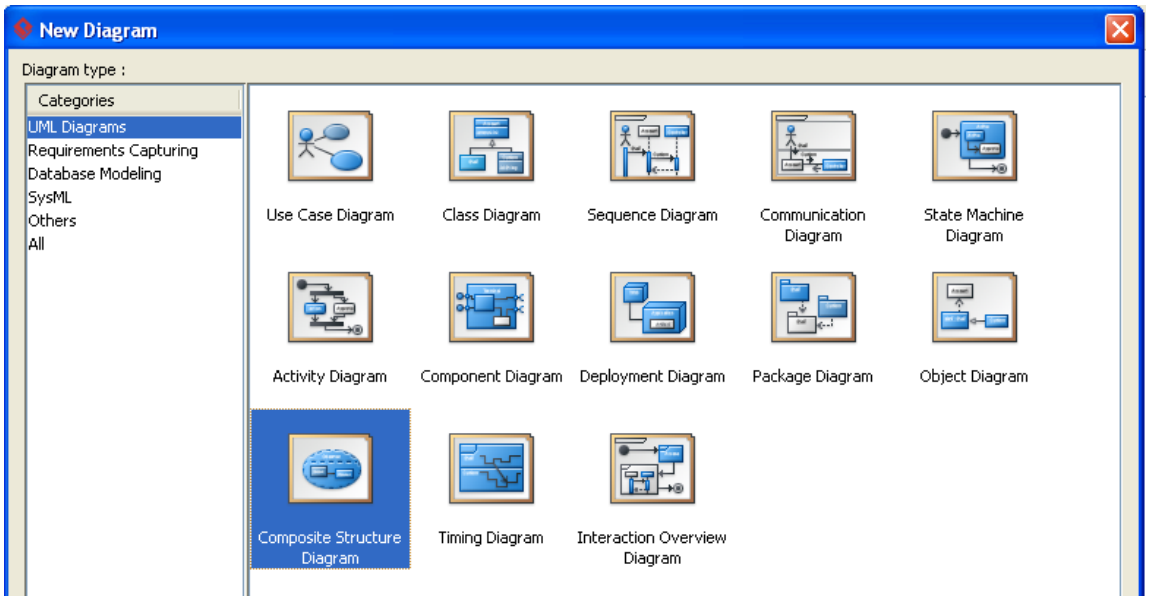
UNIDADE 4 – PASSOS FINAIS SOBRE UML
MÓDULO 1 – DIAGRAMA DE ESTRUTURAS COMPOSTAS

01

1 - VISÃO GERAL

Olá, seja bem-vindo a mais uma etapa do nosso estudo. Nesta unidade trataremos dos diagramas de estruturas compostas.



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

02

Um **diagrama de estruturas** compostas modela as partes de uma classe, componente, ou colaboração, incluindo os pontos de interação (portas) utilizados para acessar os recursos da estrutura.

O conceito visual vem do antigo diagrama de contexto composto, das primeiras da UML.

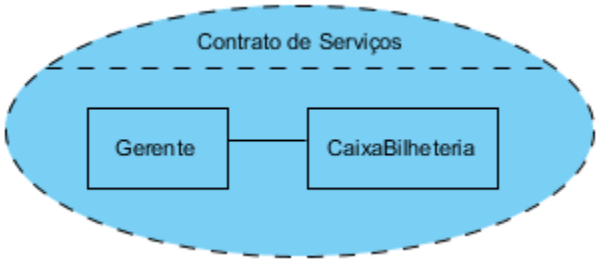
Na figura abaixo podemos ver um exemplo hipotético de um diagrama de estruturas compostas. Neste exemplo, é apresentada uma estratégia de preços para o sistema de venda de ingressos para um cinema. A estratégia de preços é construída usando um desconto e um conjunto de faixas de preço:

Exemplo de diagrama de estruturas compostas apresentando o relacionamento de duas partes

Os diagramas de estruturas compostas também modelam colaborações.

Uma **colaboração** descreve um comportamento, o recurso que realiza o comportamento e os papéis que os recursos assumem durante o comportamento.

A figura a seguir modela a colaboração na qual o gerente do cinema estabelece os serviços a serem realizados pelo caixa da bilheteria (perfil, funcionalidades e permissões):



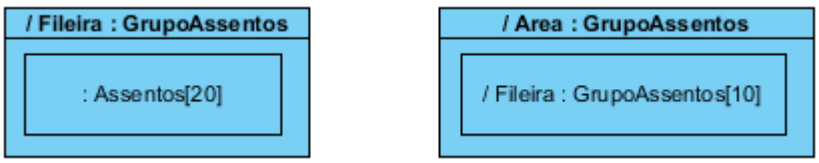
Exemplo de diagrama de estruturas compostas apresentando uma colaboração entre dois entes.

03

2 - Modelando as classes e suas partes

O diagrama de estruturas compostas modela uma classe como um grande retângulo com o nome da classe em um compartimento na parte superior do retângulo. Esta classe pode servir como uma espécie de container, onde outras classes podem residir no seu interior (chamadas de “partes”). Todas essas classes (sejam elas containeres ou partes) desempenham vários papéis, dependendo da finalidade do caso de teste específico.

No exemplo abaixo, ao lado esquerdo, temos uma classe container de nome “GruposDeAssentos”, desempenhando o papel de “Fileira”, que contém 20 classes partes de nome “Assentos”. Esse grupo de assentos representa uma fileira de poltronas do cinema. Ao lado direito temos outra classe container, de nome “GrupoDeAssentos”, desempenhando o papel de “Area”, que contém 10 classes partes (10 grupos de assentos) representando 10 fileiras:



Dois exemplos de classes containeres e classes partes.

Explicando em outras palavras, no exemplo da esquerda, o grupo de assentos está realizando o papel de uma fileira, que é constituído de um grupo de assentos; a multiplicidade é modelada pelo valor entre colchetes (“[20]”). Já no exemplo da direita, o grupo de assentos está realizando o papel de uma área do cinema constituída de várias fileiras (um setor ou sessão de poltronas), constituído de 10 fileiras de poltronas.

A UML não permite estruturas hierárquicas de mais de um nível, ou seja, não é possível criar uma classe container que contém outra classe container que contém uma classe parte.

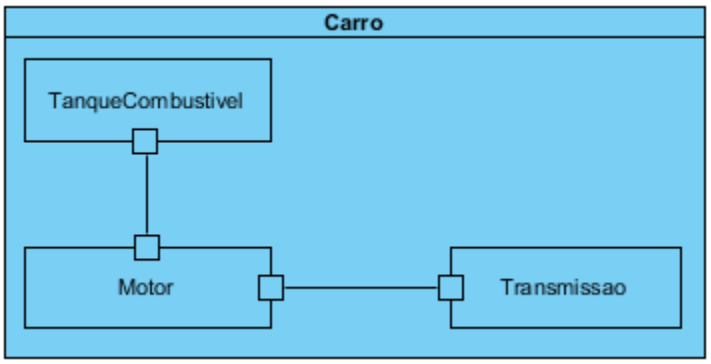
04

3 - Modelando conectores e portas

O diagrama de estrutura composta também modela a forma como as partes de um objeto agregado são conectadas para funcionar corretamente.

Por exemplo, um carro é construído com um motor, uma transmissão, e um tanque de combustível (e muitas outras partes). O motor deve receber combustível para funcionar, e deve ser ligado à transmissão, a fim de fazer o carro andar.

A figura abaixo modela a classe carro e três objetos partes: o motor, a transmissão e o tanque de combustível. Os conectores são as linhas entre cada par de peças e as portas são os pequenos quadrados em cada ponto de contato:



Exemplo de conectores e portas em um diagrama de estrutura composta

Uma porta define um ponto de interação distinto sobre o objeto. A porta pode especificar os tipos de interações em termos de interfaces requeridas (entradas) e interfaces fornecidas (saídas). A interface necessária descreve a necessidade do objeto para um serviço prestado por algum outro objeto. Todos os pedidos de serviços através de uma interface requerida são de entrada, o que significa que o objeto procura ajuda de algum outro objeto. Veja um exemplo.

Veja um exemplo

Por exemplo, o motor necessita de uma fonte de combustível, de modo que o motor precisa olhar para fora de si a encontrar uma fonte, tal como um tanque de combustível, e obter a sua ajuda. Uma interface fornecida descreve um serviço que o objeto oferece para outros objetos, tais como quando o motor fornece energia para a transmissão ou quando o tanque de combustível fornece combustível para o motor.

05

A porta é modelada como um quadrado no limite do objeto. A visibilidade padrão é pública. É possível, no entanto, desenhar o quadrado dentro do limite do objeto e declará-lo como privado, nos casos em que a interface só é usada internamente pelo objeto.

Uma porta pode especificar qualquer número de interfaces. O conjunto de interfaces define a porta. Na figura anterior, o motor define uma porta que requer uma interface de combustível. O tanque de combustível define uma porta que proporciona uma interface de combustível. Da mesma forma, a transmissão define uma porta que requer uma interface de potência, e o motor fornece uma interface de potência. A soma das portas e serviços associados define o limite do objeto. Em outras palavras, o objeto é encapsulado dentro do conjunto de interfaces. A encapsulação incentiva a reutilização.

Por exemplo, o tanque de combustível, no nosso exemplo, poderia ser usado com qualquer motor que requer o mesmo tipo de interface fornecida pelo tanque de combustível.

Você não precisa usar portas para ligar as partes. Você pode conectar os objetos mesmo sem existirem portas. Elas simplesmente fornecem um meio para encapsular a parte, de modo que os recursos das partes fiquem escondidos e as interfaces dos objetos tenham prioridade na discussão da modelagem. É decisão do profissional modelador se este nível de detalhamento ajuda ou dificulta o processo de modelagem.

06

Finalmente, existem dois tipos de portas:

- de sinal e
- serviço.

Sinal significa que a porta serve apenas para informar se aquele objeto está presente ou não, algo como uma resposta para “Ei, tem alguém aí?”.
Serviço significa que ali existe uma funcionalidade essencial, algo que alguém precisa.

Os dois tipos são mutuamente exclusivos. Se a porta estiver definida como “isSignal”, e o atributo isSignal estiver definido como verdadeiro, então essa porta só pode retransmitir sinais. Se a porta estiver definida como “isService”, e o atributo isService estiver definido como verdadeiro, então essa porta define os serviços prestados pelo classificador, como os serviços de combustível e energia no exemplo do carro.

Se isService for falso, a porta não é uma parte dos recursos publicados externamente no classificador, e pode ser alterada ou eliminada sem alterar a definição do classificador. Por exemplo, o motor também tem juntas e parafusos de montagem, mas eles não fornecem serviços para o motor ou outras partes do carro. Eles podem ser substituídos ou mesmo suprimidos em favor de outras tecnologias sem afetar o plano fundamental do carro. O mesmo não pode ser dito para o tanque de combustível, que fornece um serviço essencial. Excluí-lo ou substituí-lo requer um novo ajuste no serviço que ele estava provendo.

07

4 - Modelando uma colaboração

A **colaboração** representa como os elementos do modelo cooperam para executar algum comportamento essencial. Os elementos participantes podem incluir classes e objetos, associações e links, atributos e operações, interfaces, casos de uso, componentes e nós.

O comportamento pode ser um caso de uso, uma operação, um conjunto de operações, ou um mecanismo geral do sistema. Em outras palavras, a colaboração pode ser modelada em muitos níveis diferentes de abstração.

Quando uma colaboração descreve a implementação de uma única operação, ela inclui as classes, associações, interações, e os papéis que cada classe desempenha no comportamento. A colaboração pode ser modelada a partir de duas perspectivas:

Em primeiro lugar, a estrutura pode ser modelada com um **diagrama de classes** que mostra apenas as classes, associações, atributos e operações relevantes para a realização especificada pela colaboração.	Em segundo lugar, a perspectiva comportamental pode ser modelada com um **diagrama de interação**: um diagrama de sequência, um diagrama de colaboração, ou ambos.

08

Um uso comum para colaborações é **modelar padrões de projeto**. Um padrão modela um comportamento comum que requer um conjunto padrão de interações entre um conjunto de objetos que se encaixam com as funções definidas pelo padrão. Por exemplo, o padrão de design “Composite” define um modo para organizar objetos de complexidade variável, de modo que eles podem ser utilizados da mesma forma, independentemente da sua complexidade. A colaboração é modelada como um oval tracejado, com o nome da colaboração dentro.

A figura abaixo mostra o Composite (para o padrão de design) modelado como uma colaboração:

Padrão para modelagem de colaboração

A colaboração não especifica classes particulares, mas define os papéis que as classes possuem, a fim de alcançar o objetivo da colaboração.

Um papel, formalmente chamado de "**ClassifierRole**", define o conjunto de características que um classificador tem que possuir a fim de cumprir uma responsabilidade particular na colaboração.

Isso é muito parecido com definir as posições dos jogadores em um time de futebol. Podemos definir os papéis dos atacantes, da defesa e do goleiro, sem saber que quaisquer pessoas específicas poderão desempenhar essas funções.

O ClassifierRole também pode definir valores de atributos e comportamentos que uma instância deve possuir para desempenhar o papel. Por exemplo, um goleiro deve ser capaz de saltar determinada distância e atender a critérios específicos de altura. Veja outro exemplo.

Exemplo
Um exemplo prático seria um componente capaz de reproduzir diversos tipos de arquivos multimídia, seja ele vídeo ou somente áudio. Você poderia descrever padrões mínimos de resolução, quantidade de frames por segundo etc. para certificar que o arquivo será compatível com o próprio hardware que reproduzirá a imagem e o som.

