Análise e Projetos Orientados a Objetos

Análise E Projeto Oo & Uml 2.0

Neste capítulo são apresentados os conceitos fundamentais do curso: modelo, UML, análise e projeto orientado a objetos, objeto e classes de objetos.

Modelo

Antes de entrar nos detalhes de UML, é preciso ater-se ao conceito de modelo.

Por exemplo, um projeto arquitetônico é feito segundo diversas perspectivas: do arquiteto, projeto arquitetônico em si, do engenheiro eletricista, projeto elétrico, do engenheiro civil, projeto hidráulico e estrutural. Construímos modelos de sistemas complexos para melhor compreendê-los.

Abstrair e refinar incrementalmente são palavras-chaves. Em certos momentos, o projetista deve focalizar na interação entre componentes do sistema sem se preocupar com seus detalhes internos de funcionamento, então ele abstrai estes detalhes.

Em outros momentos, é preciso detalhar o comportamento dos componentes. Enfim projetar um sistema significa fazer modelos sob diferentes perspectivas e graus de abstração, representando-os por meio de uma notação precisa, refinando-os sucessivamente até transformá-los em algo próximo da implementação lembrando sempre de verificar se os requisitos são satisfeitos.

A modelagem visual (com auxílio de diagramas) ajuda a manter a consistência entre os artefatos (produtos) ligados ao desenvolvimento de um sistema: requisitos, projeto e implementação.

Resumidamente, a modelagem visual pode melhorar a capacidade de uma equipe a gerenciar a complexidade de software.

UML

UML significa Unified Modeling Language ou Linguagem de Modelagem Unificada de projetos orientados a objetos. Como o próprio nome diz, UML é uma linguagem e não um método!

Por notação entende-se especificar, visualizar e documentar os elementos de um sistema OO. A UML é importante, pois:

◊ serve como linguagem para expressar decisões de projeto que não são óbvias ou que não podem ser deduzidas do código;

◊ provê uma semântica que permite capturar as decisões estratégicas e táticas;

◊ provê uma forma concreta o suficiente para a compreensão das pessoas e para ser manipulada pelas máquinas.

◊ É independente das linguagens de programação e dos métodos de desenvolvimento.

Breve histórico

Nos anos 90, conhecida como a época da “guerra dos métodos”, vários métodos coexistiam com notações muitas vezes conflitantes entre si. Dentre estes, os mais conhecidos eram:

◊ OMT (Object Modelling Technique) de Rumbaugh;

◊ Método de Booch;

◊ OOSE (Object Oriented Software Engineering) de Jacobson;

Inicialmente, Rumbaugh (OMT) e Booch fundiram seus métodos (e notações) resultando no Método Unificado em 1995 quando trabalhavam juntos na Rational Software (atualmente uma divisão da IBM). Jacobson juntou-se a eles mais tarde e seu método OOSE foi incorporado à nova metodologia (RUP).

Salienta-se que além do método, eles unificaram a notação de projeto e a chamaram UML. Então, UML representa a unificação das notações de Booch, OMT e Jacobson.

Também agrega as idéias de inúmeros autores, tais como Harel e seu diagramas de estados, Shlaer-Mellor e o ciclo de vida dos objetos. Em suma, UML é uma tentativa de padronizar os artefatos de análise e projeto: modelos semânticos, sintaxe de notação e diagramas.

Na década de 90, surge uma organização importante no mundo dos objetos a OMG (Object Management Group), uma entidade sem fins lucrativos onde participam empresas e acadêmicos para definirem padrões de tecnologias OO.

◊ Outubro de 1995: primeira versão rascunho, versão 0.8 draft.

◊ Julho de 1996: revisão devido ao ingresso de Jacobson, versão 0.9 draft.

◊ Parceiros UML (HP, IBM, Microsoft, Oracle e Rational Software) desenvolveram a versão 1.1 e a propuseram OMG

◊ A OMG aceita a proposta em novembro de 1997 e assume a responsabilidade de realizar manutenção é revisão da UML

◊ Em março de 2003, a OMG lançou a versão 1.5

◊ Em outubro de 2004, a OMG lançou versão 2.0 adotada1

Análise e Projeto Orientados a Objetos

Há vários métodos de desenvolvimento de software. Na década de 80 houve preponderância dos métodos estruturados. Atualmente o paradigma OO é mais forte e, por isso, é importante diferenciar análise e projeto estruturado e orientado a objetos.

Vários autores participam da corrente de análise, projeto e programação estruturados: 1979 - Tom DeMarco: Análise estruturada (DEMARCO, 1989)

1982 - Gane e Sarson: Análise estruturada (GANE & SARSON, 1983)

1985 - Ward e Mellor: Análise estruturada para sistemas tempo real (WARD & MELLOR, 1986) 1989 - Yourdon: Análise estruturada moderna (YOURDON, 1990)

Na análise e projeto estruturados, o processo a ser informatizado é visto como um conjunto de funções com dados de entrada, processamento e dados de saída, ou seja, a ênfase esta em funções que agem sobre dados. Estas funções podem ser decompostas em subfunções (decomposição funcional). As principais características são:

◊ preocupação com a modularidade e coesão;

◊ desenvolvimento em diferentes níveis de abstração (top-down).

Os principais diagramas empregados nas diversas metodologias estruturadas são:

◊ dicionários de dados, modelagem do fluxo de dados (DFD);

◊ modelos de dados: diagrama entidade e relacionamento (DER) e modelo entidade-relacionamento (MER).

