Requisitos de Software

Visão Geral

Professor: Osvaldo Kotaro Takai
E-mail: otakai@uol.com.br

A Importância dos Requisitos

- A engenharia de requisitos é um termo usado para descrever as atividades envolvidas na elucidação, documentação e manutenção de um conjunto de requisitos de um sistema de software.
- O objetivo é descobrir o que os stakeholders necessitam que o sistema faça por eles.
- Estudos mostram que a falha no gerenciamento de requisitos é a principal causa dos fracassos em projetos de software.
- Como o sistema é baseado em requisitos, a engenharia de requisitos é crítico para o sucesso de projetos de desenvolvimento de software.

Definindo Requisitos

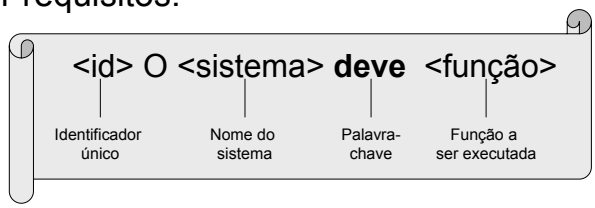
- Podemos definir um requisito como “uma especificação que deve ser implementada”.
- Existem dois tipos de requisitos:
  - Funcional: o que o sistema deve fazer.
  - Não-funcional: uma propriedade específica ou restrição sobre o sistema.
- Requisitos são (ou deveriam ser) a base de todo o sistema. São declarações **do que** o sistema deve fazer.
- Requisitos devem ser apenas uma declaração **do que** o sistema deve fazer e **não de como** deve fazer.
- Nós podemos especificar **o que** um sistema deve fazer e qual comportamento deve exibir sem necessariamente dizer qualquer coisa sobre **como** esta funcionalidade deve ser realizada.

Especificação de Requisitos de Software (SRS)

- Muitas empresas produzem um documento chamado SRS que contém a especificação do que o sistema de software deve fazer.
- Normalmente, esse documento é escrito em linguagem natural, mas ferramentas de engenharia de requisitos, tais como RequisitePro ou DOORS (www.telelogic.com) são usados com grandes vantagens.
- O SRS é o principal artefato de entrada para a fase de análise e projeto OO.

Requisitos Bem-Formados

- A UML não faz nenhuma recomendação sobre como escrever um documento SRS.
- A UML trata os requisitos usando use cases.
- No entanto, apenas use cases não são suficientes. Precisamos de SRS e ferramentas de gerenciamento.
- Recomendamos um formato muito simples para declarar requisitos:



Requisitos Funcionais e Não Funcionais

- Requisitos Funcionais
  - Um requisito funcional é uma declaração de o que o sistema deve fazer (função).
  - Por exemplo, para um caixa eletrônico (ATM), alguns requisitos funcionais podem ser:
    1. O sistema ATM deve verificar a validade do cartão.
    2. O sistema ATM deve validar a senha digitada pelo cliente.
    3. O sistema ATM não deve liberar mais que R$ 1.000,00 por cartão num período de 24 horas.

Requisitos Funcionais e Não Funcionais

- Requisitos Não-Funcionais
  - Um requisito não-funcional é uma restrição imposta ao sistema.
  - Por exemplo, para um caixa eletrônico (ATM), alguns requisitos não-funcionais podem ser:
    1. O sistema ATM deve ser escrito em C++.
    2. O sistema ATM deve se comunicar com o banco usando um algoritmo de criptografia de 256 bits.
    3. O sistema ATM deve validar a senha em três segundos ou menos.
  - Verifique que o requisito não-funcional especifica, ou restringe, como o sistema será implementado.

Requisitos de Software

Modelagem Use Case

Professor: Osvaldo Kotaro Takai
E-mail: otakai@uol.com.br

Modelagem Use Case

- A Modelagem Use Case (MUC) é uma forma de engenharia de requisitos.
- A MUC segue os seguintes passos:
  1. Definir a fronteira do sistema
  2. Encontrar os atores
  3. Encontrar os use cases
     - Especificar o use case
     - Criar cenários

Modelagem Use Case

- O resultado da MUC é o modelo Use Case.
- Existem quatro componentes nesse modelo:
  1. Atores: papéis interpretados por pessoas ou coisas que usam o sistema
  2. Use Cases: coisas que os atores podem fazer com o sistema
  3. Relacionamentos: associações importantes entre atores e usa cases
  4. Fronteira do sistema: Conjunto de atores e use cases do sistema

A Fronteira do Sistema

- A primeira coisa a ser feita quando pensamos em construir um sistema é decidir onde está a fronteira do sistema.
- Isto é, precisamos definir o que é parte do sistema (dentro da fronteira do sistema) e o que é externo (fora da fronteira do sistema).
- Isso parece óbvio, mas muitos projetos sofrem de sérios problemas devido à incerteza sobre a fronteira do sistema.

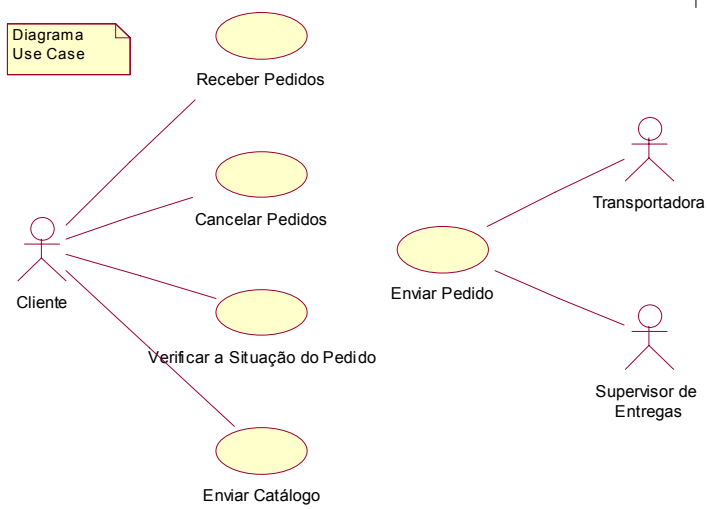
A Fronteira do Sistema

- A falta ou imprecisão na definição da fronteira do sistema pode causar enorme impacto nos requisitos funcionais (e algumas vezes, não funcionais). Requisitos a mais ou a menos é um grande problema.
- A fronteira do sistema é definida por quem usará o sistema (isto é, atores), e pelos benefícios específicos que o sistema irá fornecer a essas atores (isto é, use cases).

A Fronteira do Sistema

- A fronteira do sistema é definido pelo conjunto de atores e use cases apresentados num Diagrama Use Case.
- Exemplo de um Diagrama Usa Case.

A Fronteira do Sistema



O que são Atores?