09

A notação para um ClassifierRole usa a seguinte sintaxe:

NomeDoObjeto / NomeDoPapelDoClassificador : NomeDoClassificador [,NomeDeOutrosClassificadores]

Por padrão UML, o nome inteiro deve ser sublinhado quando se refere a uma instância. Quando apenas o nome da função é necessário e nenhuma referência à instância é implícita, não é necessário sublinhar o nome.

A seguir apresentamos exemplos de descrição completa e nomes alternativos para objetos e papéis:

- : ClasseBanco – BancoDoBrasil

- : ClasseBanco – PagamentoDebitoContaCorrente
- ObjetoBancoTransferenciaBancaria : ClasseBanco
- ObjetoBanco / TransferenciaBancaria: ClasseBanco
- ObjetoBanco / TransferenciaBancaria
- / TransferenciaBancaria - BancoDoBrasil
- / TransferenciaBancaria – TransferênciaEntreContasCorrente
- ObjetoBanco
- / TransferenciaBancaria: ClasseBanco – BancoDoBrasil
- / TransferenciaBancaria: ClasseBanco – TransferênciaEntreContasCorrente

Os papéis descrevem um objeto que, na verdade, é uma montagem de vários objetos. O componente então é basicamente uma interface, ou seja, a visão que outras classes e a aplicação vê. O programador também vê e manipula seus componentes internos por intermédio destas interfaces. Dessa forma, objetos simples e complexos são manipulados da mesma maneira, sem que o programador necessite entender a estrutura interna de cada um.

: ClasseBanco – BancoDoBrasil

Uma instância anônima da classe Banco.

: ClasseBanco – PagamentoDebitoContaCorrente

Usado para modelar o fato de que qualquer instância do tipo ClassBanco poderia cumprir os requisitos da parte especificada na colaboração.

ObjetoBancoTransferenciaBancaria : ClasseBanco

Uma instância chamada ObjetoBancoTransferenciaBancaria do tipo ClasseBanco. Usado para modelar o fato de que uma instância particular, do tipo ClasseBancoDoBrasil, cumpre os requisitos da parte especificada na colaboração. Ou seja, que o objeto em questão consegue realizar uma transferência bancária.

ObjetoBanco / TransferenciaBancaria: ClasseBanco

Mesma ideia do exemplo anterior, uma instância chamada ObjetoBanco, do tipo ClasseBanco, capaz de resolver o papel de uma TranferenciaBancaria. Em outras palavras, modela o fato de que uma instância específica, de um tipo específico, se comporta de uma maneira específica, e cumpre os requisitos da parte especificada na colaboração.

ObjetoBanco / TransferenciaBancaria

Mesma ideia do exemplo anterior, uma instância chamada ObjetoBanco de um tipo desconhecido de classe, realizando o papel TransferenciaBancaria (sabemos o nome do objeto, mas não o tipo). Usado para modelar o fato de que uma instância específica, comportando-se de forma específica, cumpre os requisitos da parte especificada na colaboração.

/ TransferenciaBancaria - BancoDoBrasil

Uma instância de um tipo não específico, desempenhando o papel de transferência bancária.

/ TransferenciaBancaria – TransferênciaEntreContasCorrente

Usado para modelar o fato de que a solução exige que qualquer tipo de objeto possa cumprir o papel, desde que ele esteja em conformidade com as regras para o papel.

ObjetoBanco

Uma instância denominada ObjetoBanco de um tipo desconhecido.

/ TransferenciaBancaria: ClasseBanco – BancoDoBrasil

Uma instância anônima do tipo ClasseBanco desempenhando o papel de TransferenciaBancaria.

/ TransferenciaBancaria: ClasseBanco – TransferênciaEntreContasCorrente

Uma instância anônima como a anterior, modelando o fato de que a solução vai funcionar com qualquer instância desse tipo que desempenhar este papel.

10

5 - Modelando uma ocorrência de colaboração

UML 2.0 continua com o conceito fundamental de que **uma colaboração descreve a estrutura e comportamento essencial para realizar uma tarefa padrão do sistema**. A colaboração pode especificar os atributos necessários e operações, a comunicação entre os objetos e os relacionamentos necessários. Mas todos estes requisitos são demonstrados de uma forma que não obriga ou pressupõe quaisquer classes particulares.

Na verdade, a colaboração é frequentemente expressa em termos de interfaces em vez de classes. Qualquer classe que possa apoiar as interfaces necessárias poderá participar da colaboração.

Para explorar essa abordagem, UML 2.0 introduz o conceito de uma **ocorrência de colaboração**.

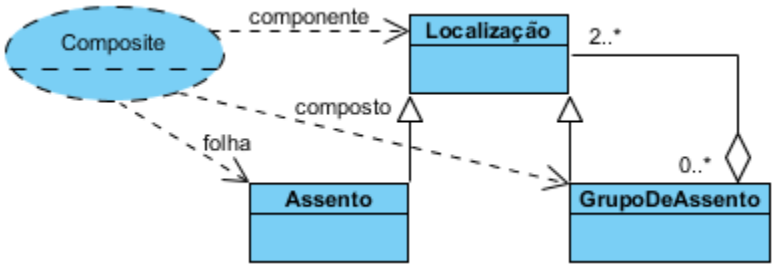
A ocorrência de colaboração é uma única instância de colaboração que está ligada a um conjunto específico de elementos, isto é, classes e associações. Portanto, pode haver qualquer número de ocorrências de colaboração para qualquer colaboração.

Por exemplo, no sistema do cinema, poderíamos usar a colaboração composta para combinar grupos de assento, sessões, estratégias de preço, e assim por diante para formatar cada uma das tecnologias de venda de ingresso disponível. Os papéis, regras e requisitos para os participantes são idênticos em cada caso, embora as classes e aplicações reais possam variar substancialmente. Ou seja, tanto o sistema que venderia ingressos pela Internet, quanto o sistema que vende ingressos no totem de autoatendimento, quanto o sistema de venda de bilhetes na bilheteria do sistema, todos os três que possuem implementações diferentes, tratam das mesmas ações e operações. São arranjos arquitetural e tecnologicamente diferentes que agem com o mesmo propósito.

11

Fora a colaboração de ocorrência, a única mudança na UML 2.0 é que ela usa o diagrama de estrutura composta para modelar a estrutura de colaboração. Além disso, uma vez que a colaboração é por si só um classificador, ela pode usar qualquer tipo de diagrama comportamental para representar seus requisitos comportamentais, tais como uma máquina de estado ou diagrama de sequência.

Como exemplo de uma ocorrência de colaboração, a figura abaixo representa um conjunto de classes do sistema de teatro para explorar os benefícios do padrão de projeto composto para trabalhar com várias combinações de assentos:



Aplicando a estrutura de colaboração para o sistema de cinema

Para aplicar um padrão a classes já existentes, cada classe assume um dos papéis de colaboração. A seta de dependência aponta para a classe que assume o nome do papel desempenhado. A imagem acima modela o padrão aplicado utilizando três classes do sistema de cinema utilizadas para definir a combinação de assentos. Um assento individual é uma folha. Um agrupamento de assentos, tais como uma fileira de assentos ou uma seção composta de várias fileiras é um objeto composto. Uma localização é um componente que pode ser ou um assento individual ou um grupo de assentos. Usando esse padrão, um usuário pode reservar uma seção inteira tão facilmente como reserva de um único assento.

Dentro de uma colaboração, a seta de dependência é chamada de **conector**. O conector liga um papel a um elemento do modelo. Exemplo

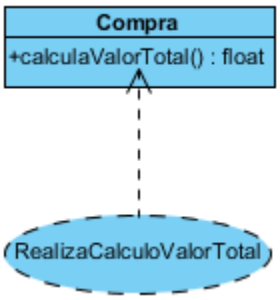
Exemplo

Por exemplo, na imagem anterior o papel “componente” está ligado à classe de localização. A colaboração define os papéis. A existência de uma colaboração liga ou mapeia os papéis para classes específicas que cumprem a finalidade da colaboração. No exemplo anterior, a colaboração é realizada pela “Localização”. Todos os papéis de colaboração são contabilizados por um conjunto de classes, associações e generalizações.

12

Uma colaboração pode também modelar a implementação de uma operação individual.

A figura abaixo mostra como associar uma colaboração com a operação que realiza / implementa.



Modelando uma colaboração que realiza uma operação

Esta representação é mais valiosa quando a implementação é complexa. A colaboração pode incluir diagramas estruturais e comportamentais para explicar a colaboração.

A vantagem de usar uma colaboração para descrever as exigências é a separação da aplicação dos requisitos. Esta separação permite a modelagem, construção e manutenção da aplicação, sem alterar ou perder de vista o requisito original.

O exemplo acima significa nunca perder de vista a necessidade de calcular o valor total da compra, mesmo que você precise mudar o algoritmo que faz isso uma dúzia de vezes ao longo da vida da aplicação. Ou seja, é uma espécie de lembrete para você manter o foco da funcionalidade, mesmo que a lógica mude, que as fórmulas mudem, o método deve sempre desempenhar aquele papel esperado. Veja um exemplo mais real.

Algumas ferramentas de UML implementam o diagrama de estruturas compostas dentro do diagrama de classes. Outras apresentam diagramas específicos para estruturas compostas, porém pode ser que nem todas as funcionalidades presentes na modelagem de diagrama de classes estejam presentes.

Veja um exemplo

Veja este exemplo mais real:

Suponha que você comece a modelar um sistema que trabalha com reservas de viagens, e no primeiro momento o seu sistema venda apenas diárias de hotel. Há nele uma funcionalidade final onde o cliente efetua o pagamento. A regra de fazer esse pagamento pode ser apenas uma pré-reserva, sem descontar nada no cartão de crédito, ou descontar uma diária, ou efetuar o pagamento completo por todas as diárias de hotel. Não importa a regra de negócio, o método deve ser capaz de efetuar a reserva para o cliente. Como a reserva será feita (por uma das três possibilidades apresentadas) será apenas uma regra de negócio implementada dentro do componente acionado pelo pagamento. Ainda, futuramente, quando o seu sistema também fizer reservas de carros e passagens aéreas, a classe que faz o pagamento terá a mesma forma de ativação, será a implementação que ela disparar que realizará o que foi previsto como regra de negócio.

13

Resumo

Neste módulo, aprendemos que:

a) A UML suporta a descrição dos objetos da mesma forma que fazemos com as classes. Para modelar objetos que são construídos por junção de outros objetos, a UML define o

diagrama de estrutura composta, que modela montagens usando a inserção visual de um objeto dentro de outro.

b) O diagrama de estrutura composta é uma alternativa para modelar os relacionamentos de agregação e composição. Classes que representam tipos de objetos parciais (filhas) são colocadas dentro da borda do ícone da classe agregada (pais) para representar contenção física.

c) Portas definem as interfaces entre as partes dentro de um diagrama de estrutura composta.

d) A UML modela o comportamento de objetos em situações específicas, utilizando diagramas de interação. Mas a UML também suporta uma colaboração, que define um padrão de interações entre objetos e as interfaces necessárias para fazer o trabalho de interação.

e) Colaborações podem ser utilizadas em qualquer dos outros diagramas da UML para definir um padrão previsível de classificadores e interações, sem identificar os tipos específicos de classificadores que preencham os requisitos de colaboração eficazmente, separando a especificação do sistema da implementação.

f) A colaboração modela um tipo padrão de comportamento dentro de um sistema. O comportamento é modelado como um conjunto de classificadores, suas relações e as interações entre eles. A colaboração é definida em termos dos papéis que os classificadores participantes devem estar em conformidade a fim de participar na colaboração. A colaboração pode ser realizada/implementada por um conjunto de classificadores que estejam em conformidade com os tipos e as interações definidas pela colaboração. Cada classificador assume um papel definido pela colaboração e implementa as interações especificadas para esse papel.

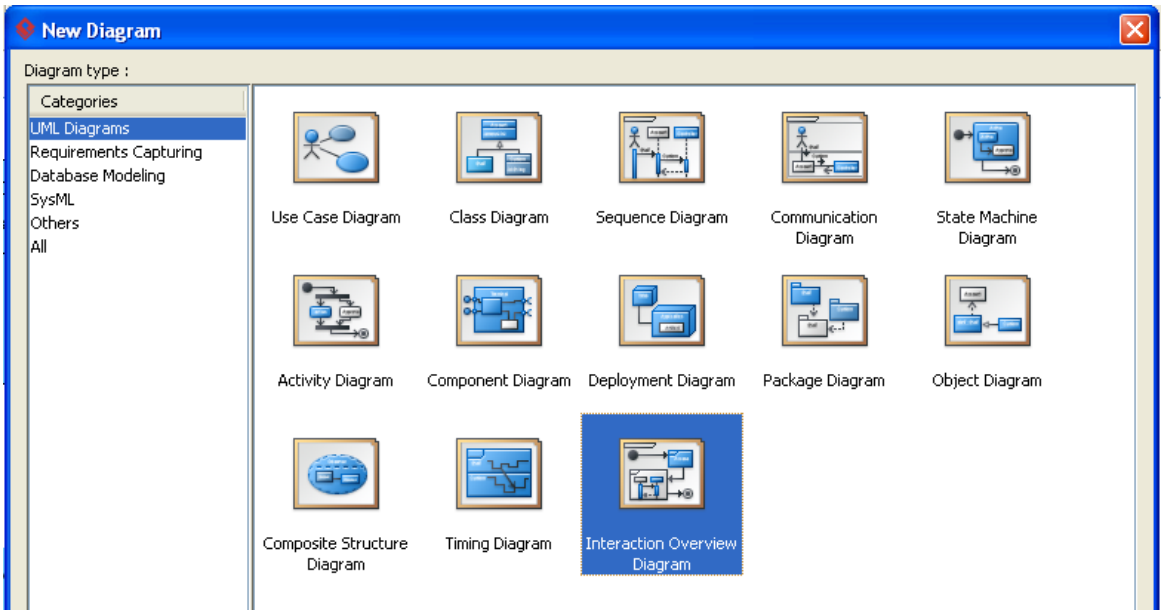
UNIDADE 4 – PASSOS FINAIS SOBRE UML

MÓDULO 2 – DIAGRAMAS DE VISÃO GERAL DE INTERAÇÃO E DE TEMPO

01

1 - DIAGRAMA DE VISÃO GERAL DE INTERAÇÃO

Olá, estudaremos agora dois diagramas: diagrama de visão geral de interação e diagrama de tempo.