Análise e Projeto Orientados a Objetos

Análise, projeto e programação orientados a objetos: Coad e Yourdon (1979), Rumbaugh (1991), Grady Booch (1991), Jacobson(1992).

Diferentemente da análise e projeto estruturados, na orientação a objetos o processo a ser informatizado é visto como um conjunto de objetos que interagem para realizar as funções. As vantagens do modelo OO são:

◊ maior grau de abstração;

◊ maior encapsulamento;

◊ modelos apoiados em conceitos do mundo real;

◊ reutilização (reusabilidade).

Neste curso, não é abordado o ciclo de vida de desenvolvimento de software que são inúmeros: cascata, iterativo, incremental, ágil, extremo e outros. No entanto, as fases clássicas do ciclo de vida são utilizadas (engenharia de requisitos, análise, projeto, implementação, testes, manutenção e operação).

Objeto e Classe

Apresenta-se uma breve revisão de objeto e classes de objeto assim como a notação UML de ambos.

Objeto

É uma abstração que representa uma entidade do mundo real pode ser algo concreto (computador, carro) ou abstrato (transação bancária, histórico, taxa de juros). Um objeto num sistema possui três propriedades: estado, comportamento e identidade.

◊ Estado: definido pelo conjunto de propriedades do objeto (os atributos) e de suas relações com os outros objetos. É algo que muda com o tempo, por exemplo, um objeto turma pode estar no estado aberto ou fechado. Inicia no estado aberto e fecha quando 10 alunos fazem inscrição.

◊ Comportamento: como um objeto responde às solicitações dos outros e tudo mais o que um objeto é capaz de fazer. É implementado por um conjunto de operações. Ex. objeto turma pode ter operações acrescentar aluno ou suprimir aluno.

◊ Identidade: significa que cada objeto é único no sistema. Por exemplo, o objeto turma Tecno-OO manhã é diferente do objeto Tecno-OO tarde.

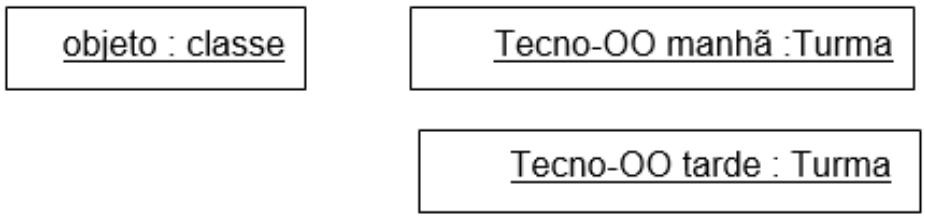

Figura 1. Notação de objeto em UML

Classe

Uma classe é uma descrição de um conjunto de objetos com propriedades, comportamento, relacionamentos e semântica comuns. Uma classe pode ser vista como um esqueleto/modelo para criar objetos.

Exemplo: classe turma

◊ Atributos: sala, horário

◊ Operações: obter local, adicionar estudante, obter horário

Dicas

◊ Classes devem encerrar uma só abstração do mundo real. Por exemplo, uma classe estudante contendo também o histórico do estudante não é uma boa solução. Melhor é dividi-la em duas classes: estudante e histórico.

◊ Utilizar substantivos para nomear as classes.

◊ Nas fases iniciais de desenvolvimento, pode-se suprimir os atributos e os métodos deixando somente os compartimentos.

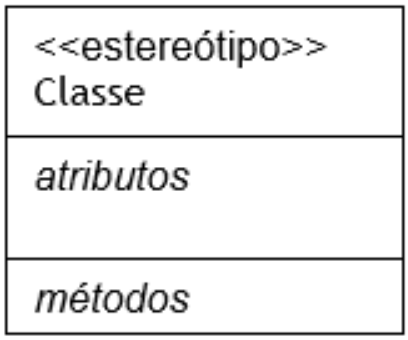

Figura 2. Notação UML para classe.

Noção Geral de Análise e Projeto Oo

Objetivo deste capítulo é apresentar de maneira geral o método de análise e projeto de sistemas orientados a objetos utilizado durante o curso.

São descritas as fases principais do método e os diagramas UML mais indicados para cada uma delas. Este método é uma simplificação do RUP (Rational Unified Process).

Visão Geral

No curso, seguiremos o seguinte método:

Análise de requisitos: lista de requisitos funcionais e não-funcionais e diagrama de Casos de Uso.

Levantamento das classes candidatas: montar o diagrama de classes com um levantamento preliminar das classes, com atributos, métodos e relações (quando possível).

Estudo da interação entre objetos: diagramas de interação

Refinamento do diagrama de classes

Definição do comportamento de classes com estado através de máquinas de estados e diagrama de atividades

Modelo de implantação

Modelo de implementação

Codificação

Os passos de 3 a 5 se repetem até que se chegue próximo da implementação. Eventualmente é preciso revisar os requisitos e verificar as implicações das mudanças no projeto.

Análise De Requisitos

Consiste em determinar os serviços que o usuário espera do sistema e as condições (restrições) sob as quais o sistema será desenvolvido e operar. As necessidades do usuário podem ser muito variadas, o analista deve ser capaz de retirar os requisitos funcionais e não-funcionais destas necessidades:

◊ Funcionais: lista de serviços que o sistema deve oferecer ao usuário.

◊ Não funcionais: propriedades e características desejadas do sistema relativas à capacidade de armazenamento, tempo de resposta, configuração, uso (ex. uso intuitivo), confiabilidade, etc.