- Um ator especifica alguém ou alguma coisa que interage diretamente com o sistema.
- O ator está sempre fora da fronteira do sistema.
- Um ator é representado em UML como “stick man”:

Cliente

O que são Atores?

- É importante ficar muito claro e cristalino de que Atores são externos ao sistema, mas que o sistema pode manter informações sobre esses Atores. Ou seja, uma representação interna de indivíduos que utilizam o sistema.

Identificando Atores

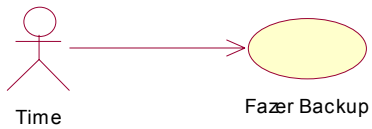
- As seguintes questões podem ajudar a identificar os atores do sistema:
  1. Quem ou o que utiliza o sistema?
  2. Quais são os papéis que eles interpretam durante a interação com o sistema?
  3. Quem instala o sistema?
  4. Quem inicia ou desativa o sistema?
  5. Quem mantém o sistema?
  6. Que outros sistemas interagem com o sistema?
  7. Quem obtém ou fornece informações para o sistema?
  8. Existe alguma coisa que acontece em tempos determinados?

Identificando Atores

- Pontos a serem lembrados durante a identificação de atores:
  1. Atores são externos ao sistema – portanto, eles estão fora do controle do sistema
  2. Atores interagem diretamente com o sistema
  3. Atores representam papéis que pessoas ou coisas interpretam em relação ao sistema, não são pessoas ou coisas específicas
  4. Uma pessoa ou coisa pode interpretar vários papéis. Por exemplo, num sistema de biblioteca você pode ser Usuário ou Bibliotecário (caso você trabalhe numa biblioteca)
  5. Cada ator deve ter um nome curto e que faça sentido na perspectiva de negócio
  6. Cada ator deve ter uma breve descrição que esclareça o qual é o seu papel na perspectiva de negócio

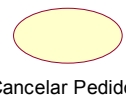
Time como um Ator

- Quando for necessário modelar coisas que acontecem em tempos específicos, mas que não são ativados por quaisquer ator, você pode introduzir um ator chamado Time.
- Um exemplo disso pode ser a funcionalidade de backup automático que é executado todas os dias às 19:00 horas.



O que são Use Cases?

- Um Use Case é alguma coisa que um Ator quer que o sistema faça.
- Use Cases são sempre iniciados por um Ator
- Use Cases são sempre escritos do ponto de vista de um Ator.
- Use Cases podem descrever funcionalidade de sistemas e subsistemas (parte de um sistema).
- Usa Cases podem ser utilizados para modelar processos de negócio.



Identificando Use Cases

- A melhor forma de identificar Use Cases é iniciar a partir da lista de Atores e considerar como cada Ator usa o sistema.
- Usando esta estratégia, pode-se obter uma lista de Use Cases candidatos.
- Cada Use Case deve ter num nome curto, iniciando com um verbo no infinitivo. Por exemplo: Solicitar, Fazer e Cancelar.
- Conforme os Use Cases vão sendo identificados, novos Atores podem ser descobertos. Isso é ótimo!

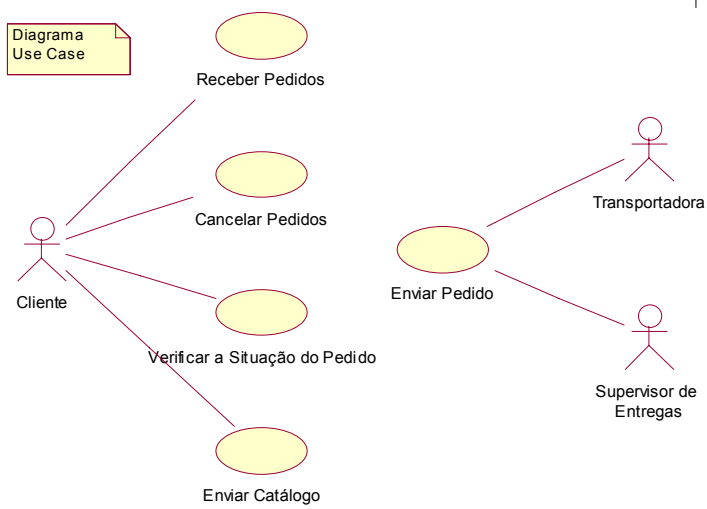
Identificando Use Cases

- É importante considerar cuidadosamente os requisitos funcionais do sistemas antes de finalizar o processo de descobrir Atores e Use Cases.
- A Modelagem Use Case é interativa e executada por um processo de refinamento sucessivo.
- As seguintes questões podem ajudar a identificar Use Cases:
  1. Quais funções um ator específico quer que o sistema tenha?
  2. O sistema armazena ou recupera informações? Se sim, quais atores ativam esse comportamento?
  3. Algum ator tem que ser avisado quando o sistema muda seu estado?
  4. Existe algum evento externo que afete o sistema? Quem informa o sistema sobre esse evento?

Diagrama Use Case

- No diagrama Use Case, nós representamos a fronteira do sistema.
- Ele mostra os Atores, que estão fora da fronteira do sistema, e Use Cases, que estão dentro da fronteira do sistema.
- O relacionamento entre um Ator e um Use Case é mostrado usando uma linha. Essa linha indica que o Ator e Use Case se comunicam de alguma forma.
- Exemplo de um Diagrama Use Case

Diagrama Use Case



O Glossário do Projeto

- O Glossário do Projeto pode ser um dos principais artefatos do projeto.
- Todo domínio de negócio tem sua linguagem própria e o principal propósito da engenharia e análise de requisitos é entender e capturar essa linguagem.
- O glossário define um dicionário dos principais termos de negócio.
- Ele deve ser conhecido por todos do projeto, inclusive pelos stakeholders.
- Além da definição dos principais termos, o Glossário do Projeto deve resolver sinônimos e homônimos.

O Glossário do Projeto

- **Sinônimos** são palavras diferentes com o mesmo significado.
  - Como analista OO você deve escolher uma das palavras (aquele que seja a mais utilizada) e não usar nenhum outro sinônimo.
  - Os outros sinônimos devem ser completamente excluídos do modelo para evitar que se criem, por exemplo, duas classes com nomes distintos, mas que fazem mais ou menos a mesma coisa.
  - Se fosse permitido o uso de sinônimos, a semântica das classes poderia, com o tempo, divergir.

O Glossário do Projeto

- **Homônimos** ocorrem quando a mesma palavra possui significados diferentes para pessoas diferentes.
  - Isso sempre traz problemas de comunicação e de desentendimentos.
  - A maneira de resolver esse problema é escolher apenas um significado para o termo.