Tela de diagramas UML do Visual Paradigm, em destaque, o diagrama de visão geral de interação ao lado do diagrama de tempo (Timing Diagram)

Em uma aplicação convencional, normalmente é suficiente limitar o diagrama de sequência tendo como base um cenário, ou seja, um caminho que atravessa o caso de uso. Dessa maneira, é fácil perdermos a visão do projeto inteiro, pois não vemos as interações entre os casos de uso, e a forma como eles interagem entre si (excluindo aqui as inclusões e extensões, e focando no uso do sistema pelo usuário final).

A UML 2.0 oferece o **diagrama de visão geral de interação** como um meio de combinar a controle de fluxo de um diagrama de atividades com as especificações das mensagens de um diagrama de sequências.

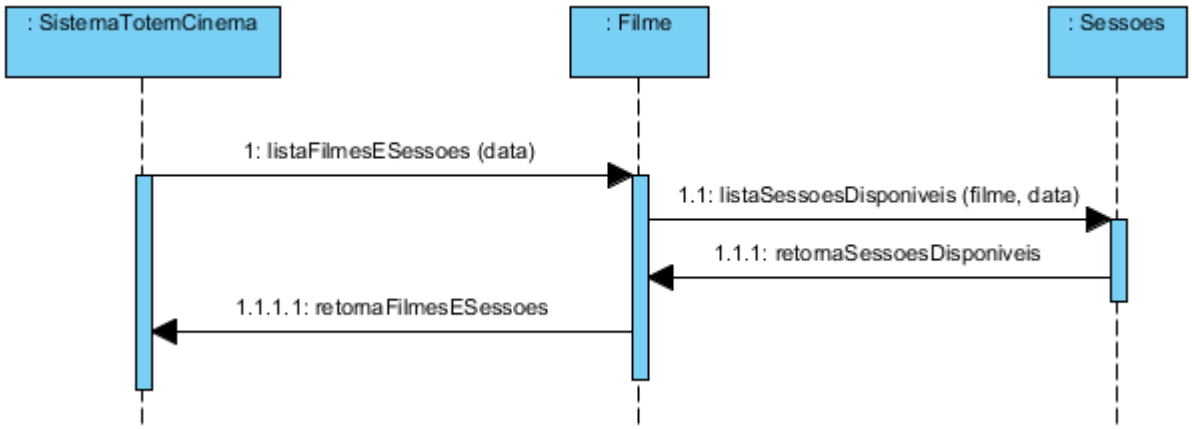
Quando um diagrama de atividades usa atividades e fluxo de objetos para descrever um processo, o diagrama de visão geral de interação usa interações e ocorrências. As linhas de vida e mensagens encontradas em um diagrama de sequências aparecem apenas dentro das interações ou ocorrência de interações. Entretanto, as linhas de vida dos objetos que participam nos diagramas de visão geral de interação podem ser listadas junto com o nome dos diagramas.

02

1.1 Modelando interações e ocorrência de interações

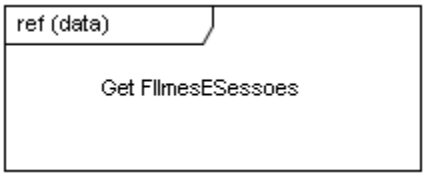
Quando falamos de **interação**, estamos tratando da relação entre o usuário e o sistema, ou seja, como o usuário opera o sistema. Já quando falamos de **ocorrência de interação**, estamos tratando de uma interação referenciada em outro quadro de interação, ou seja, um fragmento comum que pode ser reutilizado em outros diagramas.

A notação para uma interação e uma ocorrência de interação é a mesma que já aprendemos quando falamos de diagrama de sequência. A figura abaixo apresenta um exemplo de notação para interação, onde o usuário solicita ao sistema informações sobre filmes e horários das sessões:



Exemplo de notação de interação

A figura a seguir apresenta um exemplo de notação de ocorrência de interação. Observe que retornar quais filmes e sessões estão disponíveis é um pedaço reutilizável de uma interação com o usuário:



Exemplo de notação de ocorrência de interação

Ambas, interação e ocorrência de interação, são possíveis de serem modeladas em um diagrama de visão geral de interação, mas, por um propósito de simplificação, os exemplos que veremos neste módulo tratarão apenas de ocorrências de interação (entenda isso como “um pedaço” de um diagrama mais complexo).

03

1.2 Modelando o sequenciamento de interações

As interações podem ser sequenciadas assim como as atividades. Utilize as setas contínuas para modelar o fluxo entre uma interação e a próxima.

Lembre-se das seguintes regras ao modelar fluxos de interação:

1-	Os nós inicial e final iniciam e terminam o fluxo.
2-	Nós de decisão e mesclagem guiam o fluxo por meio de caminhos lógicos e mesclam-se para

uma sequência comum.

3- Nós do tipo garfo e junção suportam o comportamento concorrente.

04

1.3 Nós do tipo garfo e junções

Os exemplos que utilizaremos neste módulo tratam do processo de selecionar uma sessão de cinema para assistir. No início do processo, o usuário recebe uma lista de todos os filmes disponíveis e respectivos horários que irão ocorrer nos próximos sete dias (é possível comprar um bilhete para um filme que será passado até sete dias à frente).

Não existe uma sequência obrigatória entre os dois comportamentos “mostrar filmes e horários” e “escolher sessão de cinema”, dessa forma, nós podemos utilizar um garfo e uma junção para permitir a execução concorrente das duas ocorrências de interação.

A imagem a seguir apresenta a notação de garfo e de junção para o exemplo proposto:

Exemplo de notação de garfo e junção para mostrar duas interações concorrentes.

05

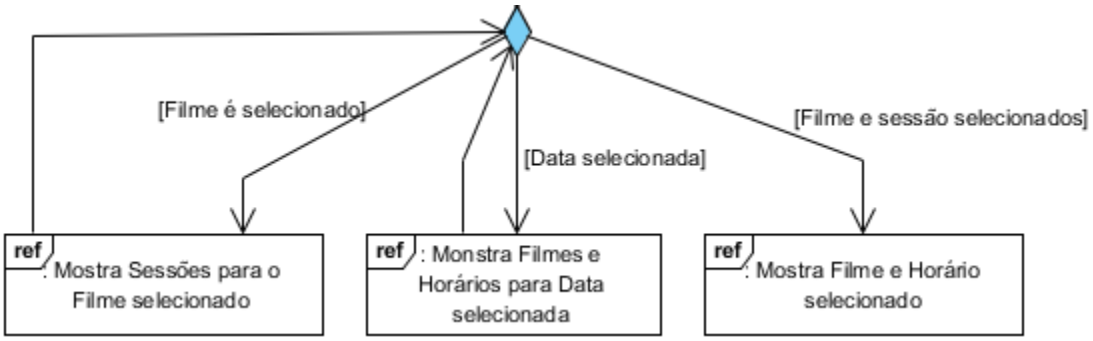
1.4 Nós de decisão e junção

O próximo passo para selecionar uma sessão de cinema é decidir entre as diversas opções que são oferecidas. O usuário pode:

- Selecionar um ou mais filmes para ver os horários disponíveis para este(s) filme(s).
- Selecionar uma data e ver os filmes e horários disponíveis para esta data.
- Selecionar um filme apresentado
- Cancelar a compra de bilhetes.

O primeiro nó de decisão precisa possibilitar uma das quatro opções citadas.

A figura a seguir apresenta o ponto de decisão e as sequências de interação de cada escolha:



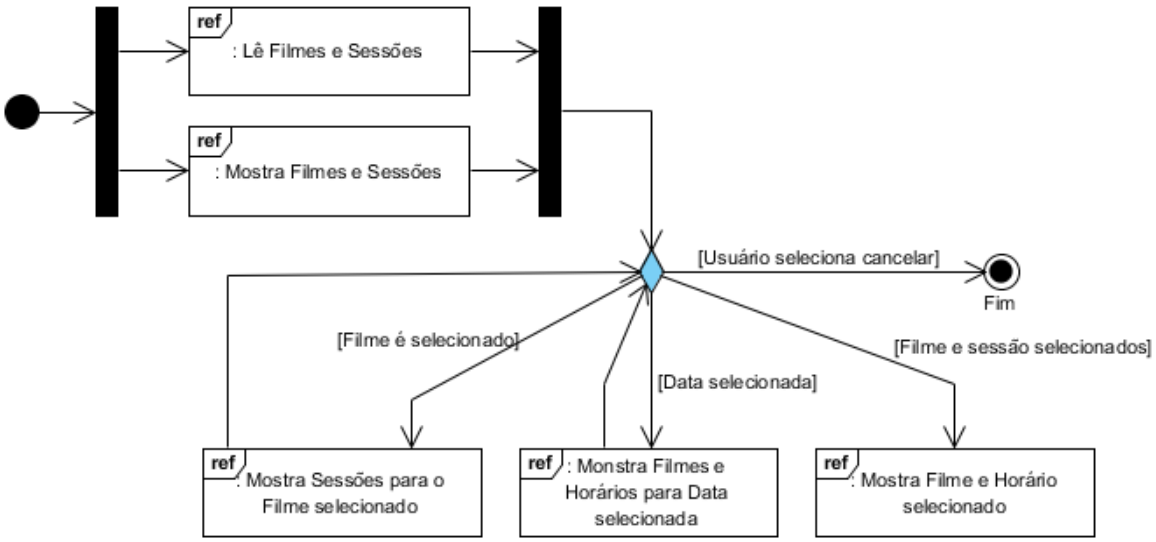
Notação de uma decisão e uma mesclagem.

06

1.5 Nós inicial e final

O ponto inicial é um nó de início. Assim como o ponto final é um nó de encerramento.

A imagem abaixo complementa os exemplos anteriores com os nós de início e fim.



Exemplo de diagrama completo

07

2 - Diagrama de tempo

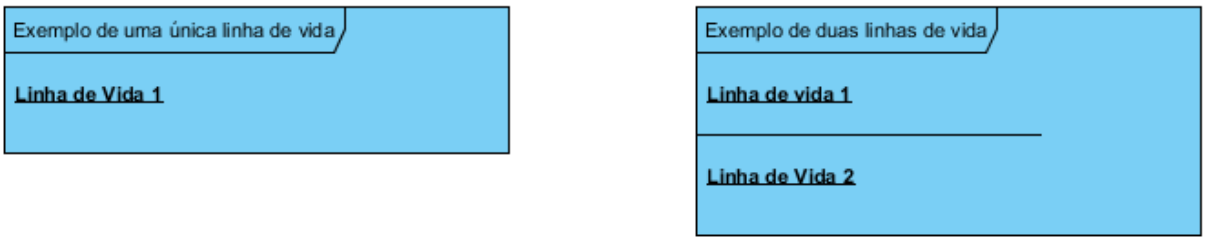
Um diagrama de tempo é um diagrama de interação de propósito especial que incide sobre os eventos ao longo da vida de um objeto.

É uma mistura de máquina de estado e diagrama de interação, com tempo de duração e restrições. O diagrama tempo utiliza uma linha estado e uma linha de tempo geral.

2.1 Modelando uma linha de vida

Uma linha de vida em um diagrama de tempo formam um espaço retangular dentro da área de conteúdo de um quadro. Ele é geralmente alinhado horizontalmente para ler a partir da esquerda para a direita. Várias linhas de vida podem ser empilhadas dentro do mesmo quadro para modelar a interação entre eles.

A figura abaixo mostra uma única linha de vida dentro de um quadro à esquerda, e duas linhas de vida dentro de um quadro à direita.



Exemplo de modelagem de linhas de vida em um diagrama de tempo

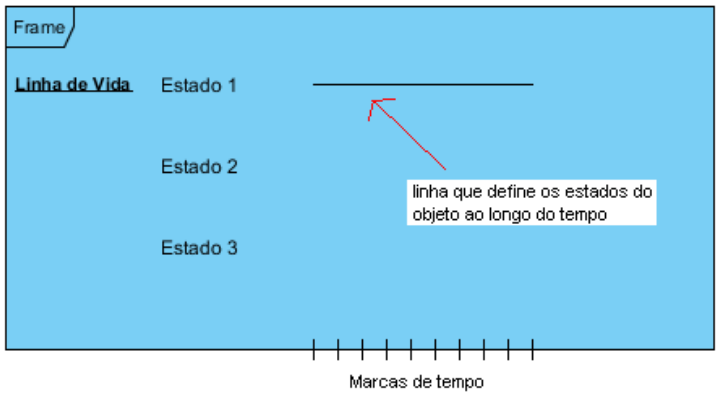
08

2.2 Modelando a linha de estado e a linha de tempo (marcas de tempo)

Uma linha de estados ou de uma condição representa o conjunto de estados e tempo válidos. Os estados são empilhados na margem esquerda da linha de vida de cima para baixo.