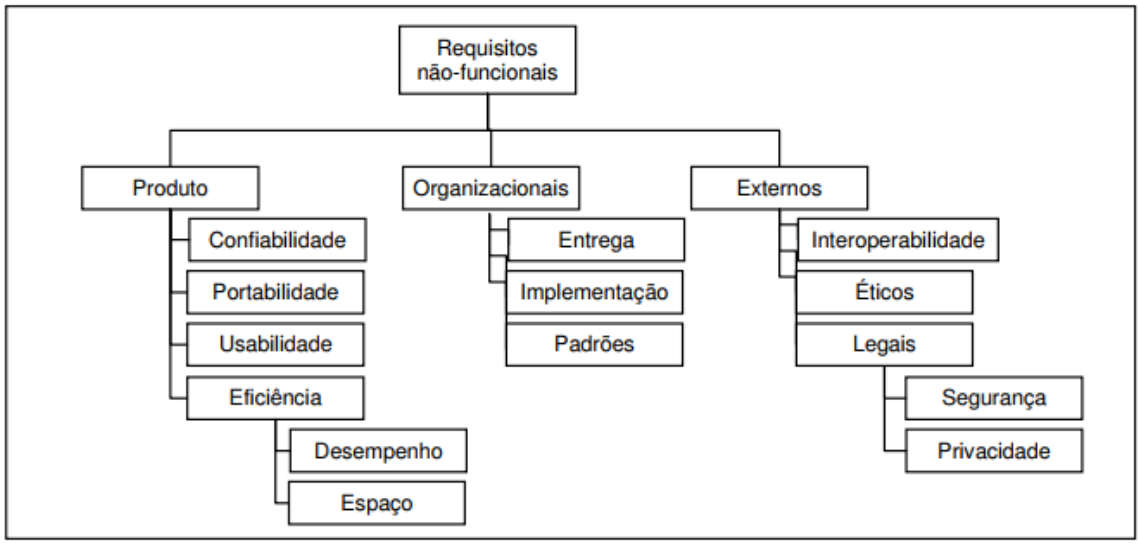

Figura 3: Taxonomia de requisitos não-funcionais (extraída de

Papel dos Casos de Uso na Análise de Requisitos

Casos de uso representam funcionalidades completas para o usuário e não, funcionalidades internas do sistema. Outro ponto importante é que o diagrama de casos de uso é um artefato de comunicação entre cliente, usuários e desenvolvedores. Por ser extremamente simples e, consequentemente, de fácil compreensão, incentiva a participação do cliente e usuários no processo de desenvolvimento. Também serve como um contrato entre a equipe/empresa desenvolvedora e o cliente.

Casos de Uso

A coleção de casos de uso representa todos os modos pelos quais o sistema pode ser utilizado pelos atores envolvidos. Um caso de uso é uma seqüência de ações realizadas colaborativamente pelos atores envolvidos e pelo sistema que produz um resultado significativo (com valor) para os atores. Um ator pode ser um usuário ou outro sistema.

Para uma calculadora de linha de comando cujo objetivo é executar expressões aritméticas (ex. -2 + 3*5), o diagrama de casos da figura 4 pode ser considerado adequado.

Figura 4. Diagrama de casos de uso para a calculadora.

Diagrama de casos de uso é apenas um panorama visual das funcionalidades do sistema, é necessária uma descrição textual para detalhar os casos de uso. A tabela 1 ilustra esta documentação para o caso de uso resolver expressões aritméticas.

Tabela 1: documentação para o caso de uso resolver expressões aritméticas básicas.

Nome do caso de uso	Efetuar expressões aritméticas básicas
Descrição	Permite resolver expressões envolvendo números inteiros e reais e as operações básicas de soma, subtração, multiplicação e divisão sem parênteses.
Ator Envolvido	Usuário
Pré-condições	Sistema deve estar em execução aguardando por uma expressão

Pós-condições	Expressão aritmética resolvida ou abandono da expressão pelo usuário.
Fluxo básico	
Usuário	Sistema
	{Solicita expressão}
	Solicita a expressão. (A2)
Fornece a expressão	
	{Valida expressão}
	Avalia se a expressão é sintaticamente correta (A1)
	{Calcula valor}
	Calcula o valor da expressão.
	{Mostra o resultado}
	Informa o resultado da expressão.
	{Fim} Fim do caso de uso.
Fluxos alternativos	
A1: em {Valida expressão}	Se o usuário fornecer uma expressão sintaticamente incorreta, informá-lo sobre o erro e retomar o fluxo básico em {Solicita expressão}
A2: em {Solicita expressão}	usuário pode decidir encerrar o caso de uso sem fornecer uma expressão. Nesse caso retomar o fluxo básico em {Fim}

Portanto, a saída da fase de análise de requisitos é composta por:

◊ lista de requisitos funcionais e não-funcionais;

◊ diagrama de casos de uso e definições textuais dos casos.

Análise e Projeto

Análise é a solução conceitual dada ao problema. Marca o início da definição informática, mas sem levar em conta detalhes da implementação tais como a linguagem a ser utilizada e o sistema gerenciador de banco de dados. Preocupa-se principalmente com as classes do domínio do problema e suas relações e também com os casos de uso.

Projeto é a solução informática dada ao problema. A separação entre análise e projeto é tênue, pois o projeto acaba sendo o resultado de sucessivos refinamentos do modelo conceitual de análise.

Diagramas de Interação

Há vários tipos de diagramas de interação na UML. Exemplifica-se o uso do diagrama de seqüência cuja utilidade é estudar as interações entre os objetos com o objetivo de refinar o diagrama de classes, identificando relações entre classes, seus métodos e atributos. A figura 5 mostra um cenário onde o usuário fornece uma expressão aritmética sintaticamente correta.