O Glossário do Projeto

- No Glossário de Projeto, deve-se registrar os termos preferidos, bem como quaisquer de seus sinônimos.
- Talvez você tenha que convencer alguns stakeholders de que alguns termos são melhores do que outros.
- A UML não padroniza o formato para o Glossário do Projeto.
- Assim, recomendamos que use um formato semelhante ao de um dicionário, organizado em ordem lexicográfica dos termos do glossário.

Detalhar Use Cases

- Após criar o diagrama use case e identificar os principais atores e use cases, iniciamos a especificação de cada use case.
- Essa atividade é conhecida como “Detalhar Use Cases”
- O resultado desta atividade é a especificação de cada use case.
- Não existe um padrão na UML para uma especificação use case. No entanto é comum usar o formato apresentado no próximo slide:

Especificação Use Case

Use Case: Pagar Taxa	Nome do Use Case
ID: UC1	Identificador único
Atores: Time Governo	Os atores envolvidos com o use case
Pré-condições: 1. É final de trimestre	O estado do sistema antes que o use case possa começar
Fluxo de eventos: 1. O use case inicia quando for final de trimestre. 2. O sistema calcula o valor do importo devido ao Governo. 3. O sistema envia um pagamento eletrônico ao Governo.	Os passos reais do use case
Pós-condições: 1. O Governo recebe o valor correto do imposto.	O estado do sistema quando o use case finalizar

Especificação Use Case

- Cada use case tem um nome e uma especificação. A especificação consiste de:
  - **Pré-condições** – são coisas que devem ser verdadeiras antes que o use case possa ser executado – eles são restrições sobre o estado do sistema.
  - **Fluxo de eventos** – os passos do use case.
  - **Pós-condições** – coisas que devem ser verdadeiras ao final da execução do use case.
  - No exemplo, o Governo sempre recebe o imposto, de uma forma ou de outra; por isso é a pós-condição do use case.

Pré e Pós-condições

- Pré e Pós-condições são restrições
  - As pré-condições restringem o estado do sistema antes que o use case possa iniciar sua execução. Pense nelas como guardiãs que previnem que um ator inicie a execução do use case até que todas as condições tenham sido atendidas.
  - Pós-condições restringem o estado do sistema após a finalização da execução do use case.
- Uma outra maneira de pensar:
  - Pré-condições especificam o que deve ser verdadeiro antes que o use case possa ser executado
  - Pós-condições especificam o que deverá ser verdade após a execução do use case.

Pré e Pós-condições

- Pré e Pós-condições nos ajudam a projetar sistemas que funcionem corretamente.
- Pré e Pós-condições sempre devem ser declarações simples sobre o estado do sistema que podem ser testados como verdadeiro ou falso – eles são conhecidos como condições booleanas.

Fluxo de Eventos

- O fluxo de eventos lista os passos de um use case.
- Ele sempre começa com um ator fazendo alguma coisa para iniciar o use case.
- Uma boa maneira de iniciar um fluxo de eventos é:
  - O use case inicia quando o <ator><alguma ação>
- Lembre-se que o Time pode ser um ator, assim, o use case pode também ser iniciado com uma expressão de tempo no lugar do ator, como no exemplo do use case: Pagar Taxa.
- O fluxo de eventos consiste de uma seqüência de breves passos que são declarativos, numerados e ordenados no tempo.

Fluxo de Eventos

- Cada passo no fluxo do use case deve ser da forma:

  <número>O <alguma coisa><alguma ação>

  - Exemplo:
    1. O use case inicia quando o cliente seleciona a opção “fazer pedido”.
    2. O cliente entra com seu nome e endereço no formulário.
- Esses passos são bem formatados. Em ambos os casos, temos uma declaração simples de alguma coisa executando alguma ação.

Fluxo de Eventos

- O fluxo de eventos do use case pode também ser capturada como uma prosa. No entanto, nós não recomendamos isso, pois geralmente é muito impreciso. (Takai quer a prosa)
- Nós podemos mostrar alternativas num fluxo do use case ramificando ou listando os fragmentos do comportamento sob o cabeçalho de um fluxo alternativo do use case. Veremos isso depois.

Ramificação dentro do fluxo

- Com freqüência você precisa indicar que existem algumas possibilidades dentro do fluxo de eventos de um use case.
- Um boa maneira de fazer isso é usar o “Português estruturado”.
- Nós introduzimos um conjunto simples de palavras-chaves que você pode usar para expressar ramos, repetições e até fluxos alternativos.
- Vale a penas saber que alguns modeladores de use cases podem não gostar de ramos dentro de use cases.

Ramificação dentro do fluxo

- Eles dizem que, se existe um ramo então um novo use case deveria ter sido criado.
- Estritamente falando, esse argumento tem seu mérito – no entanto, nós tomamos uma posição mais pragmática: é desejável uma pequena quantidade de ramos pois isso reduz o número total de use cases e permite uma representação mais compacta dos requisitos.
- Você irá ver técnicas específicas para lidar com use cases realmente complicados, onde ramos podem ser totalmente inapropriados.

Ramificação dentro do fluxo

- Palavra-Chave **Se**
  - Usada para indicar um ramo no fluxo de eventos.

Use Case: Gerenciar Carrinho de Compras
ID: UC10
Atores: Cliente
Pré-condições: 1. O Carrinho de Compras está visível.
Fluxo de Eventos: 1. O use case inicia quando o Cliente seleciona um item no Carrinho de Compras. **2. Se** o Cliente selecionar "remover item" **então** 2.1. O sistema remove o item do Carrinho de Compras. **3. Se** o Cliente informar uma nova quantidade **então** 3.1. O sistema atualiza a quantidade do item no Carrinho de Compras.
Pós-condições: 1. O conteúdo do Carrinho de Compras foi atualizado.

Ramificação dentro do fluxo

- A figura anterior mostra um fluxo de eventos muito bem estruturado com 2 ramos.
- Cada ramo é prefixado com a palavra-chave **Se**, começa com uma simples expressão Booleana, tal como: "**Se** o usuário informar uma nova quantidade", que é verdadeira ou falsa.
- O texto identado sob a declaração **Se** é o que será executado se a expressão for verdadeira.
- O bloco da declaração **Se** é definido pela identação e numeração, dispensando qualquer declaração explícita de final de bloco.