No exemplo abaixo, os estados são: **estado 1**, **estado 2**, e **estado 3**. A linha de tempo marca o tempo usando entalhes ao longo da parte inferior da região da linha de estados (semelhante às marcas de minutos de um relógio de ponteiros, porém horizontalmente). O incremento de tempo é específico para a situação modelada, e pode ser definido em qualquer unidade de tempo.

Veja a imagem abaixo:



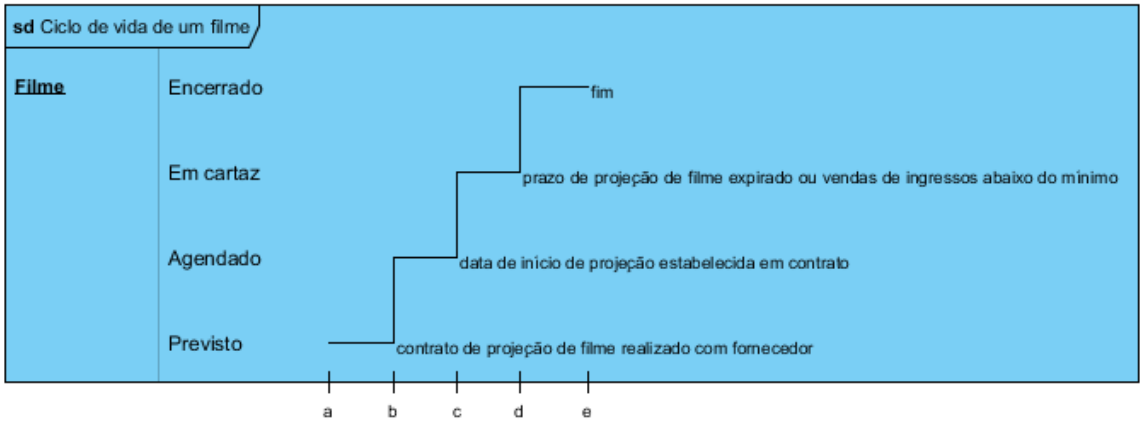
Modelando uma linha de vida, estados e marcas de tempo.

09

As alterações de um estado para outro são representadas por uma **alteração no nível da linha**. Para o período de tempo em que o objeto está em um determinado estado, a linha do tempo corre paralela a esse estado. Uma alteração no estado aparece como uma mudança vertical a partir de um nível para o outro. A causa da mudança, da mesma forma que ocorre em uma máquina de estado, é o recebimento de uma **mensagem**, ou seja, um evento que causa uma mudança, uma condição dentro do sistema, ou mesmo apenas a passagem do tempo (no caso de uma regra de negócio).

O nome da mensagem é colocado no vértice em que a linha de estado muda de nível para significar que a mensagem é a razão para a mudança.

A título de exemplo, a figura a seguir representa quatro possíveis estados do ciclo de vida de um filme para um sistema de cinema e possui três nomes de mensagens (além da mensagem de fim):



Exemplo de regras de negócio que determinam mudanças de estado.

10

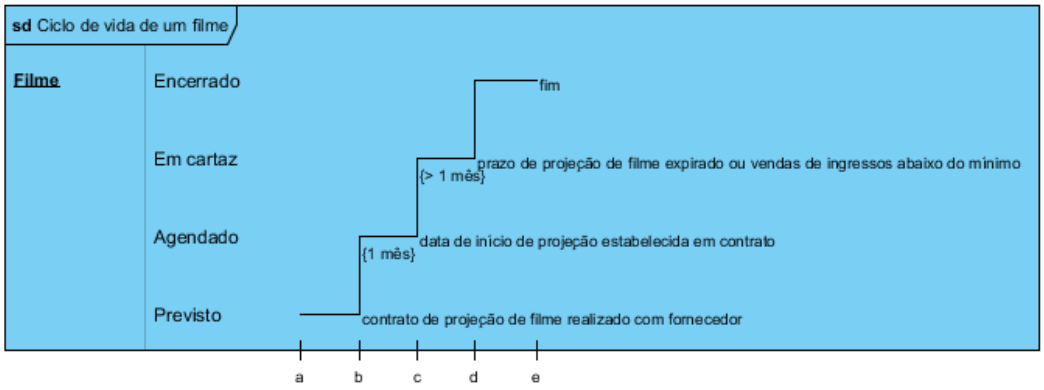
Assim como em um diagrama de sequência, o **diagrama de temporização pode documentar tempo de duração e restrições**.

O próximo exemplo acrescenta duas restrições de tempo na programação dos filmes:

1- A primeira indica que o tempo entre a assinatura do contrato e a data de agendamento para que o filme entre em cartaz é de 1 mês.

2- A segunda dita que o tempo mínimo que o filme deve ficar em cartaz é de 1 mês.

Note que o prazo máximo é ditado por outra regra de negócio que determina que o prazo máximo será aquele acordado no contrato com o fornecedor do filme, ou menor, caso as vendas de ingressos fiquem inferior ao mínimo estipulado. Tanto a restrição de tempo quando as restrições de regras de negócio podem realizar as mudanças de estado previstas.

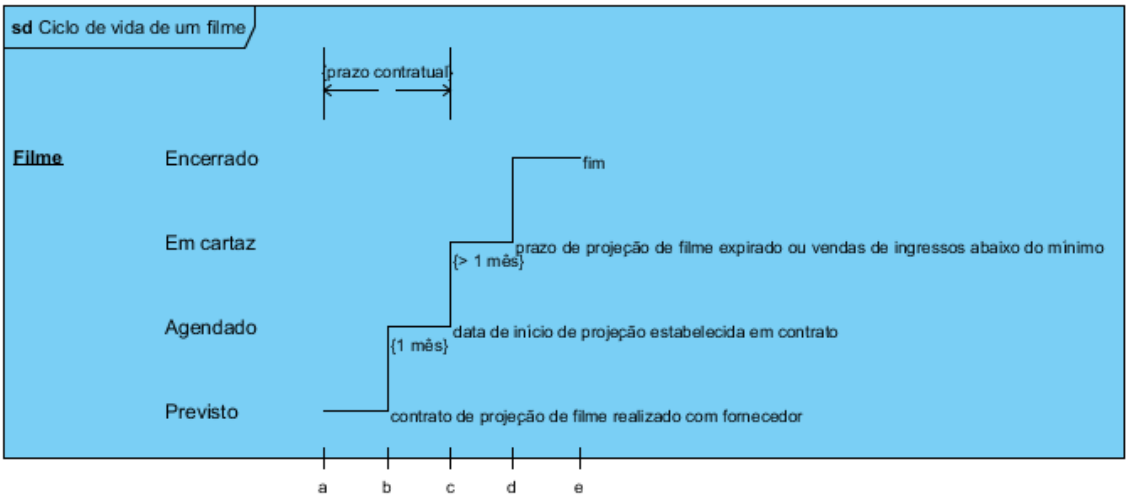


Exemplos de regras de tempo que determinam mudança de estados.

11

De modo semelhante, **uma restrição de duração pode ser adicionada para restringir o comprimento de tempo entre eventos**.

Na próxima figura, a restrição de duração indica que o tempo entre a assinatura do contrato de locação do filme e o agendamento para que ele entre em cartaz é de um prazo definido em contrato (“prazo contratual”).



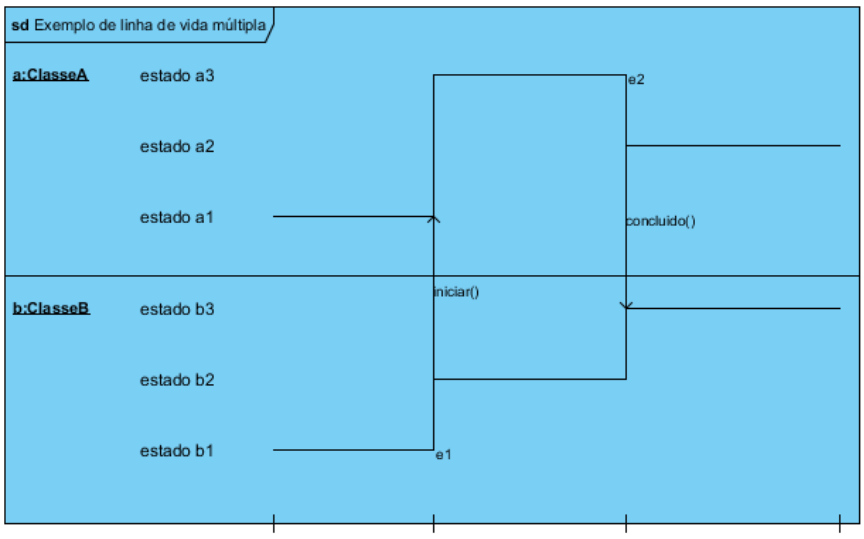
Exemplo de restrição de duração de um estado.

12

2.3 Modelando múltiplas linhas de vida

Interação implica em mais de um objeto. Para modelar vários objetos que participam da interação, simplesmente empilhe as linhas de vida, e, em seguida, mostre a interação como flechas que passam entre as linhas de vida.

Por exemplo, na abaixo, o evento **e1** causa uma mudança de estado em **b:ClasseB**. O objeto **b:ClsseB** responde enviando a mensagem de **iniciar()** para o objeto **a:ClasseA**. Posteriormente, o evento **e2** causa uma mudança no estado do objeto **a:ClasseA**, na qual ele termina por enviar a mensagem **concluido()** para **b:classeB**, fazendo com que b mude de estado novamente.

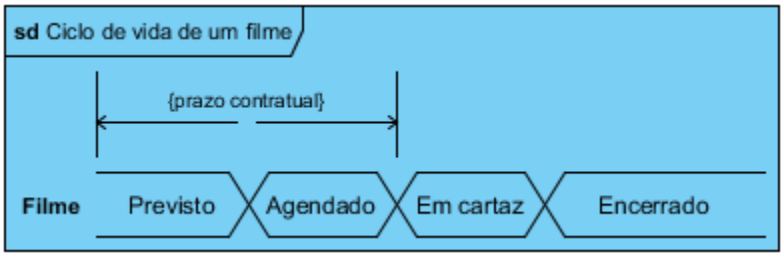


Exemplo de modelagem de interações entre linhas de vida diversas

13

2.4 Modelando uma linha de vida compacta

A linha de vida oferece uma apresentação alternativa. Alinham-se os estados horizontalmente e mostra-se a transição entre cada estado, conforme mostrado na figura abaixo:



A linha de estados também pode modelar os valores de atributos que definem cada estado em vez de usar nomes de estado.

14

RESUMO

Neste módulo aprendemos que:

a) A UML 2.0 oferece quatro diagramas de interação, o diagrama de sequência, diagrama de comunicação, diagrama de visão geral da interação, e um diagrama de tempo. Todos os quatro diagramas utilizam a notação de frame (retângulo) para delimitar uma interação. O uso de frames apoia a reutilização de interações como ocorrências de interação. Cada ocorrência de interação é uma referência a uma interação que é totalmente definida no seu próprio diagrama.

b) Uma interação é uma série de mensagens transmitidas entre objetos, a fim de realizar uma tarefa.

c) Uma linha de vida proporciona um meio para identificar a ordem de acontecimentos que afeta a condição e o comportamento de um objeto durante a execução de uma interação.

d) Uma mensagem é a forma básica de comunicação em uma interação em qualquer tipo de diagrama.

e) A ocorrência de um evento define o ponto numa interação em que uma mensagem ou sinal é enviado ou é recebido.

f) Uma restrição de duração é uma restrição sobre a duração do tempo que leva para completar uma tarefa.

g) Uma restrição de tempo define um tempo específico quando uma tarefa deve ser concluída.

h) A colaboração pode ser modelada utilizando uma combinação de uma estrutura composta e um diagrama de sequência.

i) O diagrama de Comunicação substituiu o diagrama de Colaboração da UML 1.4. Mensagens em um diagrama de Comunicação suportam iteração, execução paralela, e expressões de guarda.

j) Um diagrama de visão geral de interação combina o fluxo de controle a partir de um diagrama de atividades com interações e ocorrências de interação. Ele suporta todos os elementos do diagrama de atividades como, mecanismos de controle de fluxo, nós, garfos, junções, etc.

k) O diagrama de tempo modela as mudanças de estado em uma linha de estados para cada objeto em uma interação. Ele efetivamente combina uma máquina de estado e um diagrama de interação.

UNIDADE 4 – PASSOS FINAIS SOBRE UML
MÓDULO 3 – NÍVEL DE DETALHAMENTO DA MODELAGEM
01

1 - DIAGRAMAS UML

Olá, seja bem-vindo a mais uma parte do nosso estudo, quando trataremos basicamente sobre **o que** modelar e **quanto** modelar em um projeto.

Cada projeto de *software* pode requerer um ou mais tipos de diagramas para ser modelado. Talvez projetos simples exijam somente os casos de uso e o diagrama de classes, já um projeto crítico pode exigir a modelagem de todos os diagramas da UML. Cada diagrama também exigirá um determinado rigor do que deverá ser modelado (maior detalhamento ou menor detalhamento).

Por exemplo, em um diagrama de classes, definir as classes, métodos e atributos parece essencial em qualquer projeto, já definir valores padrão para seus atributos pode não ser tão necessário na maioria dos projetos (isso é diferente de dizer que tal informação é mais importante do que outra, apenas que determinada informação é necessária naquele contexto e outra não é).