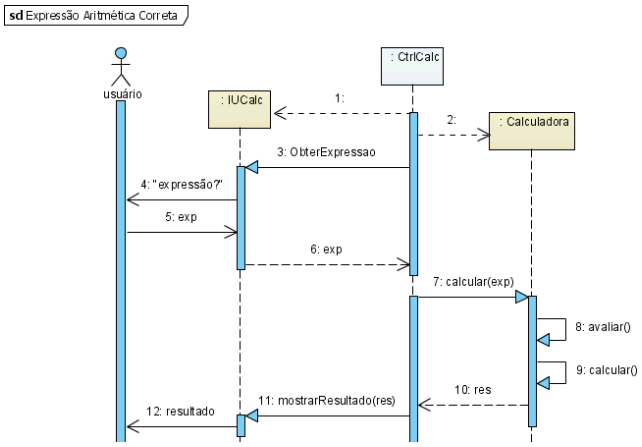

Figura 5. Diagrama de seqüência para a calculadora.

Refinamento do Diagrama de Classes

A partir das informações do diagrama de seqüência, é possível:

◊ identificar métodos associados às classes. Por exemplo, a classe IUCalculadora deve ter um método mostrarResultado(<resultado>).

◊ identificar as relações entre classes. Pelo diagrama de seqüência, fica clara a existência de uma relação de dependência entre a classe de controle e as duas outras como ilustra a figura 6.

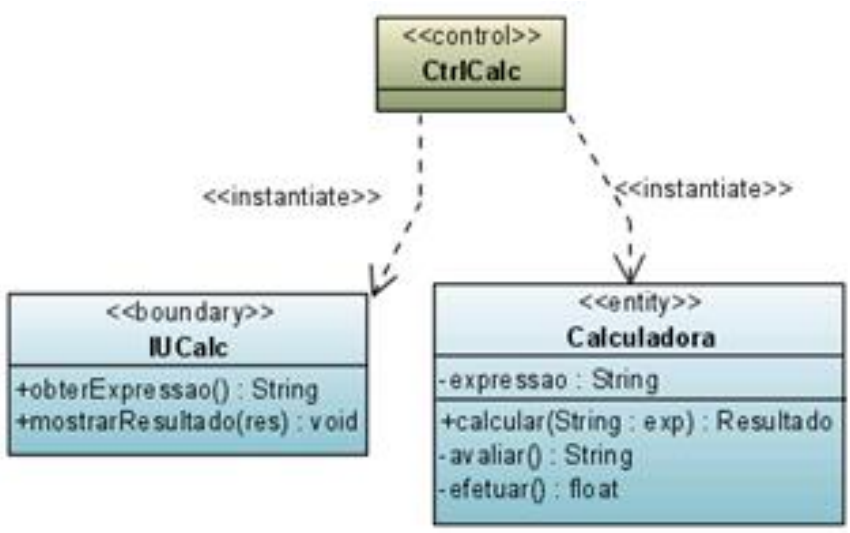

Figura 6. Diagrama de classes.

Definir o Comportamento das Classes

Nem todas as classes de um sistema possuem mais de um estado. Para as classes mais complexas, podemos especificar seus comportamentos utilizando máquinas de estado. A figura 7 mostra uma possível máquina de estados para a calculadora.

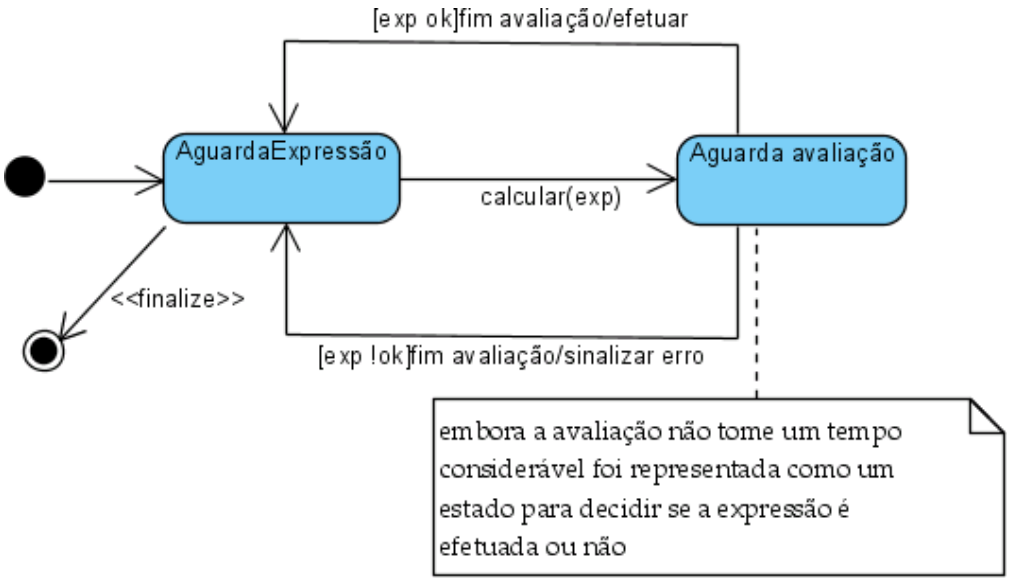

Figura 7. Máquina de estados para calculadora.

Os métodos de uma classe podem ainda ser detalhados por meio de um diagrama de atividades como ilustra figura 8.