Ramificação dentro do fluxo

- Fluxos Alternativos:
  - Algumas vezes não é possível usar o **Se**. Por exemplo: como usar a declaração **Se** para dizer que o Cliente pode sair da tela do Carrinho de Compras em qualquer momento?
  - Pode-se perceber que, se quisermos indicar essa situação usando a declaração **Se**, teríamos que fazê-lo em quase todas as linhas do fluxo de eventos. Nada prático!
  - É melhor expressar ramos que podem ocorrer em qualquer ponto do fluxo de eventos como um ou mais **fluxos alternativos**. Por exemplo:

Ramificação dentro do fluxo

Use Case: Exibir Carrinho de Compras
ID: UC11
Atores: Cliente
Pré-condições: 1. O Cliente efetuou login no sistema.
Fluxo de Eventos: 1. O use case inicia quando o Cliente seleciona “exibir Carrinho de Compras”. 2. **Se** não existir itens no Carrinho de Compras **então** 2.1. O sistema informa ao Cliente que ainda não existe itens no Carrinho de Compras. 2.2. O use case termina. 3. O sistema exibe uma lista com todos os itens do Carrinho de Compras do Cliente incluindo: ID, nome, quantidade e preço dos produtos.
Pós-condições:
Fluxo Alternativo 1: 1. Em qualquer momento, o Cliente pode sair da tela do Carrinho de Compras.
Pós-condições:
Fluxo Alternativo 2: 1. Em qualquer momento, o Cliente pode sair do sistema.
Pós-condições:

Ramificação dentro do fluxo

- Como pôde ser visto, acrescentou-se uma nova seção no final do use case, para cada fluxo alternativo. Essa seção contém:
  - O fluxo de eventos do fluxo alternativo. Normalmente, este fluxo de eventos deve ser simples, com poucos passos, e sempre deve começar com uma condição Booleana indicando quando esse fluxo deverá ser executado.
  - As pós-condições do fluxo de eventos alternativo.
- Não coloque as pós-condições dos fluxos alternativos como pós-condições do fluxo principal.

Repetição no Fluxo: Para

- Algumas vezes há a necessidade de repetir várias ações dentro do fluxo de eventos.
- Isso não ocorre com freqüência na modelagem use case, mas quando ocorrer, é útil ter um estratégia.
- Pode-se modelar repetições usando a palavra-chave **Para**. O formato é:

n. **Para** (expressão de iteração) **faça**
n.1. Alguma coisa.
n.2. Outra coisa.
n.3. ...
n+1.

Repetição no Fluxo: Para

Use Case: Encontrar Produtos
ID: UC12
Atores: Cliente
Pré-condições:
Fluxo de Eventos: 1. O Cliente seleciona a opção “encontrar produtos”. 2. O sistema solicita o critério de busca ao Cliente. 3. O Cliente informa o critério solicitado. 4. O sistema procura por produtos de acordo com o critério de busca informado pelo Cliente. **5. Se** o sistema encontrar produtos que satisfaçam o critério de busca **então** 5.1. **Para** cada produto encontrado **faça** 5.1.1. O sistema mostra uma pequena imagem do produto. 5.1.2. O sistema mostra um resumo dos detalhes do produto. 5.1.3. O sistema mostra o preço do produto. **6. Senão** 6.1. O sistema informa ao Cliente que nenhum produto pôde ser encontrado.
Pós-condições:
Fluxo Alternativo: 1. Em qualquer momento, o Cliente ir para uma página diferente.
Pós-condições:

Repetição no Fluxo: Enquanto

- A palavra-chave **Enquanto** é usada para modelar uma seqüência de ações no fluxo de eventos que é executado enquanto alguma condição Booleana for verdadeira.
- O formato é:

n. **Enquanto** (Condição booleana) **faça**
  n.1. Alguma coisa.
  n.2. Outra coisa.
  n.3. ...

n+1.

Repetição no Fluxo: Enquanto

- A palavra-chave **Enquanto** também é pouco utilizada.
- A seqüência de linhas identadas após a declaração **Enquanto** é repetida até que a condição booleana torne-se falsa.
- Exemplo:

Repetição no Fluxo: Enquanto

Use Case: Exibir Detalhes da Empresa
ID: UC13
Atores: Cliente
Pré-condições:
Fluxo de Eventos: 1. O use case inicia quando o Cliente seleciona a opção “exibir detalhes da empresa”. 2. O sistema exibe uma página web contendo os detalhes da empresa. 3. **Enquanto** o Cliente estiver navegando nos detalhes da empresa **faça** 3.1. O sistema toca alguma música de fundo. 3.2. O sistema exibe ofertas especiais num banner ad
Pós-condições:

Rastreando Requisitos

- Com um documento SRS e um conjunto de use cases, nós temos efetivamente duas bases de dados de requisitos funcionais.
- É muito importante relacionar essas duas bases para verificar se existe alguma no SRS não esteja coberto pelos use cases e vice-versa.
- Este é um dos aspetos de rastrear requisitos.
- Rastrear requisitos funcionais para use cases é complicado pelo fato de ser um relacionamento muitos-para-muitos entre eles.

Rastreando Requisitos

- Isso pode ser feito com facilidade criando uma matriz de rastreabilidade de requisitos:

		Use Cases			
		UC1	UC2	UC3	UC4
Requisitos	R1	χ			
	R2		χ	χ	
	R3			χ	
	R4				χ
	R5	χ			

Rastreando Requisitos

- A matriz de requisitos é uma ferramenta muito útil para verificar consistências.
- Se existir um requisito que não tenha sido mapeado para um use case, então existe ao menos um use case faltando.
- Por outro lado, se existir um use case que não possa ser mapeado para algum requisito, então os requisitos estão incompletos.

Use Cases Complexos

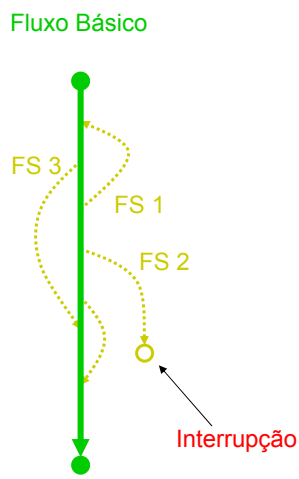
- Como regra geral, use cases devem ser mantidos o mais simples possível.
- No entanto, ocasionalmente, nos deparamos com um complexidade irredutível, ao ponto de termos que criar use cases complexos.
- Ao invés de tentar capturar essa complexidade com ramos e fluxos alternativos, é mais fácil, ou menos propenso a erros, modelar um fluxo básico e sua teia de ramificações usando cenários separados.

Use Cases Complexos

- Cenários:
  - Cenários são uma outra forma de ver um use case.
  - Um cenário é um caminho específico através do use case.
  - Durante a documentação de um use case, pode-se explorar vários caminhos específicos através do fluxo de eventos do use case, então cada um dos caminhos é um cenário.
  - A característica importante de cenários é que eles não se ramificam.
  - Assim, cada possível ramo no fluxo de eventos do use case gera, potencialmente, um cenário isolado.
  - Cada use case tem exatamente um cenário principal ou Fluxo Básico. Esse é o “caminho feliz” do use case.

