um artefato manifesta as funcionalidades de um ou mais elementos em um pacote usados para descrever uma implementação física.

d) Um nó pode ser um equipamento ou um ambiente de execução dentro de um equipamento.

e) O caminho de comunicação é o caminho físico que conecta os nós entre si.

f) A especificação de implantação define como os artefatos são alocados em um nó, incluindo o modo de concorrência, o modo de execução e o modo transacional.

g) As dependências modelam os tipos de relacionamento entre os elementos do modelo que representam os diferentes níveis de especificação dentro do modelo. As dependências que aparecem tanto em diagramas de componentes quanto em diagramas de implantação são modeladas como uma seta pontilhada com um dos estereótipos a seguir:

    a. <<Realization>> um componente realiza ou descreve os requisitos para implementação e/ou as funcionalidades de um ou mais classificadores.

    b. <<Manifest>> um componente manifesta ou define uma implementação específica para as funcionalidades de um ou mais elementos físicos de *software*.

    c. <<Implement>> um artefato implementa ou define uma implementação para os requisitos definidos por um componente.

    d. <<Deploy>> um nó implanta ou provê espaço para o armazenamento/execução de um artefato.

h) Nós podem conter e executar componentes de *software*. As conexões entre os nós podem ser modeladas como associações, incluindo especificações de estereótipos e multiplicidade.

i) Um nó pode ser modelado utilizando as mesmas regras que usamos para modelar classes. Ou seja, todos os atributos e comportamentos são válidos para os nós. Entretanto, nem todos os *software*s de modelagem permitem documentar todas essas características.

j) As associações entre os nós tipicamente utilizam estereótipos ao invés de nomes para descrever a natureza da conexão As associações dos nós representam caminhos de comunicação física, como portas TCP ou Ethernet.

k) Diagramas de componentes e implantação podem ser combinados.

l) Os componentes residem em nós.

m) A comunicação entre os componentes de um nó é modelada como dependências. As dependências modelam os requisitos da comunicação lógica.

n) A dependência estereotipada pelo tipo <<become>> descreve a migração ou o movimento do *software* de um lugar para outro.