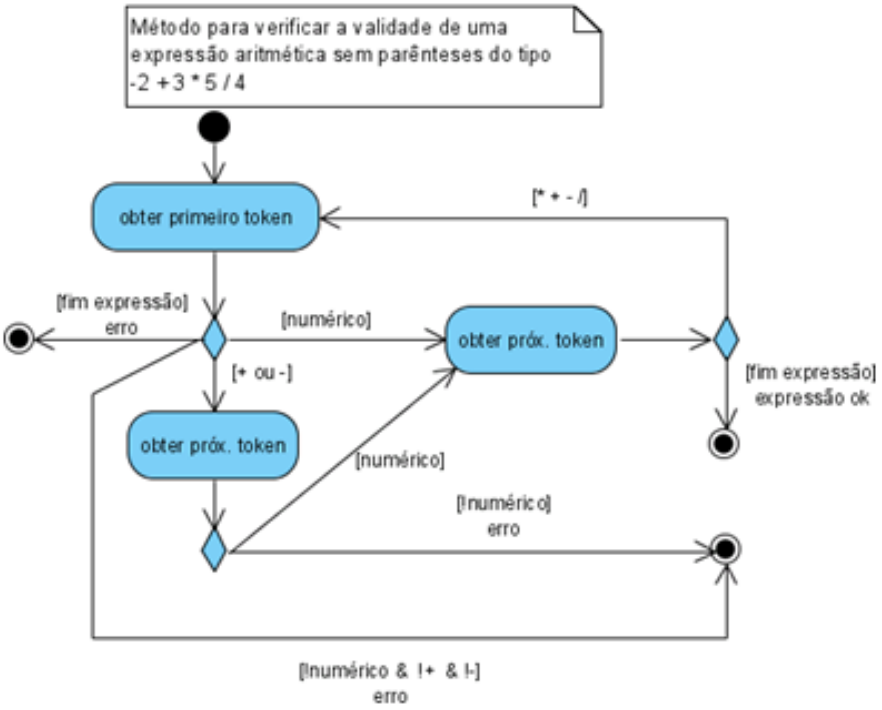

Figura 8: diagrama de atividades - detalhe de um método para verificar a sintaxe de expressão aritmética.

Implantação

diagrama de implantação representa as necessidades de hardware e sofware básico (ex. servidores). Para tornar o diagrama mais realista, a figura 9 supõe que a calculadora é um serviço ofertado por um servidor de aplicações Web.

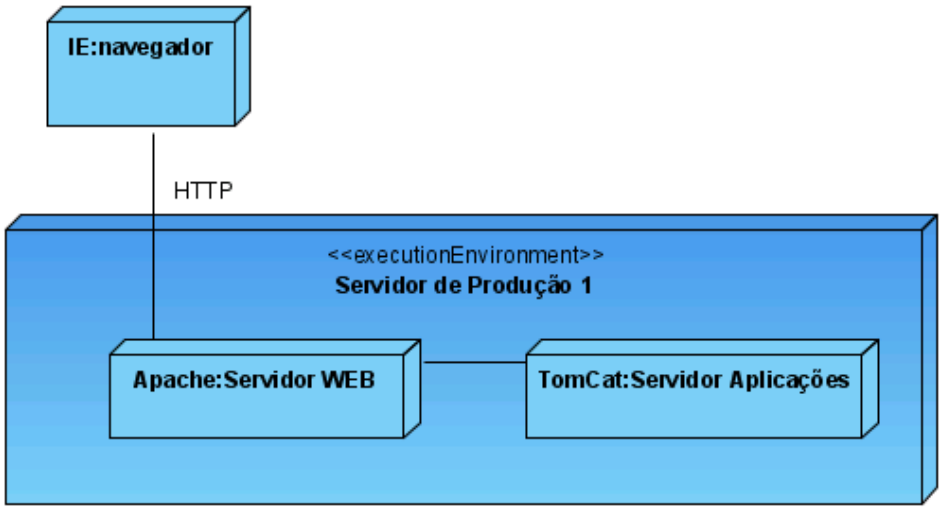

Figura 9: diagrama de implantação (deployement).

Componentes do Sistema

objetivo é documentar os componentes do sistema (fontes, bibliotecas) e suas relações. A figura 10 ilustra o diagrama de componentes para a calculadora e mostra a dependência entre seus componentes.

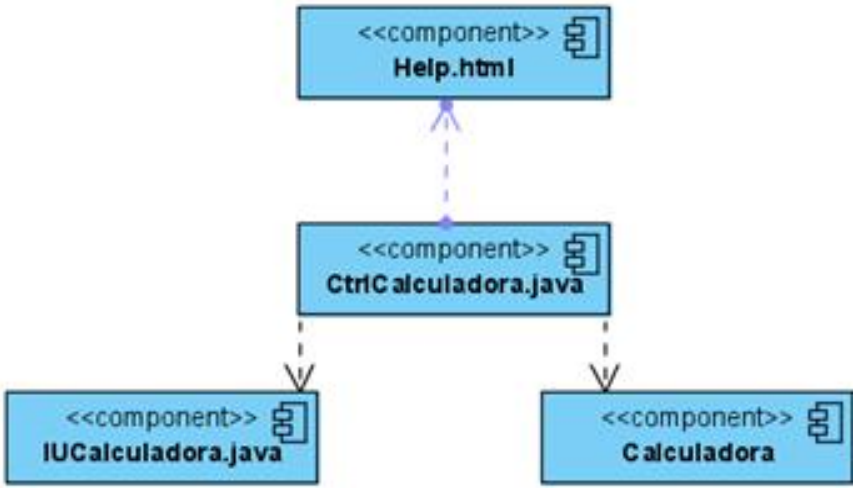

Figura 10: diagrama de componentes para a calculadora.