Use Cases Complexos

- Tudo, no Fluxo Básico, ocorre como esperado e desejado, não existem erros, desvios, interrupções ou ramos.
- Um use case pode ter muitos cenários secundários, ou Fluxos Secundários.
- Fluxos Secundários podem capturar erros, ramos e interrupções no fluxo básico.



Use Cases Complexos

- Especificando o Fluxo Básico
  - Quando você usa a abordagem de cenários para documentar use cases, a especificação use case contém o Fluxo Básico e uma lista de nomes dos Fluxos Secundários numa seção apropriada.
  - Os Fluxos Secundários são normalmente documentados em separado e da mesma maneira que um use case é documentado.
  - A seguir um exemplo de um use case com seu Fluxo Básico:

Use Case: Fechar Pedido
ID: UC14
Atores: Cliente
Pré-condições:
Fluxo Básico: 1. O use case inicia quando o Cliente seleciona a opção “fechar pedido”. 2. O sistema exibe o pedido do Cliente. 3. O sistema solicita a identificação do Cliente. 4. O Cliente entra com uma identificação válida. 5. O sistema recupera e exibe os detalhes do Cliente. 6. O sistema solicita informações do cartão de crédito – nome escrito no cartão, número e data de expiração. 7. O Cliente informa os dados do cartão conforme solicitado. 8. O sistema solicita confirmação do pedido. 9. O Cliente confirma o pedido. 10. O sistema debita no cartão de crédito. 11. O sistema exibe a fatura.
Fluxos Secundários: Identificador do Cliente Inválido Detalhes do Cartão Inválido Limite Excedido do Cartão de Crédito Cartão de Crédito Expirado
Pós-condições:

Use Cases Complexos

- Especificando Fluxos Secundários
  - Você deve especificar os Fluxos Secundários da mesma forma que são especificados os use case.
  - Deixe sempre claro de como o cada Fluxo Secundário se inicia e assegure-se de que ele seja apenas uma caminho específico através do fluxo de eventos do use case sem nenhuma ramificação.
  - Cada Fluxo Secundário deve ser rastreável de volta para o seu use case.
  - O exemplo a seguir ilustra uma boa maneira de especificar Fluxos Secundários.

Use Cases Complexos

Use Case: Fechar Pedido **Fluxo Secundário: Identificador do Cliente Inválido**
ID: UC15
Atores: Cliente
Pré-condições:
Fluxo Secundário: 1. O use case inicia no passo 3 do use case Fechar Pedido, quando o Cliente informa um identificador de Cliente inválido. 2. **Enquanto** identificador do Cliente inválido e o número de tentativas for menor que três **faça** 2.1. O sistema solicita a identificação do Cliente 3. O sistema informa ao Cliente que a sua identificação não foi reconhecida.
Pós-condições:

Use Cases Complexos

- Encontrando Fluxos Secundários
  - Os Fluxos Secundários são identificados inspecionando o Fluxo Básico.
  - Em cada passo do Fluxo Básico, procure por:
    - Possíveis fluxos alternativos;
    - Erros que podem ocorrer;
    - Interrupções que podem ocorrer – coisas que podem acontecer em qualquer tempo.

Use Cases Complexos

- Quantos cenários existem?
  - Um use case possui apenas um cenário principal (fluxo básico) e pode ter vários cenários secundários (fluxo secundário).
  - Você deve tentar limitar o número de fluxos secundários ao mínimo necessário. Para isso, existem duas estratégias:
    - Pegue os fluxos secundários mais importantes e documente-os.
    - Onde existir grupos de fluxos secundários muito similares, documente um deles como um exemplo e (se necessário) adicione notas nesse exemplo sobre as diferenças entre eles.

Use Cases Complexos

- O princípio básico na modelagem use case é manter a quantidade de informações no mínimo necessário.
- Isso significa que muitos fluxos secundários nunca serão especificados.
- Lembre-se que o objetivo da modelagem use case é identificar use cases e seus cenários para entender o comportamento do sistema e não com o propósito de criar um modelo use case completo.
- Assim, a modelagem use case pára quando houver um sentimento de que atingiu o entendimento.
- Você sempre poderá retornar e acrescentar detalhes quando achar que algum aspecto do comportamento do sistema foi esquecido ou não foi realmente entendido.

Use Cases Complexos

- Quando aplicar a modelagem use case
  - Pelo fato de use cases capturarem as funções do sistema sob o ponto de vista de seus atores, a modelagem use case não são tão apropriados para sistema que tenha apenas um ou nenhum ator.
  - Use cases capturam requisitos funcionais.
  - Use cases não capturam requisitos não-funcionais.

Use Cases Complexos

- Use cases são a melhor escolha quando:
  - O sistema é dominado por requisitos funcionais
  - O sistema tiver muitos tipos de usuários e que apresentem diferentes funcionalidades.
  - O sistema tiver muitas interfaces.
- Use cases não é uma boa escolha quando:
  - O sistema é dominado por requisitos não-funcionais
  - O sistema tem poucos usuários.
  - O sistema tiver poucas interfaces.

Requisitos de Software

Modelagem Use Case Avançada

Professor: Osvaldo Kotaro Takai
E-mail: otakai@uol.com.br

Modelagem Use Case Avançada

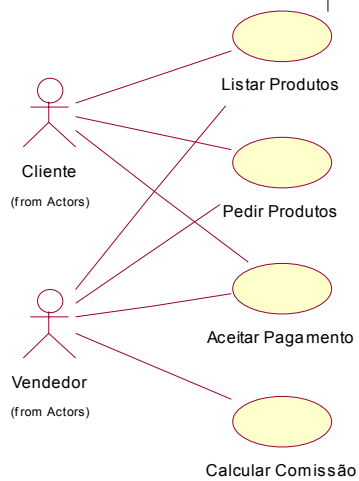
- Discutiremos relacionamentos que são possíveis entre Atores e Atores e entre Use Cases e Use Cases. Tais relacionamentos são:
  - Generalização de Atores: um relacionamento de generalização entre um Ator mais genérico e um Ator mais específico.
  - Generalização Use Case: um relacionamento de generalização entre um Use Case mais geral e um Use Case mais específico.
  - <<include>>: relacionamento entre Use Cases que permite que um Use Case inclua comportamento de outro.
  - <<extend>>: relacionamento entre Use Cases que permite que um dos Use Cases estenda seu comportamento com um ou mais fragmentos de comportamentos de outro.

Modelagem Use Case Avançada

- É muito importante manter todos os modelos tão simples quanto possível. Assim, esses relacionamentos devem ser usados com ponderação.
- Use-os somente quando a claridade do modelo Use Case for beneficiada.
- É muito fácil saturar o modelo com <<include>> e <<extend>>, por isso, evite-os.