A escolha do **que** modelar e **até onde** modelar deve ser uma decisão do projeto (da equipe, dos padrões adotados e da necessidade).

As instruções a seguir representam uma espécie de ordem de relevância quanto ao o que deve ser modelado em cada diagrama. Listamos para cada diagrama da UML, a informação mais significativa para a menos significativa.

02

1.1 Diagrama de objetos

Os itens mais importantes para se modelar em um diagrama de objetos são:

- Retângulo para simbolizar o próprio objeto.
- Nome do objeto na parte superior do objeto.
- Referência a qual classe o objeto pertence.
- Nome dos atributos, tipos, valores padrão e multiplicidade.
- Nome dos métodos, parâmetros e tipos de retorno.
- Nome do estereótipo acima do nome do objeto.
- Links entre os objetos.
- Especialização dos links (herança, agregação, composição, etc).
- Nome dos links.
- Tipificação do objeto por meio de ícones que definem objetos de controle, interface ou

03

1.2 Diagrama de sequências

Os itens mais importantes para se modelar em um diagrama de sequências são:

- Retângulo para simbolizar os objetos.
- Linhas pontilhadas para representar a linha de vida dos objetos.
- Setas de mensagens entre objetos, numeradas sequencialmente.
- Nomenclatura das mensagens para identificar os métodos que invocam as mensagens.
- Parâmetros e retornos.
- Especialização das linhas para indicar mensagens síncronas, assíncronas e retornos.
- Restrições e regras de negócio entre chaves.
- Restrições de tempo.

1.3 Diagrama de comunicação

Os itens mais importantes para se modelar em um diagrama de comunicação são:

- Retângulo para simbolizar os objetos.
- Linhas para representar comunicação entre os objetos.
- Setas de mensagens entre objetos, numeradas sequencialmente.
- Nomenclatura das mensagens para identificar os métodos que invocam as mensagens.
- Parâmetros e retornos.
- Especialização das linhas para indicar mensagens síncronas, assíncronas e retornos.

04

1.4 Diagrama de visão geral de interação

Os itens mais importantes para se modelar em um diagrama de visão geral de interação são:

- Retângulo para simbolizar as interações e as ocorrências de interações.
- Linhas de vida (quando aplicável).
- Setas para representar comunicação entre os objetos.
- Garfos, junções e decisões para representar possibilidades de fluxos alternativos.
- Condições de guarda.
- Parâmetros.
- Nós de início e de fim.

1.5 Diagrama de tempo

Os itens mais importantes para se modelar em um diagrama de tempo são:

- Retângulo para simbolizar os frames (quadros do contexto do diagrama).
- Linhas de vida.
- Estados.
- Linha de estados.
- Linha de tempo.
- Mensagens e eventos.
- Restrições de tempo e duração.

05

1.6 Diagrama de máquina de estados

Os itens mais importantes para se modelar em um diagrama de máquina de estados são:

- Estados inicial e final.
- Estados.
- Setas para indicar transições, com nome das mensagens.
- Nós estáticos e dinâmicos para mudanças de estados.
- Parâmetros e restrições das transições.
- Decisões e condições de guarda.
- Ações.

1.7 Diagrama de caso de uso

Os itens mais importantes para se modelar em um diagrama de caso de uso são:

- Atores.
- Casos de uso.
- Links de comunicação.
- Especialização dos links (herança, agregação, composição, etc).
- Relacionamentos de extensão e inclusão.
- Descrição textual do caso de uso.

06

1.8 Diagrama de atividades

Os itens mais importantes para se modelar em um diagrama de atividades são:

- Nós de início e fim.
- Retângulos arredondados com os nomes das atividades.
- Setas para indicar o fluxo entre os elementos do diagrama.
- Garfos e junções.
- Decisões e mesclagens.
- Nome/regras das decisões.
- Condições de guarda.

1.9 Diagrama de componentes

Os itens mais importantes para se modelar em um diagrama de componentes são:

- Retângulo para definir os componentes.
- Interfaces requeridas e provedoras.
- Linhas para expressar fluxos de comunicação.
- Portas.
- Classes para especializar componentes.
- Links entre as classes.
- Especialização dos links (herança, agregação, composição etc.).
- Regras de multiplicidade.

07

1.10 Diagrama de implantação

Os itens mais importantes para se modelar em um diagrama de implantação são:

- Caixas para definir os componentes e pacotes.
- Linhas para expressar links de comunicação.
- Regras de multiplicidade.
- Especificação dos componentes.

1.11 Diagrama de pacotes

Os itens mais importantes para se modelar em um diagrama de componentes são:

- Retângulo para definir os pacotes.
- Classes que compõem os pacotes.
- Linhas para expressar associações e dependências.

08

2 - Outros diagramas que auxiliam a UML

A UML não modela todo o trabalho de uma equipe de desenvolvimento de sistema, há outros modelos muito relevantes e importantes que trabalham junto com a UML para prover toda a documentação do projeto de *software*. Iremos abordar brevemente de alguns dos diagramas mais importantes.

2.1 Análise textual

A análise textual é uma técnica que permite identificar os elementos de um projeto de *software*. Muitas vezes um projeto de um sistema de informação é iniciado por meio de entrevistas com os usuários. Essas entrevistas são comumente documentadas em histórias que podem se tornar casos de uso.

Ao analisar essas histórias descritas pelos usuários você pode identificar itens de informação candidatos a se tornarem casos de uso, classes, atores, atividades, ações, fluxos, mensagens, regras de negócio etc. É primordial para a modelagem correta que você saiba identificar e separar cada um desses elementos.

Algumas ferramentas de modelagem UML incorporam funcionalidades de análise textual. Basicamente funcionam assim:

A partir desse ponto você já poderá criar seu projeto de modelagem com uma série de informações candidatas a se tornarem elementos reais da modelagem.

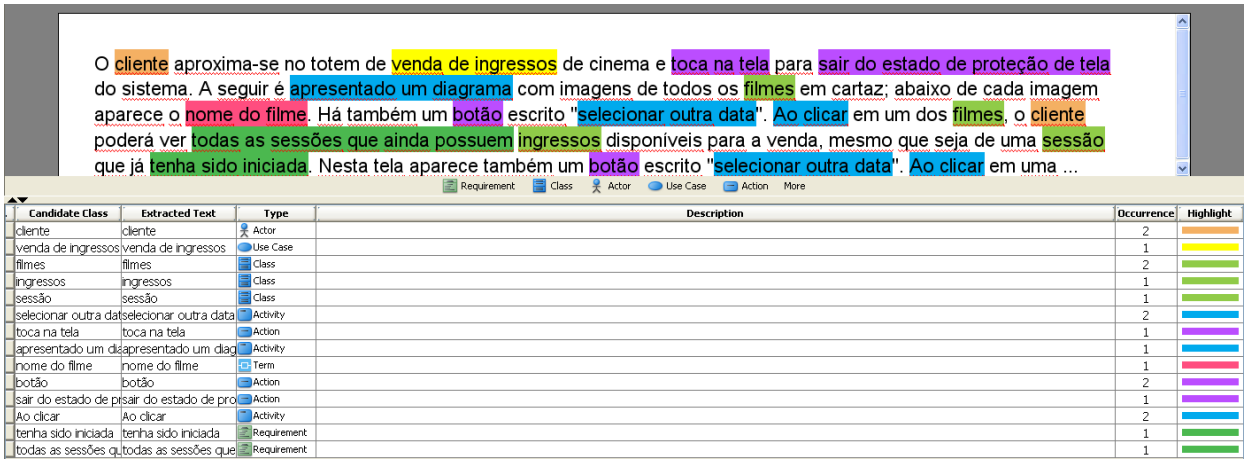
09

A imagem a seguir apresenta um trecho de uma análise textual do sistema de venda de ingressos do cinema. Observe:

a) as palavras e expressões destacadas;

b) o cadastro destas informações na tabela abaixo do texto;

c) a classificação dos itens identificados.



Candidate Class	Extracted Text	Type	Description	Occurrence	Highlight
cliente	cliente	Actor		2	
venda de ingressos	venda de ingressos	Use Case		1	
filmes	filmes	Class		2	
ingressos	ingressos	Class		1	
sessão	sessão	Class		1	
selecionar outra dat	selecionar outra data	Activity		2	
toca na tela	toca na tela	Action		1	
apresentado um dia	apresentado um diag	Activity		1	
nome do filme	nome do filme	Term		1	
botão	botão	Action		2	
sair do estado de pr	sair do estado de pro	Action		1	
Ao clicar	Ao clicar	Activity		2	
tenha sido iniciada	tenha sido iniciada	Requirement		1	
todas as sessões qu	todas as sessões que	Requirement		1	

Análise textual, cada cor refere-se a um tipo diferente de elemento.

Como você pode observar, da análise textual podemos identificar várias expressões candidatas a se tornarem elementos da modelagem UML, como atores, casos de uso, classes, atividades, ações, termos, requisitos, e outros.

A partir da análise textual normalmente as equipes de modelagem começam a modelar os casos de uso, e em seguida os diagramas de classes. Posteriormente, modelam-se os diagramas de sequências ou atividades e outros necessários.

10

2.2 Diagramas de macroprocessos

Os diagramas de macroprocessos modelam o negócio da empresa geralmente sobre o foco **fornecedor → empresa → cliente**, atendendo toda a cadeia produtiva da organização.

Independentemente do tipo de organização, em qualquer situação uma empresa recebe produtos e informações dos seus “fornecedores”, as processam e entregam para seus “clientes”.

Exemplos:

- Um mercado recebe mantimentos dos seus fornecedores e vende para seus clientes.
- Uma farmácia recebe fármacos e outros produtos de seus fornecedores e vende para seus clientes.
- Um banco recebe dinheiro de seus clientes de investimento (que representam fornecedores) e efetua serviços (pagamentos e empréstimos) para seus clientes.
- Uma entidade pública recebe insumos de outros órgãos públicos (fornecedores) e presta serviços para os cidadãos (clientes).

Os diagramas de macroprocessos proveem uma visão sequencial do sistema sobre o ponto de vista do negócio, ou seja, de como a empresa/organização funciona. Esse fluxo é denominado “Cadeia de Valor”. O diagrama que representa a cadeia de valor de uma organização auxilia a formar uma visão sistêmica sobre o sistema a ser modelado e orienta por onde começar a modelagem dos casos de uso.

11

Em níveis maiores de detalhamento, os diagramas de macroprocessos apresentam o fluxo do negócio da empresa. Identificar esses fluxos é base para modelagem de sistemas de informação que futuramente irão gerir o negócio da empresa.

Os **elementos básicos de um diagrama de macroprocessos** são:

- Ator

Pessoa física ou jurídica que interage com o macroprocesso. Geralmente modelado como um ícone de um boneco.

- Macroprocesso

Conjunto de ações que representam um negócio para a empresa. Modelado como um trapézio.

- Decisão

Controlam a separação do fluxo para caminhos distintos. Modelado como um losango.

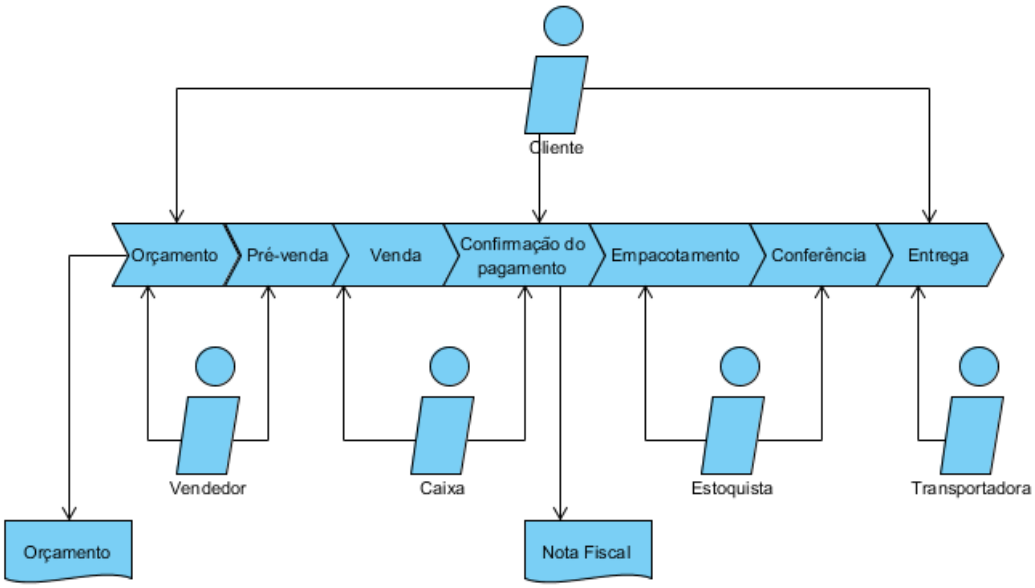
- Fluxo

Linhas que indicam entrada e saída de informação.

- Documento

Instrumento que representa um conjunto de informações que entra ou sai do macroprocesso.

O diagrama a seguir apresenta uma ideia hipotética de um macroprocesso de venda de produtos:



Macroprocesso de venda de produtos

12

2.3 Modelagem de banco de dados

A modelagem de banco de dados até o início do ano 2000 era umas das formas mais comuns de se iniciar a modelagem de um sistema. Sob a orientação da **arquitetura cliente-servidor**, os analistas pensavam o sistema em termos de "o que deve ser armazenado no banco de dados", as funcionalidades eram tratadas depois, e documentadas em texto puro, fluxogramas, modelagem de processos e outras técnicas.