Modelagem Estrutural E Comportamental

Com esta rápida introdução à UML, é possível observar que alguns diagramas são mais indicados para modelar a estrutura do sistema e outros, o comportamento. A figura 11 mostra esta divisão.

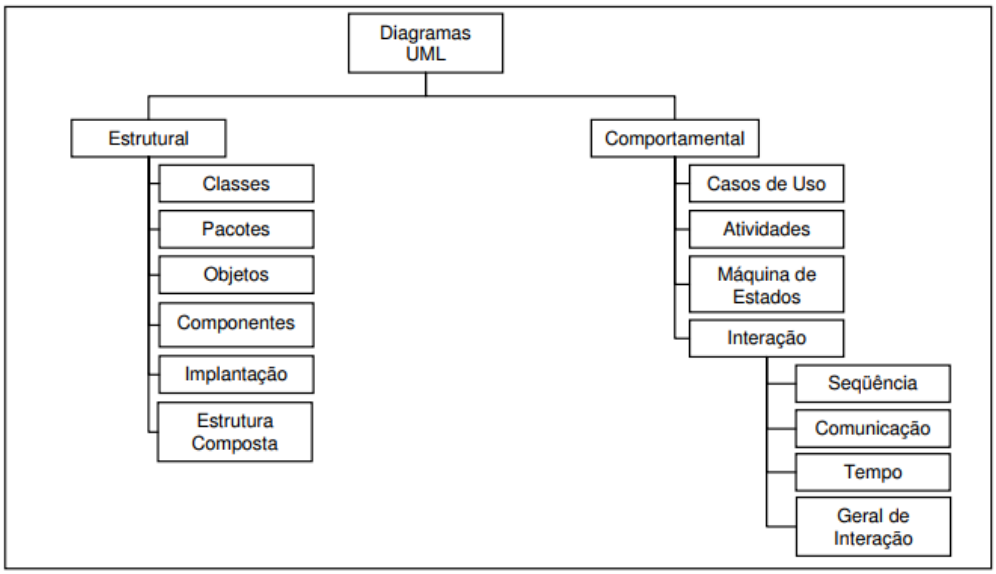

Figura 11. Diagramas estruturais e comportamentais da UML.

Segue uma breve descrição dos diagramas UML ainda não descritos neste documento:

Pacotes: representa uma coleção de classes que juntas formam uma unidade. Também pode servir para agrupar um conjunto de casos de uso com similaridades funcionais. Os pacotes podem apresentar relações, por exemplo, um pacote de classes pode depender de outro para executar suas funções.

Objetos: É um instantâneo da execução do sistema, retrata os objetos instanciados e suas relações em um dado momento.

Componentes: segundo a definição de (OMG, 2007, pg. 146), um componente é um módulo ou parte de um sistema que encapsula seu conteúdo (comportamento e dados). Um componente exibe seu comportamento através de interfaces bem definidas e pode depender de outros componentes.

Deployement (Implantação ou Distribuição): para representar a arquitetura física do sistema, ou seja, para representar as relações entre os componentes (artefatos) e os locais de execução (nodos: máquinas ou sistemas servidores).

Estrutura Composta: “descreve a estrutura interna de uma classe ou componente, detalhando as partes internas que o compõe como estas se comunicam e colaboram entre si” (Guedes, 2004).

Atividades: pode ser utilizado para diversos fins, um deles é a especificação mais detalhada de métodos complexos ou do encadeamento dos casos de uso.

Interação.Comunicação: mostra as interações entre uma coleção de objetos sem a linha do tempo (pode ser obtido do diagrama de seqüência e vice-versa).

Interação.Tempo: mostra o estado de um objeto ao longo do tempo.

Interação.Geral: é a fusão do diagrama de atividades com o de seqüência. Permite fazer referência a diagramas de seqüência e combiná-los com controle de fluxo (ex. pontos de decisão, forks e joins).

Modelo de Casos de Uso

Modelo de casos de uso (que é mais do que o diagrama) é o principal resultado da fase de análise de requisitos. Diagramas de casos de uso são utilizados para representar de forma panorâmica os requisitos funcionais de um sistema do ponto de vista do usuário. Cabe salientar que há outras utilizações possíveis para o diagrama de casos de uso, tal como modelagem do negócio, porém, o foco nesta seção está na representação de requisitos funcionais do usuário.

Definição

É um diagrama utilizado na análise de requisitos com objetivos claros:

Compreender o problema.

Delimitar o sistema (quem está no entorno).

Definir as funcionalidades oferecidas ao usuário (não há preocupação com a implementação).

Os elementos básicos de um diagrama de casos de uso são:

◊ atores,

◊ casos de uso e

◊ relações entre os mesmos.

Atores

◊ Representam papéis desempenhados por usuários ou qualquer outra entidade externa ao sistema (ex. hardware, outros sistemas)

◊ Podem iniciar casos de uso

◊ Podem prover e/ou receber informações dos casos de uso

Figura 12: Notação UML para ator.

Como encontrar atores de um sistema

◊ Examinar o problema procurando por pessoas ou sistemas do entorno.

◊ Quais as pessoas ou departamentos interessados num determinado requisito funcional?

◊ Quem irá suprir o sistema com informações e quem irá receber informações do sistema?

◊ Quais os recursos externos utilizados pelo sistema?