Modelagem Use Case Avançada

- Generalização de Atores
  - No exemplo da figura ao lado, você pode ver que existem algumas coisas comuns entre os atores: Cliente e Vendedor, na maneira como eles interagem com o Sistema de Vendas. O Vendedor pode fazer um pedido no lugar de um Cliente.

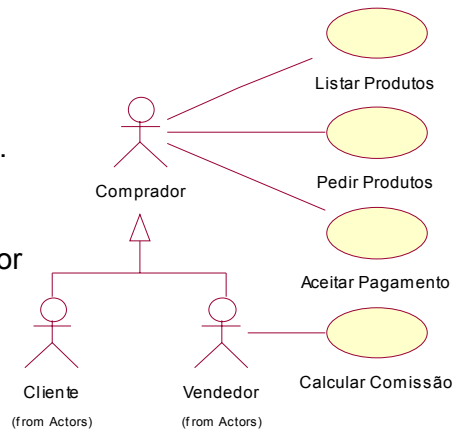


Modelagem Use Case Avançada

- Na realidade, existe apenas uma diferença entre os dois atores: o Vendedor interage com o use case Calcular Comissão.
- Deixando de lado o fato das várias linhas cruzadas no diagrama, essas similaridades de comportamentos parecem indicar que existem alguns comportamentos de atores comuns e que devem ser fatorados (decompostos) a partir de um ator mais genérico.
- Os fatores comuns podem ser generalizados num novo ator como ilustra a seguinte figura:

Modelagem Use Case Avançada

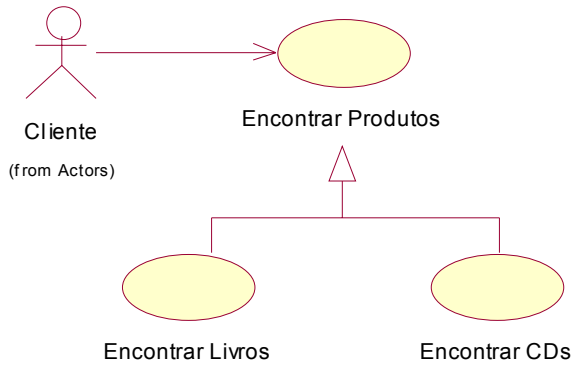
- Você cria um ator abstrato chamado Comprador.
- Os atores Cliente e Vendedor são concretos.
- Os atores concretos herdam os comportamentos do ator abstrato.



Modelagem Use Case Avançada

- Generalização Use Case
  - A generalização é usada quando tem um ou mais use cases que são especializações reais de um caso mais genérico.
  - Como na generalização de atores, você deve usá-la somente quando isso simplificar o diagrama use case.
  - Os use cases filhos podem:
    - Herdar características do use case pai
    - Adicionar novas características
    - Sobrepor (mudar) características herdadas

Modelagem Use Case Avançada



Modelagem Use Case Avançada

- Como documentar generalização use cases numa especificação use case?
- A especificação UML não define isso.
- Nós preferimos usar uma simples convenção tipográfica para destacar 3 possibilidades no use case filho:

A característica é	Convenção tipográfica
Herdar sem mudar	Texto normal
Sobrepor	*Texto em itálico*
Adicionar	**Texto em negrito**

Modelagem Use Case Avançada

Use Case: Encontrar Produtos
ID: UC12
Atores: Cliente
Pré-condições:
Fluxo de Eventos: 1. O Cliente seleciona a opção "encontrar produtos". 2. O sistema solicita o critério de busca ao Cliente. 3. O Cliente informa o critério solicitado. 4. O sistema procura por produtos de acordo com o critério de busca informado pelo Cliente. 5. **Se** o sistema encontrar produtos que satisfaçam o critério de busca **então** 5.1. O sistema mostra uma lista com os produtos encontrados 6. **Senão** 6.1. O sistema informa ao Cliente que nenhum produto pôde ser encontrado.
Pós-condições:
Fluxo Alternativo: 1. Em qualquer momento, o Cliente ir para uma página diferente.
Pós-condições:

Especificação do Use Case abstrato

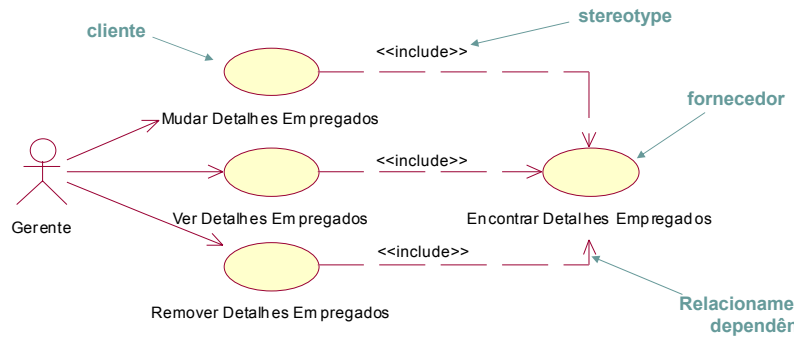
Modelagem Use Case Avançada

Use Case: Encontrar Livro
ID: UC16
ID do Use Case Pai: UC12
Atores: Cliente
Pré-condições:
Fluxo de Eventos: *1. O Cliente seleciona a opção "encontrar livro".* *2. O sistema solicita o critério de busca do livro ao Cliente.* 3. O Cliente informa o critério solicitado. *4. O sistema procura pelo livro de acordo com o critério de busca informado pelo Cliente.* ***5. Se** o sistema encontrar o livro que satisfaçam o critério de busca **então*** 5.1. *O sistema mostra uma página contendo os detalhes de no máximo 5 livros* ***5.2.* Para cada livro da lista na página Faça** **5.2.1 mostre o título, autor, preço e ISBN** **5.3. Enquanto existir mais livros Faça** **5.3.1. O sistema dá ao cliente a opção de mostrar a próxima página** **6. Senão** 6.1. *O sistema informa ao Cliente que nenhum livro pôde ser encontrado.*
Pós-condições:
Fluxo Alternativo: 1. Em qualquer momento, o Cliente ir para uma página diferente.
Pós-condições:

Especificação do Use Case Filho

Modelagem Use Case Avançada

- **<<include>>**
  - Descrever use cases pode ser, algumas vezes, muito repetitivo.
  - Suponha que se esteja desenvolvendo um sistema de RH:



Modelagem Use Case Avançada

- Quase qualquer coisa que pedimos ao sistema envolve, primeiro, localizar os detalhes de um empregado específico.
- Se você tiver que descrever esta seqüência de eventos (autenticação do ID empregado, encontrar empregado, etc.) todas as vezes que você precisar dos detalhes de um empregado, então sua descrição de use cases pode ser tornar um tanto repetitiva.
- O relacionamento <<include>> entre use cases permite incluir o comportamento de um use case fornecedor dentro do fluxo de um use case cliente.