Porém, com o advento das tecnologias de programação de última geração, o banco de dados tem-se tornado cada vez mais uma consequência da modelagem de classes, em que a parte de persistência é transformada em tabelas, campos e relacionamentos.

Por vezes, ainda, após a implantação da aplicação, administradores de dados analisam a performance do banco de dados e sugerem melhorias. Essas melhorias normalmente são recomendações de criação de índices, mudanças na estrutura de tabelas, inclusão ou exclusão de regras e até mesmo alterações na implementação física do próprio banco de dados como: replicação de dados, distribuição em pilhas de discos rígidos etc.

A parte lógica da análise dos modelos de dados normalmente é feita por **diagramas de entidade relacionamento**, em que são modeladas as seguintes informações:

- Tabelas;

- Campos;
- Tipo do campo;
- Relacionamento;
- Regras.

Tabelas
Conjunto de dados que representa o contexto de uma informação. Muitas vezes há uma relação muito clara entre as classes do sistema (que são persistidas) e as tabelas. Exemplo: tabela pessoa, tabela de vendas, tabela de endereços.

Campos
Tipo de informação de uma tabela. Representa cada uma das informações acerca do assunto modelado. Normalmente representa os atributos de uma classe. Exemplos: nome do cliente, CPF, data de nascimento.

Tipo do campo
Tipo lógico do campo. Assim como os atributos, qualifica o tipo de informação armazenada. Exemplos: inteiro, fracionário, booleano, string.

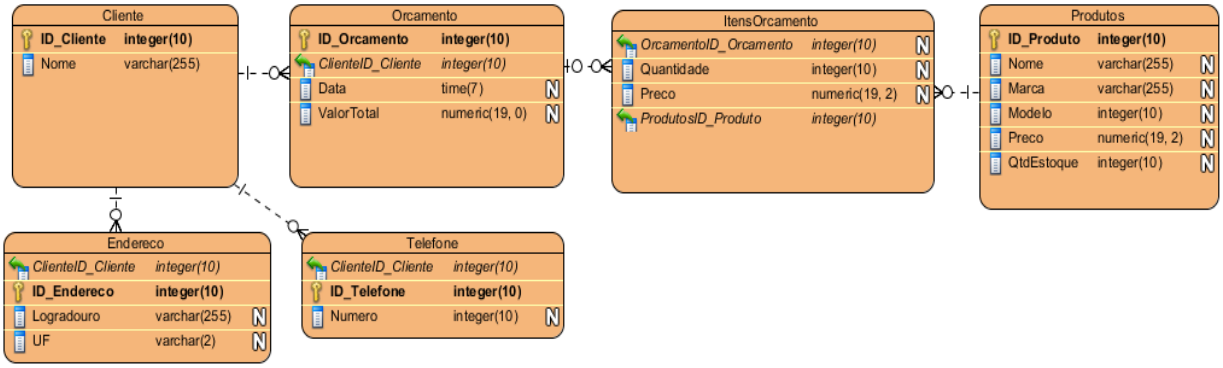
Relacionamento
Apresenta a relação entre duas tabelas, semelhante à associação entre duas classes. Exemplo: A tabela cliente relaciona-se com a tabela orçamento da seguinte forma: “um cliente possui zero ou muitos orçamentos associados”.

Regras
São pequenos trechos de código que acontecem antes e após as operações sobre as tabelas. Esses códigos muitas vezes são denominados de “triggers” (gatilhos). Permitem, por exemplo, validar operações de inserção ou exclusão de dados.

13

Muitas ferramentas de modelagem UML possuem funcionalidade de modelagem de banco de dados.

O diagrama a seguir apresenta um **exemplo simplificado de um diagrama entidade relacionamento** apresentando uma questão de sistema onde um cliente realiza um orçamento para compra de alguns produtos:



Você estudará com muitos detalhes sobre este assunto nas matérias sobre banco de dados.

14

Resumo

Neste módulo, aprendemos que:

a) Cada projeto de *software* exige um ou outro tipo de diagrama e o quanto de informação deve ser modelada em cada um.
b) Quem decide o que modelar e o quanto modelar é um conjunto formato pela equipe do projeto, necessidade do projeto e metodologia utilizada.

c) Há diagramas UML que são aplicáveis em praticamente todos os contextos (como casos de uso e diagrama de classes) e há também diagramas que raramente serão necessários e/ou aplicáveis (como diagrama de tempo e diagrama de implantação).

d) Cada diagrama UML possui informações essenciais (que devem estar presentes sempre, como, por exemplo, o nome das classes em um diagrama de classes) e informações complementares (que devem estar presentes somente quando necessário, como a multiplicidade em uma relação entre classes).

e) Existem outros diagramas que são muito comuns de virem associados aos diagramas da UML durante a modelagem de sistema. São eles:

    a. Análise textual – que provê uma ferramenta de identificação de informações candidatas a serem modeladas, como classes, casos de uso, atividades e outros.

    b. Macroprocessos – que provê uma visão sequencial do sistema sobre o ponto de vista do negócio, ou seja, de como a empresa/organização funciona. Este diagrama auxilia a formar uma visão sistêmica sobre o sistema a ser modelado e orientação de onde começar a modelagem dos casos de uso.

c. Modelo Entidade-relacionamento – que fornece uma visão do banco de dados em relação às tabelas, campos e relacionamentos, fazendo a transcrição da camada de persistência do modelo de classes em tabelas de banco de dados.

UNIDADE 4 – PASSOS FINAIS SOBRE UML
MÓDULO 4 – DEFININDO O SISTEMA

01

1 - TÉCNICA 5W2H

Neste módulo trataremos de como organizar suas ideias em um formato que direcione desde o início da modelagem até a conclusão, focando o desenvolvimento progressivo do trabalho.

Um trabalho de modelagem começa pela **definição das características e requisitos principais do sistema**. Na maioria das organizações isso é feito por meio de reuniões e entrevistas com usuários potenciais do sistema, e também por meio de leitura de documentação de outros sistemas ou sobre a teoria aplicada (exemplo: estudar sobre como funciona o trabalho de um contador para poder desenvolver melhor um sistema de contabilidade).

Uma técnica que ajuda muito nesse trabalho inicial é a chamada **5W2H**. Essa técnica representa o conjunto de sete perguntas que podem ser aplicadas durante o levantamento das informações do projeto.

As sete perguntas que dão nome à técnica são:

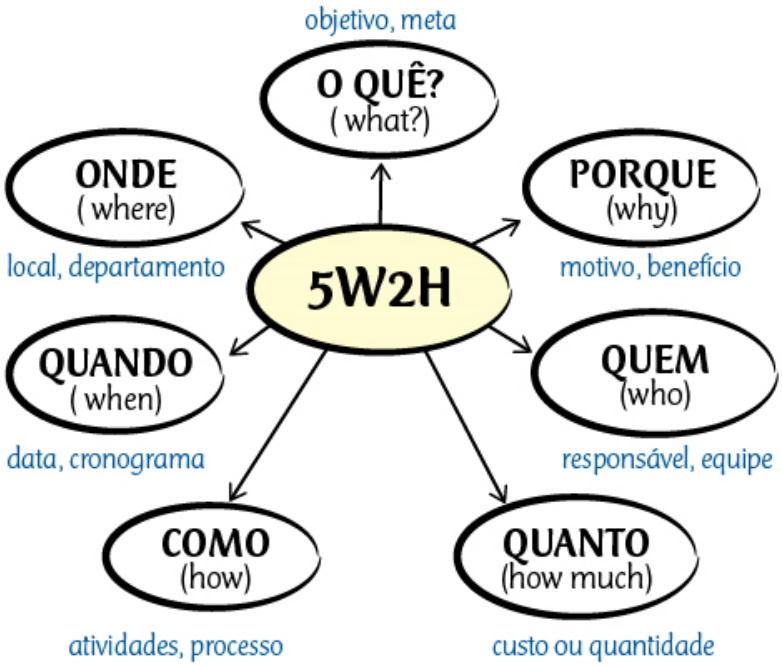
- **What** – o que;
- **Why** – por que;
- **When** – quando;
- **Who** – quem;
- **Where** – onde;
- **How** – como;
- **How Much** – quanto custa.

02

A técnica 5W2H será fundamental para que você faça um bom levantamento de requisitos, algumas perguntas farão mais sentido do que outras. Em alguns casos, algumas perguntam nem farão sentido ao contexto aplicado. Durante o projeto, pode ser que algumas perguntas sejam mais interessantes para

um determinado objetivo da modelagem, enquanto que em momentos futuros outras perguntas farão mais sentido.

Exemplo: talvez todas as perguntas sejam importantes na elaboração de casos de uso, perguntas do tipo "o que" são mais importantes para definição de classes, perguntas do tipo "como" mais importantes para relacionar classes, diagramas de sequência e de estado.



Lembre-se: uma das melhores formas de se trabalhar é ir evoluindo uma ideia em ciclos, de forma a cada vez mais coletar detalhes complementares (modelo em espiral). Depois fatie o escopo em pedaços que sejam preferencialmente independentes. Do contrário (se dependentes), organize-os de forma sequenciada.

What

Esta é a primeira pergunta a se fazer: o que é isso? O que estamos tratando aqui? A resposta desta pergunta irá te dar a noção do trabalho a ser feito, ajudando-o a definir e delimitar o escopo do projeto de *software*. Muitas vezes é necessário repetir a mesma pergunta várias vezes a fim de identificar todos os detalhes. Também é muito comum que usemos todas as perguntas da técnica ao mesmo tempo, ajudando a esclarecer melhor o trabalho.

Exemplo: O que é o sistema? É um sistema que irá gerar boletos bancários para pagamento de compras que os clientes efetuarem no sistema da empresa. Que outro sistema é esse? O sistema que proverá as informações para esse sistema de emissão de boletos chama-se "Sistema de Compras".

Why

Esta pergunta tem por objetivo fundamentar o seu projeto, trazendo informações que garantam a importância daquilo que será feito.. Quando aplicamos essa pergunta temos que ter em mente que a resposta dada justifique o projeto.

Exemplo: Por que faremos esse projeto? Porque o corpo de executivos da empresa decidiu que além dos pagamentos em dinheiro, cheque e cartão de crédito, os clientes agora poderão efetuar pagamentos por meio de boletos bancários.

When

Esta pergunta traz informações acerca de quando as coisas acontecem, ela funciona muito bem junto com a pergunta "como" (que está logo abaixo). Ajuda também a identificar prioridades, sequenciamento ou atividades que podem ocorrer em paralelo. Tudo isso formam regras de negócio.

Exemplo: Quando o boleto bancário deve ser gerado? Em dois momentos, ou quando o cliente conclui a compra ou no último dia do mês para os pagamentos que não foram efetivados. Nesse segundo caso, o sistema deve enviar os boletos por e-mail para os clientes (veja como várias regras de negócio aparecem).

Who

Essa pergunta traz informações dos usuários do sistema e outras entidades que de alguma forma participam do negócio.

Exemplo: Quem participa dessa rotina? Participam:

a) o Cliente que fez a compra e pode solicitar o boleto ou recebe-lo mesmo sem solicitação;

b) o Sistema de Compras que possui as informações dos clientes e compras efetuadas;

c) o novo sistema de Boleto, que deve ser operado automaticamente, sem a intervenção de usuários (observe que outras regras de negócio aparecem junto com as perguntas que se faz, no caso do exemplo, informações sobre o sistema funcionar de forma autônoma, baseado em eventos de calendário).

Where

A pergunta onde, assim como a pergunta como logo a seguir, possui dois focos:

a) um voltado à arquitetura da aplicação, pensando nos equipamentos que sustentam a aplicação,

onde ela ficará instalada, que tipos de equipamentos os usuários poderão usar para acessar o sistema (como tablets, smartphones, computadores pessoais, totens digitais etc.);

b) outro foco voltado ao layout do sistema e gestão da informação, onde no sistema as funcionalidades ficarão disponibilizadas: se será um sistema à parte, se é um módulo específico, se é uma sub-rotina de outra que já existe etc.

Exemplo: onde o sistema ficará hospedado? O sistema ficará hospedado no mesmo servidor de aplicação do sistema de compras. Onde fica esse servidor? Fica no datacenter na sede da empresa, na cidade de São Paulo. Onde no sistema ficará as funcionalidades previstas? Ficará num sistema totalmente à parte, um novo sistema.

How

Pergunta importantíssima, muitas vezes feita junto com a pergunta "o que", possui vários focos distintos:

a) entender o fluxo básico do sistema e seus fluxos alternativos;

b) definir toda a arquitetura de TI envolvida no sistema, desde o hardware básico, contemplando linguagem de desenvolvimento e aplicativos;

c) definir a arquitetura e processo de desenvolvimento, incluindo processo de desenvolvimento de *software*, metodologia de trabalho e de projeto, forma de validação do escopo e entregas etc.

Exemplo: Como funcionará a lógica do sistema: o sistema possui dois fluxos básicos. O primeiro deve ser integrado ao sistema de compras, onde o cliente poderá optar por essa nova forma de pagamento, e assim, o sistema deverá emitir o boleto de pagamento para a compra efetuada; confirmado o pagamento, o sistema deve registrá-lo na compra, autorizando a entrega dos produtos comprados. No último dia do mês, o sistema deve identificar quais clientes possuem compras pendentes de pagamento, nesse caso, deverá automaticamente, gerar boletos de pagamento nos valores das compras e enviá-los para os clientes. Ao efetivar o pagamento, o sistema deve registrá-lo no sistema de compras (veja que aqui surge várias complicações para serem tratadas com mais perguntas: a parte que permite o pagamento via boleto no sistema de compras precisa ser programado no sistema de compras, e não no sistema de boletos. Como o sistema de boletos saberá que um boleto foi pago? Será feita integração com o sistema bancário? Como isso será feito?). Como será a arquitetura de hardware do sistema: O sistema rodará em servidores Linux/Apache/Postgree, na linguagem PHP, assim como os demais sistemas da organização. Como será a arquitetura de desenvolvimento? A equipe utilizará o padrão da empresa que é baseado em Scrum, o projeto será definido para entregas pontuais mensais. A arquitetura de desenvolvimento envolve um ambiente de programação, outro de homologação e outro de produção, todos idênticos, a linguagem de programação será a PHP.