◊ Uma pessoa desempenha diferentes papéis?

◊ O sistema interage com outros sistemas já existentes?

Há fluxos primários ou básicos (fluxo normal de eventos) e alternativos (o que fazer se…). Para descrevê-los, é possível se inspirar na situação em que uma pessoa explica um caminho à outra. Primeiro, o fluxo básico é explicado, depois, as alternativas.

> Para ir ao churrasco, pegue a BR116 na direção São Paulo. Logo após o clube Santa Mônica, tem um retorno por baixo da pista. Faça o retorno e continue reto (não retorne à BR). Continue nesta estradinha asfaltada por 1 km, no entroncamento pegue a estrada de terra à direita, ande cerca de 500m, você verá um grande eucalipto e uma araucária. A entrada da chácara é entre os dois. Não se esqueça de trazer o pinhão. // ***primário***
>
> Se estiver chovendo muito, os 500m na terra podem ser bem difíceis porque o barro é mole. Neste caso, siga reto no entroncamento (ao invés de virar à direita) e na próxima a direita pegue a rua de paralelepípedos. Ande cerca de 1 km e depois vire na segunda a direita que vai desembocar na frente da chácara. // ***alternativo 1***
>
> Se você for comprar o pinhão no caminho, logo depois de fazer o retorno da BR tem uma venda. Se estiver fechada, um pouco mais a frente, tem um senhor da chácara Pinhais que também vende. Se não encontrar pinhão, não tem problema. // ***alternativo 2***

Fluxos documentam as responsabilidades, ou seja, como as responsabilidades especificadas nos casos de uso são divididas entre sistema e atores. No desenrolar do projeto, as responsabilidades atribuídas ao sistema devem ser distribuídas entre os objetos que compõem o sistema.

Nas fases iniciais de análise é bom concentrar-se nos fluxos básicos (cerca de 80% do tempo de execução de um sistema é ocupado pelos casos primários) e somente identificar os casos secundários.

Fluxo Básico

Um fluxo básico representa o que ocorre normalmente quando o caso de uso é executado. A descrição do fluxo básico deve conter (Bittner e Spencer, 2003):

◊ ator e o evento que o mesmo dispara para iniciar o caso;

◊ a interação normal (sem tratamento de exceções) entre ator e sistema;

◊ descrição de como o caso termina.

Exemplo: considere um sistema onde o cliente realiza compras on-line num site utilizando um carrinho de compras virtual. O projetista do sistema previu um caso chamado buscar produtos e fazer pedido especificado pelo fluxo básico seguinte - extraído de (Bittner e Spencer, 2003):

> **NOME**: Buscar produtos e fazer pedido
>
> **DESCRIÇÃO**: Este caso descreve como um cliente usa o sistema para visualizar e comprar produtos disponíveis. Para encontrar um produto, o cliente pode pesquisar o catálogo por tipo de produto, fabricante ou por palavras-chaves.
>
> **PRÉ-CONDIÇÕES**: o cliente está logado no sistema.
>
> **PÓS-CONDIÇÕES**: o cliente realiza uma compra ou não.
>
> **FLUXO BÁSICO DE EVENTOS**

O caso de uso inicia quando o ator cliente escolhe a opção de consultar o catálogo de produtos.

{Mostrar catálogo de produtos}

O sistema mostra os produtos oferecidos, ressaltando os produtos cujas categorias constam no perfil do cliente.

{Escolher produto}

O cliente escolhe um produto a ser comprado e define a quantidade desejada.

Para cada produto selecionado disponível em estoque, o sistema registra o código do produto e a quantidade solicitada reservando-a no estoque e adiciona-o ao carrinho de compras.

{Produto esgotado}

Os passos 3 e 4 se repetem até que o cliente decida efetuar a compra dos produtos.

{Processar pedido}

10.O sistema pergunta ao cliente se deseja fornecer informações sobre o pagamento.

11.O sistema utiliza um protocolo seguro para obter as informações de pagamento do cliente.

12.Executar subfluxo validar informações de pagamento

13.{Informações de pagamento não válidas}

14.O sistema pergunta ao cliente se deseja fornecer informações sobre o envio das mercadorias.

15.O sistema utiliza um protocolo seguro para obter as informações de envio.

16.Executar subfluxo validar informações de envio.

17.{Informações de envio não válidas}

18.Executar subfluxo efetuar transação financeira.

19.O sistema pergunta ao cliente se deseja comprar mais produtos.

20.Se o cliente desejar comprar mais produtos, retomar o caso no ponto {Mostrar catálogo de produtos}, se não o caso termina.

No fluxo básico acima, pode-se notar a existência de vários elementos que serão descritos a seguir: subfluxo, pontos de extensão e fluxos alternativos.

Subfluxo

Um fluxo de eventos pode ser decomposto em subfluxos para melhorar a legibilidade. No entanto, é interessante evitar muitas decomposições, pois o fluxo ficará muito fragmentado e seu entendimento dificultado. Um subfluxo deve ser atômico, isto é, ou é executado na sua totalidade ou não é executado. Para referenciar um subfluxo a partir de outro fluxo usar a notação: executar <nome subfluxo>. No exemplo buscar produtos e fazer pedido, os subfluxos seguintes são encontrados:

◊ S1 Validar informações de pagamento;

◊ S2 Validar informações de envio;

◊ S3 Efetuar transação financeira.

Estes subfluxos podem ser detalhados da mesma maneira que um fluxo básico, porém deve-se evitar muitas decomposições sob o risco de perder a visão geral do caso de uso.