Modelagem Use Case Avançada

- O use case que *inclui* é chamado de *cliente*, e o *incluído* como *fornecedor*. Isso porque, o use case incluído fornece comportamento ao use case cliente.
- Você deve especificar o ponto *exato* no use case cliente onde você precisa que o use case fornecedor seja incluído.
- A sintaxe do <<include>> é muito parecida com a chamada de função.
- A semântica do <<include>> é muito simples, veja as seguintes descrições:

Modelagem Use Case Avançada

Mudar Detalhes Empregados
ID: UC1
Atores: Gerente
Pré-condições: 1. O Gerente é válido no sistema
Fluxo de eventos: 1. O Gerente entra com o ID do empregado. 2. Inclua (Encontrar Detalhes Empregados). 3. O Gerente seleciona parte dos detalhes de empregados para mudar. 4. ...
Pós-condições:

Ver Detalhes Empregados
ID: UC2
Atores: Gerente
Pré-condições: 1. O Gerente é válido no sistema
Fluxo de eventos: 1. O Gerente entra com o ID do empregado. 2. Inclua (Encontrar Detalhes Empregados). 3. O sistema mostra os detalhes de empregados. 4. ...
Pós-condições:

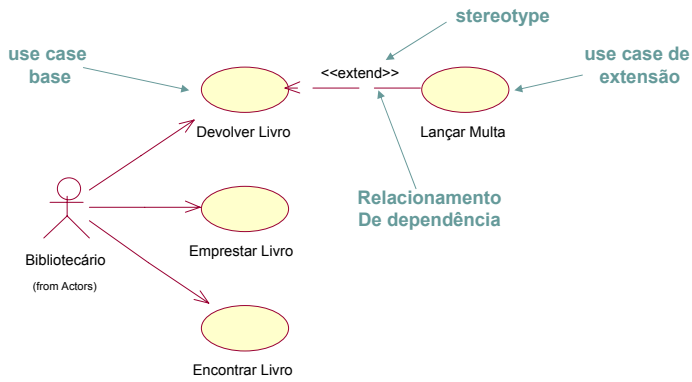
Remover Detalhes Empregados
ID: UC3
Atores: Gerente
Pré-condições: 1. O Gerente é válido no sistema
Fluxo de eventos: 1. O Gerente entra com o ID do empregado. 2. Inclua (Encontrar Detalhes Empregados). 3. O sistema mostra os detalhes do empregado. 4. O Gerente remove os detalhes do empregado. 5. ...
Pós-condições:

Modelagem Use Case Avançada

- O use case cliente executa até encontrar o ponto de inclusão, então a execução passa para o fornecedor.
- Quando o fornecedor termina, o controle volta ao cliente.
- O cliente não é completo sem os seus fornecedores.
- O fornecedor é parte integrante do cliente.
- No entanto, o use case fornecedor pode:
  - **Não ser completo**: nesse caso, contém apenas a parte do fluxo de eventos que terá sentido apenas quando incluído num cliente (fragmento de comportamento, não instanciável, não pode ser ativado diretamente por um ator).
  - **Ser completo**, nesse caso ele atua como um use case normal (instanciável, pode ser ativado por um ator)

Modelagem Use Case Avançada

- **<<extend>>:**
  - Fornece uma maneira de adicionar um novo comportamento a um use case existente. Veja a figura:



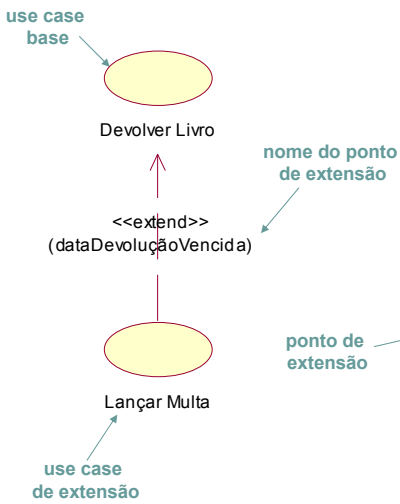
Modelagem Use Case Avançada

- O use case base fornece um conjunto de pontos de extensão que são “ganchos” onde novos comportamentos podem ser adicionados.
- Os use cases de extensão podem adicionar um conjunto de segmentos de inserção que podem ser inseridos dentro do use case base em seus “ganchos”.
- O relacionamento <<extend>> pode ser usado para especificar exatamente qual ponto de extensão dentro do use case base será estendido.
- O interessante do <<extend>> é que o use case base não tem nenhum comhecimento sobre os use cases de extensão – ele apenas fornece os “ganchos”.
- De fato, o use case base é perfeitamente completo sem suas extensões, diferentemente do <<include>> onde os use cases clientes não são completos sem os seus use cases de inclusão.

Modelagem Use Case Avançada

- Mais ainda, os pontos de extensão não são verdadeiramente inseridos no fluxo de eventos do use case base; ao invés disso, eles ficam acima do fluxo de eventos.
- Pontos de extensão são indicados no fluxo de eventos do use case base como ilustra a figura ao seguinte:

Modelagem Use Case Avançada



Devolver Livro
ID: UC9
Atores: Bibliotecário
Pré-condições: 1.O Bibliotecário é válido no sistema
Fluxo de eventos: 1. O Bibliotecário entra com o ID do emprestador. 2. O sistema mostra os detalhes do emprestador incluindo a lista de livros emprestados. 3. O Bibliotecário procura o livro devolvido nessa lista. <dataDevoluçãoVencida> 4. O Bibliotecário dá baixa no livro. 5. ...
Pós-condições: O livro foi devolvido.

Modelagem Use Case Avançada

- Note que o ponto de extensão no fluxo base não é numerado, ele aparece entre os passos numerados do fluxo.
- Portanto, eles não fazem parte do fluxo principal. Você pode pensar nesta sobreposição como um filme de acetato sobre o fluxo principal, onde os pontos de extensão são registrados.
- Em outras palavras, o fluxo do use case base não sabe ou não se preocupa onde ele está sendo estendido.
- Isso permite que você use o <<extend>> para fazer extensões arbitrárias para um fluxo do use case base.
- No exemplo, você pode ver que o use case base Devolver Livro tem um ponto de extensão chamado <dataDevoluçãoVencida> entre os passos 3 e 4 do fluxo de eventos.