How Much

Esta pergunta foca em aspectos orçamentários e financeiros, de como será estimado o custo do

projeto, como ele será pago, como a equipe de desenvolvimento será contratada e paga, como demais recursos computacionais necessários serão adquiridos.

Exemplo: Qual é o orçamento previsto para o projeto? R$ 50.000,00. Como esse custo está estimado? Todo para pagamento de salários de analistas, não será necessário adquirir novos equipamentos.

03

2 - INICIANDO COM O FOCO NO TODO

Um projeto deve começar com o olhar sistêmico completo: observe tudo o que faz parte, mesmo que no começo você ainda não saiba dos detalhes.

Há muitos anos as aplicações rodavam apenas no computador do usuário. Atualmente, a maioria dos sistemas depende de uma arquitetura complexa, que permeia vários computadores. Veja, por exemplo, o sistema de compras de produtos da empresa (e-Commerce) que funciona por meio da Internet. Normalmente, utilizamos nosso navegador Web para acessar um sistema de compras, pesquisamos sobre a disponibilidade dos produtos, pesquisamos preços, selecionamos produtos para compra, selecionamos uma opção de entrega, e concluímos a compra definindo como iremos pagar e validando o pagamento.

A arquitetura desse negócio envolve:

- nosso computador pessoal,
- a conectividade Internet,
- o sistema de compras da empresa,
- um sistema de gerenciamento de banco de dados,
- um sistema de gerenciamento de aplicações,
- sistemas operacionais,
- antivírus,
- firewall, e outros sistemas de segurança,
- sistemas bancários (para autorização do cartão de crédito etc.).

Em face dessa complexidade do ambiente de TI, você precisa encontrar uma maneira de avançar com a modelagem correta do projeto e uma maneira de descrevê-lo usando UML.

04

Durante o projeto, uma das maneiras mais recomendadas de se trabalhar é primeiramente pensar no **sistema como um todo** (estipulando todo o escopo do projeto) e só depois você trabalhará nos detalhes.

Ao criar sistemas complexos, **defina as regras** de alto nível antes de você se concentrar no design de classes individuais. Ao tomar decisões sobre arquitetura, *hardware*, rede, segurança, interfaces de *software*, componentes e bancos de dados, você **limita o número de variáveis possíveis**. Ao olhar para todo o projeto em primeiro lugar, você se certifica de que **todos os grandes requisitos são tratados**, e atua para que apenas as classes necessárias sejam adicionadas ao projeto, sem inflá-lo com informações desnecessárias.
Depois que você conseguiu organizar e delimitar o seu projeto, agora é possível concentrar-se na concepção de suas classes.

O número de arquiteturas potenciais para um projeto de um sistema de informação qualquer (um sistema de informação comum) é quase infinito. Basta pensar nas muitas tecnologias possíveis, configurações de rede, plataformas de *hardware*, definições de classe, linguagens de programação, técnicas de desenvolvimento, protocolos de comunicações remotas, e técnicas de banco de dados que você pode usar.

O processo para projetar a imagem sistêmica do projeto de *software* envolve as etapas e considerações que veremos a seguir.

05

2.1 Considere as prioridades do projeto.

É claro que você quer construir um sistema que atenda às necessidades de seus usuários. Mas, há outros fatores concorrentes que você deve considerar em seu projeto, como os seguintes:

- Os **requisitos funcionais**;
- **Flexibilidade**;
- **Considere arquiteturas modernas**;
- **Performance**;
- **Custo**;
- **Prazo**.

Cada uma dessas prioridades afeta o sistema como um todo. Nem sempre você poderá optar pela melhor modelagem possível, às vezes você terá que trabalhar naquela que é a mais econômica, outra vez será a mais segura, outra vez a que suporta a maior carga.

A definição dessas variáveis deve acontecer logo no início do projeto, antes mesmo de se iniciar a modelagem. Mudanças nas prioridades afetam a arquitetura, fazendo com que você utilize uma ou outra alternativa.

requisitos funcionais

Cada caso de uso representa uma funcionalidade necessária para seu sistema. Alguns casos de uso podem ser mais importantes do que outros, e dado o seu orçamento e prazo disponível você pode ter que escolher quais casos de uso para implementar e quais deixar para outra versão do sistema. Talvez a primeira versão do sistema de geração de boletos implemente o básico: emitir boletos ao finalizar uma compra e emitir boletos para compras não pagas. A próxima versão vai lidar com recursos de integração bancária, por exemplo.

Flexibilidade

Você pode (e deve) projetar o sistema para ser modular. Dessa forma, quando os usuários mudarem de ideia (mudar uma regra de negócio), o seu design será fácil de mudar. No entanto, fazer sistemas flexíveis leva mais tempo para projetor e custará mais caro para construir. Você poderá implementar um sistema que gere qualquer tipo de boleto de pagamento, para qualquer tipo de banco, e qualquer empresa, tudo baseado em parâmetros, de forma flexível. Caso a empresa mude de banco, agência ou conta, será fácil mudar o sistema. Certamente, essa flexibilidade terá um custo maior, quase sempre a relação custo x benefício justifica valer a pena a flexibilidade.

Considere arquiteturas modernas

A maioria das arquiteturas de desenvolvimento modernas consideram a escalabilidade, a disponibilidade, confiabilidade, rastreabilidade, segurança, criptografia, etc. Uma arquitetura que seja capaz de gerar um boleto para um cliente deve ser a mesma capaz de gerar dezenas de boletos, de bancos diferentes de clientes e compras distintas sem nenhuma mudança significativa do projeto. “Fazer bem feito uma vez para servir para sempre”. Esse fundamento é essencial para evitar mudanças corretivas e adaptativas no futuro. Seu sistema deve funcionar em escala: “o que serve para um deve servir também para muitos”. Se o sistema deve funcionar em um ambiente 24/7, então você deve considerar uma arquitetura que prevê redundância a fim de que seu sistema suporte a grande maioria das possíveis falhas.

Performance

A arquitetura que você define está intimamente ligada com a performance do sistema. Se o sistema

não está rápido o suficiente, então os clientes do sistema podem deixar de usá-lo ou considerá-lo muito ruim. Se o sistema está rápido demais, pode ser que você tenha gasto muito mais do que o necessário. Para todo sistema a seguinte pergunta é sempre válida: “quão rápido é o desejável”. Você perceberá que a grande maioria dos sistemas esperam respostas entre 1 e 5 segundos como o desejável e 10 segundos como limite para momentos de pico. Acima disso podem ser considerados “sistemas lentos”.

Custo

A grande maioria dos projetos de *software* obedecem a um orçamento. A modelagem e a construção devem estar alinhadas para não custarem acima do orçamento estipulado, do contrário, o sistema pode ficar sem partes construídas ou dar prejuízo à organização.

Prazo

Assim como o orçamento, aquele que paga pelo sistema possui requisitos de quando o sistema estará pronto para uso. Às vezes os prazos possuem um pouco de flexibilidade, outras vezes não. Consente-se em descobrir o que é mais flexível para o projeto: o prazo, o custo ou o escopo. Se o projeto andar mal (ou for mal estimado), um dos três aspectos será sacrificado.

06

2.2 Reveja os sistemas atuais

Se o seu sistema irá substituir o sistema atual, **analise a arquitetura desse sistema**. Analise não só os problemas atuais, mas identifique também os pontos positivos. Observe a necessidade de troca de dados, isso às vezes tem um impacto profundo na arquitetura do sistema. Muitas vezes o melhor é manter a arquitetura já implantada, outras vezes você poderá optar por arquiteturas mais modernas.

2.3 Defina a arquitetura

Finalmente, de posse dos principais requisitos e da situação dos sistemas atuais, você já poderá **definir a arquitetura do seu sistema**. Às vezes existem algumas alternativas a serem validadas e analisadas, e miniprojetos de validação podem ser necessários para analisar a opção mais viável:

- Como será feita a persistência dos dados?
- Qual banco de dados utilizaremos?
- Qual linguagem de programação será utilizada?
- Windows ou Linux?
- Como os sistemas se comunicarão? Por meio do banco de dados? Troca de arquivos de texto? Troca de arquivos XML? Webservices?

Tudo isso fará parte da definição da arquitetura. Muitas vezes são necessários testes de validação da arquitetura, isso ajuda a garantir que a definição proposta será viável. Nessa fase é comum fazermos uma miniaplicação (chama comumente de “ping” ou “hello”) rodar para garantir que todos os componentes arquiteturais definidos são viáveis.

Políticas de segurança, *backup* e restauração devem fazer parte da arquitetura.

07

2.4 Fatie seu projeto

A grande maioria dos modelos de desenvolvimento de *software* trabalha no conceito da decomposição: **quebre algo complexo em partes menores, mais facilmente gerenciáveis**.

Fatiar o projeto é bom em vários sentidos: você pode fazer entregas mais rápidas para seu cliente, o cliente tem menos tempo para ficar pedindo por alterações, fica mais fácil identificar detalhes de funcionalidades que ainda não estão claras, testes e avaliações pilotos já podem ser iniciadas.

No caso do sistema de boletos você precisará dos seguintes componentes:

- Componente que gere boletos;
- Componente que gerencie mensalmente compras que não foram pagas;
- Componente que identifique os dados do cliente e do boleto;
- Componente que envie boletos por e-mail;
- Componente que receba informações bancárias e sinalize que os boletos foram pagos.

Quando todos esses componentes estiverem integrados entre si, seu sistema estará completo.

2.5 Defina a estratégia de desenvolvimento

A partir dos componentes do sistema, estime e defina o projeto de desenvolvimento de *software*. Não se esqueça das etapas de teste, treinamento, implantação e ajustes. Lembre-se um bom componente deve ser flexível, funcionar independentemente e não impactar outros componentes.

08

2.6 Gerencie os riscos e problemas

Quanto mais complexa uma arquitetura, maior a chance de erros e problemas. Ao modelar e construir sistemas, temos que ter em mente a maior parte possível de possibilidades de erros acontecerem. Quais alternativas podem acontecer? Como cancelar, como desfazer, como refazer, como continuar um processo que parou no meio do caminho?

Essas regras de negócio devem ser definidas principalmente nos casos de uso e nos diagramas de sequência, de forma que você conseguir gerenciar não só os eventos previsíveis, mas também os imprevisíveis.

2.7 Reveja e revalide tudo novamente

Com a evolução dos requisitos sobre o sistema é comum que tenhamos que fazer substituições. Às vezes pensamos em usar um determinado banco de dados e depois vemos que é necessário optar por outro; essas mudanças são comuns e a revisão da arquitetura fundamental para que as mudanças ocorram o mais cedo possível para o projeto.

Quanto mais cedo você mudar, menos impacto causará no seu projeto.

09

3 - Como a UML pode ajudar na definição do sistema

Desenhar um sistema envolve uma série de passos. Por sorte, a UML provê notações e diagramas que nos ajudarão nessa tarefa. Os principais conceitos são listados a seguir:

- **Decomposição do sistema**

Diagramas de pacotes e de componentes. A partir do escopo completo do sistema, decomponha-o em partes menores, como subsistemas. Mostre a relação de dependência entre esses componentes.

- **Interfaces**

Diagramas de classes. Explore e descreva as obrigações (funcionalidades e informações) de cada subsistema. Trate cada um como se fosse uma classe, descrevendo suas operações e informações.

- ***Hardware***

Diagrama de implantação. Descreva todo o hardware necessário para seu sistema funcionar. Mostre como as partes do hardware estão interligadas, ilustrando o caminho de comunicação entre essas partes.

- **Componentes**

Diagrama de componentes. Mostre quais partes do seu sistema são realmente reutilizáveis. Mostre como elas interagem com outros componentes e como podem ser substituídas por outros componentes.

- **Implantação**

Diagrama de implantação. Indica como os componentes e subsistemas são conectados como artefatos físicos. Inclui onde esses componentes estão instalados.

10

4 - Construindo peças lógicas

Já falamos que o principal passo para construir um sistema é a "decomposição do sistema". Nesta etapa, você precisará identificar as "peças lógicas" do seu sistema, a partir da visão sistêmica que você já construiu (uma visão completa do que o sistema deve fazer).

Essas peças lógicas devem ser definidas de tal modo que sejam completas e independentes nas tarefas que elas fazem, sendo assim possível criar um conjunto de peças que possam ser substituídas por outras peças equivalentes e/ou reutilizadas por outros sistemas.

Leia aqui uma analogia em que seu sistema seja "estruturar uma cozinha completa".

Veja essa outra análise: os componentes de um sistema NÃO são os módulos do sistema: cadastro tal, relatório tal etc.; os componentes do sistema são mais parecidos com isso: interface de usuário, gerador de relatórios, persistência de dados, componentes de segurança etc.

A ideia é que você tenha uma série de componentes prontos que, ao juntá-los sobre uma lógica específica, você criará o seu sistema.