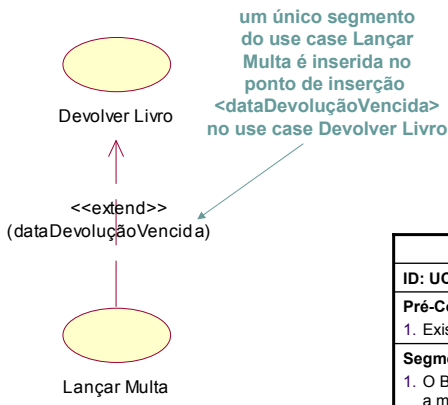
Modelagem Use Case Avançada

- Você pode ver que <<extend>> fornece uma boa maneira de lidar com casos excepcionais, ou casos em que você precisa de uma estrutura flexível porque você não pode predizer (ou apenas não sabe) todas as possíveis extensões.

Modelagem Use Case Avançada

- **Os Use Cases de Extensão**
  - Geralmente, os use cases de extensão não são completos e, assim, não podem ser instanciados.
  - Normalmente consistem apenas de alguns fragmentos de comportamento conhecidos como *segmentos de inserção*.
  - O relacionamento <<extend>> especifica os pontos de extensão no use case base onde o *seguimento de inserção* será inserido. As seguintes regras se aplicam:
    - O relacionamento <<extend>> deve especificar um ou mais pontos de extensão no use case base ou que o se refere a todos os pontos de extensão.
    - O use case de extensão deve ter o mesmo número de *segmentos de inserção* do que os pontos de extensão especificados no relacionamento <<extend>>.
    - É possível que dois use cases de extensão estendam um mesmo use case base num mesmo ponto de extensão. Se isso ocorrer, a ordem de execução das extensões é indeterminada.

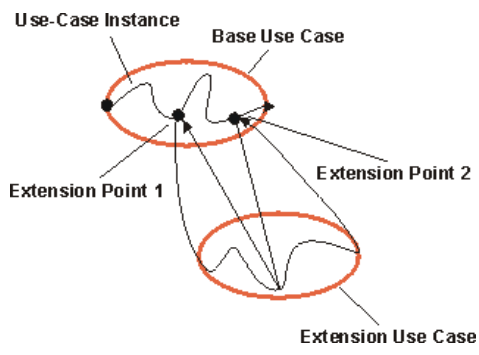
Modelagem Use Case Avançada



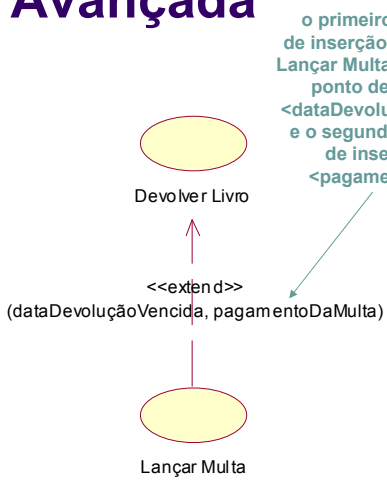
Use case de extensão: Lançar Multa
ID: UC10
Pré-Condições: 1. Existe livros em atrasado.
Segmento de Inserção: 1. O Bibliotecário usa o sistema para registrar e imprimir a multa.
Pós-condições: Multa registrada.

Modelagem Use Case Avançada

- **Múltiplos Segmentos de Inserção:**
  - Você pode ter múltiplos segmentos de inserção num use case de extensão.
  - Isso é útil quando você não consegue capturar a extensão claramente num único segmento de inserção devido a necessidade de retornar ao fluxo principal do use case base para fazer alguma coisa.



Modelagem Use Case Avançada



Use case de extensão: Lançar Multa
ID: UC10
Pré-Condições: 1. Existe livros em atrasado.
Segmento de Inserção 1: 1. O Bibliotecário usa o sistema para registrar e imprimir a multa.
Segmento de Inserção 2: 1. Se o emprestador escolher pagar o total da multa imediatamente então 1.1. O Bibliotecário aceita o pagamento da multa. 1.2. ...
Pós-condições: Multa registrada.

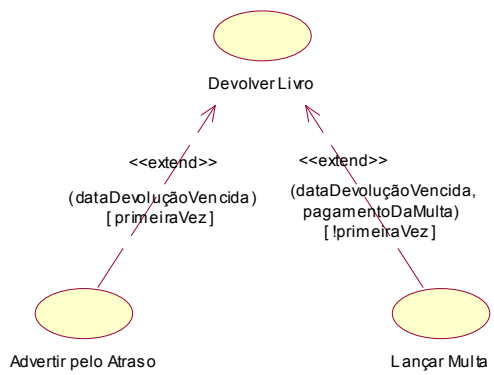
Modelagem Use Case Avançada

- No exemplo, você pode imaginar que depois de registrar e imprimir a multa, voltamos para o fluxo principal para processar o próximo livro com data de vencimento em atraso e, então, finalmente, no ponto de extensão <pagamentoDaMulta>, nós damos ao emprestador a opção de pagar toda a multa.
- Isto é claramente mais eficiente do que ter que imprimir e receber o pagamento para cada multa individualmente, o que seria o caso se tivéssemos combinado os dois seguimentos em um único seguimento de Lançar Multa.
- Esse exemplo é interessante porque você pode ver o segundo seguimento de inserção começando com a declaração **Se**. Como tal, ele é um fluxo condicional, e só por isso, torna-se um bom candidato para ser um use case de extensão.
- Use cases de extensão, por sua vez, podem ter extensões e inclusões. Mas, cuidado; não é muito bom ter muitos <<include>> e <<extend>> num modelo use case.

Modelagem Use Case Avançada

- **Extensões Condicionais**
  - O exemplo a seguir ilustra um sistema de biblioteca um pouco mais agradável onde o emprestador é advertido na primeira vez em que um livro é entregue em atraso e cobrado a multa nas outras vezes.
  - Podemos modelar isso adicionando um novo use case de extensão, Advertir pelo Atraso, e então colocar condições sobre os relacionamentos de extensão.
  - As condições são expressões booleanas, e a inserção é feita, se e somente se, a expressão avaliada for verdadeira.

Modelagem Use Case Avançada



Modelagem Use Case Avançada

- **Quando usar Características Avançadas**
  - Use as características avançadas quando elas simplificarem o modelo use case.
  - Lembre-se que o modelo use case é uma declaração de requisitos e, como tal, deve ser acessível aos stakeholders e modeladores.
  - Com base em nossa experiência em várias empresas:
    - Geralmente, os stakeholders entendem facilmente atores e use cases com pouco de treinamento e auxílio.
    - Stakeholders acham difícil de entender a generalização de atores.
    - O uso abusivo de <<includes>> pode tornar o modelo difícil de entender.
    - Stakeholders acham muito difícil entender o <<extend>>.
    - Um número surpreendentemente grande de modeladores compreendem mal a semântica do <<extend>>
    - A generalização de use cases deve ser evitado.