É como um jogo de Lego, onde ao juntar as pecinhas plásticas de uma determinada forma, você cria um animal, uma casa ou um carrinho.

Leia aqui

Suponha uma analogia em que seu sistema seja “estruturar uma cozinha completa”. Você pode pensar em peças independentes como o fogão, a geladeira, os armários, a batedeira, o liquidificador, etc. Cada componente desses é completo e independente; se necessário, podemos pegar a geladeira e trocá-la por uma mais moderna, podemos pegar o fogão e utilizá-lo em outro lugar. Tudo é flexível e mutável. Essa é a ideia por trás dos componentes de um sistema.

11

Após definir a visão do projeto, você criará **diagramas de componentes** para ilustrar as peças necessárias para o seu sistema, posteriormente, para criar os artefatos reais, você utilizará alguma linguagem de programação criando-os do zero ou utilizando componentes já existentes.

Veja esse exemplo: suponha que você precise de um sistema que:

- armazene receitas culinárias,
- inclua imagens de pratos elaborados, e
- gere relatórios em formato PDF,
- tudo isso deve ser armazenado em um banco de dados MySQL e rodar em Java.

Certamente, se você pesquisar no Google, você irá encontrar componentes prontos que manipulam imagens e que geram relatórios em PDF. De forma nativa, o Java já possui componentes que permitem ler e gravar dados no banco de dados do MySQL. Dessa forma, você poderá pegar esses componentes prontos e trazê-los para o seu sistema, focando apenas no “como meu usuário quer que o sistema funcione para guardar, pesquisar e gerar relatórios das receitas culinárias”.

Esse pacote que contém a lógica do sistema é essencialmente o que você precisará codificá-lo no futuro.

12

4.1 Dicas para divisão de subsistemas

Use as sugestões a seguir para identificar subsistemas e separar os componentes do seu projeto:

- **Separe os componentes dos subsistemas em tipos**;
- **Use agregação**: se você possui uma agregação muito complexa no seu domínio, pense na possibilidade de criar subsistemas que contenham o agregador principal e suas partes.

- **Oriente seus subsistemas em razão dos casos de uso**: crie subsistemas que contenham todas as classes da sua aplicação que são necessárias para que cada caso de uso funcione perfeitamente. Você poderá combinar casos de uso semelhantes em um único subsistema.

- **Agrupe classes de domínio**: considere criar um subsistema que armazene todas as classes de domínio (classes de domínio referem-se ao domínio ou à linguagem do usuário). Essas classes de domínio aparecem em vários casos de uso da sua aplicação (e eles precisam persistir em um banco de dados). Juntar todas as classes de domínio em um único lugar facilita a padronização e provê uma forma comum para armazená-las em um banco de dados.

Nem todas as técnicas acima são compatíveis entre si. Por exemplo, a arquitetura em três camadas (apresentação, aplicação e dados) não é compatível com a arquitetura orientada a casos de uso (agrupadas por funcionalidades), entretanto, um sistema muito complexo pode ser criado pela combinação de várias arquiteturas.

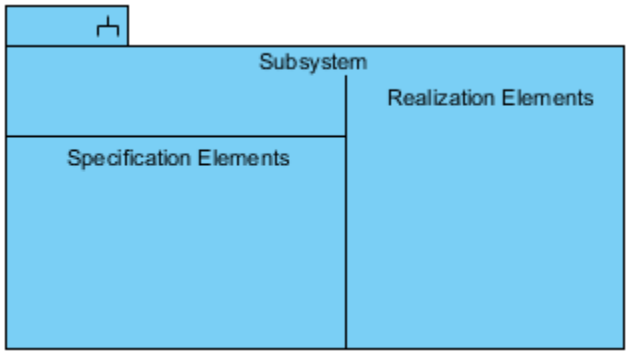
tipos
Separe os componentes dos subsistemas nos seguintes tipos:

- **Apresentação**: o subsistema de apresentação reúne tudo o que é pertinente à interação com o usuário: formulários, layout, navegação, menu de acesso, tela de login, interfaces disponíveis (se funcionará em um computador, um tablet e/ou um smartphone).
- **Aplicação**: o subsistema de apresentação representa a lógica do sistema e comunica-se com a apresentação e com os dados.
- **Dados**: o subsistema de dados é responsável pela persistência dos dados, guardando-os em um repositório (seja um arquivo de texto, XML, ou um SGBD como o Oracle ou SQL Server).

13

4.2 Empacotando suas classes

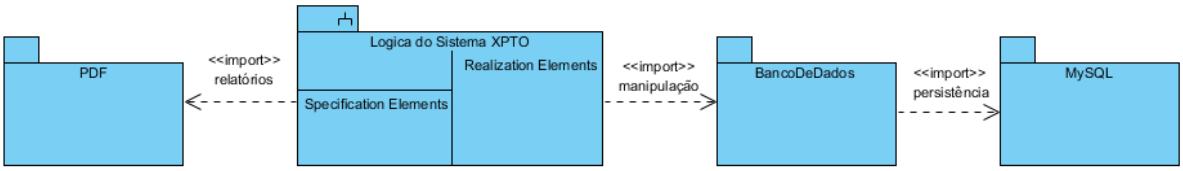
Você precisará criar **subsistemas** para agrupar suas classes em visão conceitual do seu projeto. A notação básica para um subsistema é um **retângulo** (ou o símbolo de uma pasta) com o nome do subsistema no topo e com a notação do estereótipo “subsytem”. Opcionalmente, você verá que algumas ferramentas incluem um pequeno ícone de um “garfo” no canto superior direito do retângulo. Esse ícone facilita a compreensão rápida pela equipe de desenvolvedores de que aquilo se trata de um subsistema que faz parte de um contexto maior.



Exemplo do layout de um subsistema

Subsistemas são um tipo de pacote. A ideia aqui é a mesma, da mesma forma que pacotes agrupam classes, subsistemas podem agrupar classes do seu projeto.

Cada subsistema é proprietário das classes que ele contém, não é possível haver dois subsistemas sendo proprietários de uma mesma classe. Você não precisa colocar uma mesma classe em mais de um subsistema, visto que um subsistema pode ser importado e reusado por outros subsistemas.

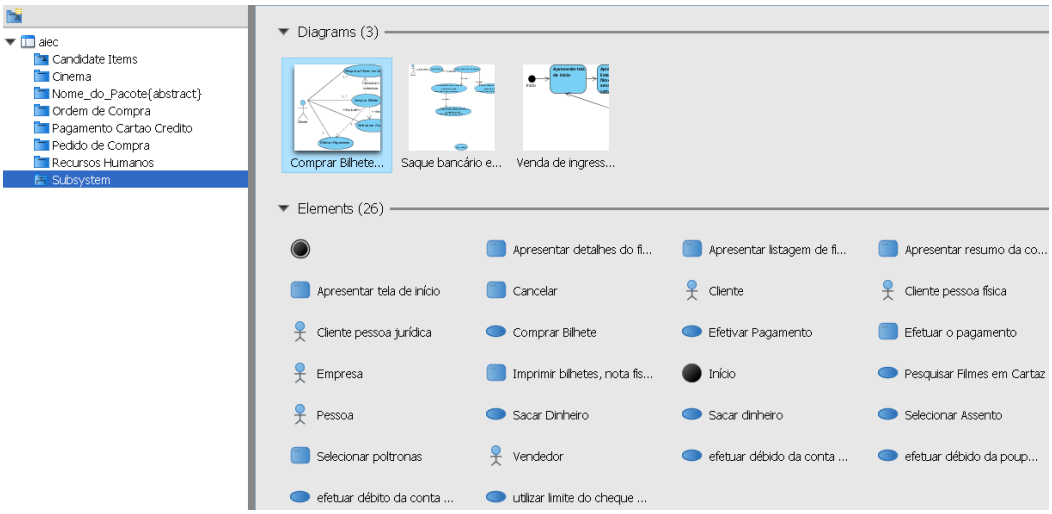


Exemplo de como um subsistema acessa outros pacotes.

14

Ao organizar seus diagramas em pacotes, o *software* que gerencia seus modelos organiza os diagramas e elementos de forma que fica fácil a localização desses itens.

Veja como exemplo a tela abaixo do *software* Visual Paradigm organizando os diagramas em um subsistema:



Exemplo do *software* Visual Paradigm, como ele organiza um Subsistema

No exemplo acima podemos ver que o subsistema criado possui três diagramas, e que esses diagramas possuem todos os elementos citados na imagem.

15

5 - Trabalhando com componentes

Já aprendemos que um sistema deve ser organizado em subsistemas.

Um subsistema isolado, autônomo e modular (em relação a um sistema maior) é chamado de **componente**. Na UML, componentes são como peças intercambiáveis – você pode substituir um componente por outro.

Na vida real, substituímos componentes quando procuramos otimizar o *software*, agregando mais funcionalidades, aumentando performance, aumentando segurança, tornando-o mais robusto etc. A maior vantagem de utilizar componentes é porque podemos trocá-los por outros sem necessitar mudar nada no sistema. Componentes tornam o sistema mais flexível, fácil de manter, escalável e reutilizável.

Componentes podem ser pequenos ou enormes.

Quando você modelar e construir componentes reutilizáveis, tenha atenção especial em definir bem os limites do componente. Esses limites são descritos pelas funcionalidades e interfaces de cada componente.

Os **critérios** mais importantes para a criação de componentes reutilizáveis são:

- Esconda o trabalho interno;

- Crie interfaces;
- Especifique as interfaces necessárias.

Esconda o trabalho interno

O trabalho interno do componente deve ser protegido de acesso externo. Nenhuma dependência deve haver entre a parte interna do componente e qualquer outro objeto, ele deve sempre ser capaz de funcionar sozinho, apenas pelos seus métodos públicos. Ou seja, encapsule o trabalho interno.

Crie interfaces

Uma interface define quais operações podem ser realizadas em um componente. Objetos fora do componente utilizam as interfaces para se comunicar.

Especifique as interfaces necessárias

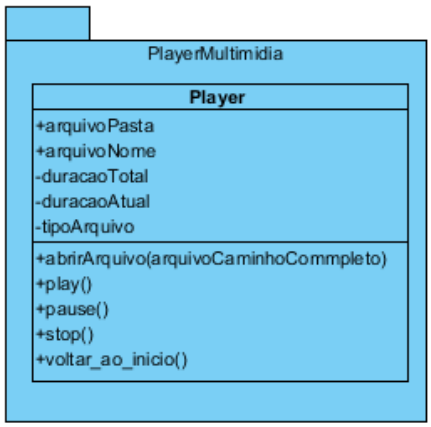
Algumas vezes, seus componentes precisarão acessar objetos fora do componente. Para isso, você precisará consultar as interfaces disponíveis nos outros componentes. Durante a modelagem, certifique-se de que haverá um local onde todas as interfaces possam ser consultadas.

16

5.1 Modelando interfaces

Já que um componente é acessado por meio de suas interfaces, precisamos definir uma forma de modelar essas interfaces. A UML estabelece um diagrama que contenha um componente e dentro dele uma classe para detalhar a respectiva interface.

Veja no exemplo abaixo como seria um componente hipotético de tocador de arquivos multimídia:



Pacote de tocador multimídia, expondo suas interfaces

Observe que não vemos como ele faz para tocar os arquivos, nem como ele faz para distinguir um arquivo de vídeo de um arquivo de música, o que importa é que temos acesso às operações que precisamos para tocar, pausar e parar a execução do player.

17

RESUMO

Neste módulo, aprendemos que:

a) A técnica 5W2H é interessantíssima para facilitar a coleta de informações relativas à modelagem de sistemas.

b) A técnica 5w2H é aplicada por meio de entrevistas a todos os interessados pela aplicação: usuários finais, usuários potenciais, gestores, etc.

c) A técnica 5H2W é composta pelos seguintes elementos:

   a. What – o que – define o que é o conceito.

   b. Why – por que – apresenta uma justificativa pela qual o item deve pertencer ao projeto.

   c. When – quando – apresenta eventos temporais que implicam em requisitos para o sistema.

   d. Who – quem – identifica os usuários e sistemas que interagem com o projeto.

   e. Where – onde – apresenta locais físicos e interfaces onde o sistema deve funcionar.

   f. How – como – identifica como o sistema funciona, seus fluxos e todas as características pertinentes ao relacionamento dos componentes.

   g. How much – quanto custa – estimativas de custo do projeto.

d) Todo projeto deve iniciar com uma visão sistêmica que defina todo o seu escopo, depois devemos fatiar o sistema em partes menores a fim de se obter os detalhes de cada parte.

e) Cada sistema prioriza uma ou mais questões em particular, identificá-las no começo do projeto é fundamental para que uma estratégia correta seja definida. Os critérios de priorização são: requisitos funcionais, flexibilidade, arquiteturas modernas, performance, custo e prazo.

f) A definição da arquitetura depende de vários fatores, os principais são: decomposição do sistema, interfaces, hardware, componentes e implantação.

g) A construção de um sistema deve ser preferencialmente orientada à arquitetura, que pode prever uma orientação por camadas ou por casos de uso, por exemplo.

h) As classes devem ser empacotadas de acordo com a estratégia adotada, formando subsistemas. Futuramente, os subsistemas serão organizados em pacotes, que serão codificados e implantados no ambiente físico.

i) Subsistemas podem incluir outros subsistemas. Um subsistema pode ser desenhado pelas suas interfaces utilizando diagramas de classes que expressam suas informações e funcionalidades.

j) *Software*s de modelagem utilizam subsistemas para organizar as informações, também criam uma espécie de índice que facilita ao modelador identificar os itens de modelagem.