Pontos de extensão

São pontos precisos num fluxo de eventos que servem para inserir comportamentos adicionais. Pontos de extensão podem ser privados ou públicos. São privados se visíveis somente dentro do caso onde foram definidos ou públicos se visíveis nos casos que estendem o caso onde foram definidos.

No exemplo Buscar produtos e fazer pedido, os pontos de extensão seguintes são encontrados:

◊ {Mostrar catálogo de produtos}

◊ {Escolher produto}

◊ {Produto esgotado}

◊ {Processar pedido}

◊ {Informações de pagamento não válidas}

◊ {Informações de envio não válidas}

Um ponto de extensão pode definir

◊ uma localização único dentro do fluxo, por exemplo, {Mostrar catálogo de produtos}, {Escolher produtos} e {Processar pedido};

◊ um conjunto de localizações que representam um certo estado do caso de uso, por exemplo,

{Produto esgotado} que poderia aparecer em vários pontos do fluxo de eventos;

◊ uma região entre dois pontos de extensão, por exemplo, {Escolher produtos} e {Processar pedido}.

Fluxo Alternativo

Um fluxo alternativo apresenta um comportamento opcional, de outra forma, que não é parte do comportamento normal de um caso de uso. Fluxos alternativos são utilizados para representar tratamento de exceções ou um comportamento alternativo complexo que tornaria o fluxo básico muito longo ou de difícil compreensão.

Fluxos alternativos sempre são dependentes da existência de uma condição que ocorre em um ponto de extensão de outro fluxo de eventos. Há três tipos de fluxos alternativos:

Específico: iniciam num ponto de extensão.

Regional: podem ocorrer entre dois pontos de extensão.

Geral: podem ocorrer em qualquer ponto do caso de uso.

No exemplo Buscar produtos e fazer pedido, os fluxos alternativos seguintes são encontrados: Tratar produto esgotado (específico) e pesquisar por palavras-chaves (regional).

A2 Tratar produto esgotado

Em **{Produto esgotado}** quando não há a quantidade requisitada do produto em estoque.

1. O sistema informa que o pedido não pode ser completamente satisfeito.
2. *// a descrição deste fluxo continua com oferta de quantidades e produtos alternativos ao cliente*
3. O fluxo de eventos básico é retomado no ponto onde foi interrompido.

A1 Pesquisar por palavras-chaves

Entre **{Mostrar catálogo de produtos} e {Escolher produto}** quando o cliente escolhe realizar uma pesquisa por palavras-chaves.

1. O sistema pergunta ao cliente pelos critérios de busca do produto.
2. O cliente fornece os critérios de busca de produto.
3. *// a descrição deste fluxo continua*
4. O fluxo de eventos básico é retomado em **{Escolher produto}**.

Não há fluxo geral para o exemplo, mas poderia ser definido da seguinte maneira: em qualquer ponto do caso de uso Buscar produtos e fazer pedido...

Por que representar um fluxo alternativo separadamente do fluxo básico se é possível representá-lo com um se (if) no fluxo básico?

◊ Fluxos alternativos são opcionais e descrevem comportamentos que estão fora do comportamento normal esperado.

◊ Nem todos os fluxos alternativos representam funcionalidades essenciais, muitos deles podem não ser necessários, podem ser muito caros ou não prover funcionalidades importantes o suficiente para dispêndio de esforços de desenvolvimento.

◊ Fluxos alternativos permitem adicionar funcionalidade ao fluxo básico de maneira incremental ou remover funcionalidade à medida que tempo e dinheiro se esgotam.

Por exemplo, qual a importância de realizar pesquisas por palavras-chaves no exemplo em uso? Se for apenas uma das alternativas de busca não inviabiliza a funcionalidade do fluxo básico como um todo.

Agora se alguém perguntar qual a importância do fluxo básico buscar produtos e realizar pedido, é fácil ver que não pode ser deixado de fora.

Diagrama de Atividade

Um diagrama de atividade pode ser empregado para representar os fluxos de eventos de um caso de uso.

Sua utilização não suprime a descrição textual, pelo contrário, ele deve ser visto como uma ilustração simplificada da descrição textual.

Se todos os detalhes da descrição textual forem colocados no diagrama, este ficará extremamente poluído e perderá sua utilidade: tornar o caso de uso mais compreensível aos leitores.

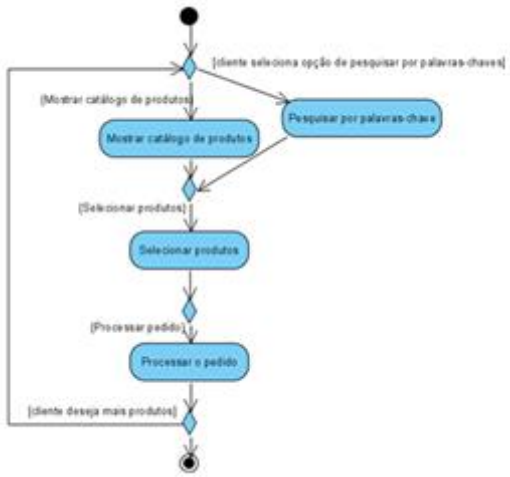

Figura 13: Exemplo de diagrama de atividades.

Cenários

Cenários são instâncias de execução dos casos de uso. Os fluxos alternativos representam as possibilidades de execução de um caso de uso.

No exemplo buscar produtos e realizar pedido, o fluxo alternativo pesquisar produtos por palavras-chaves é uma alternativa à simples visualização do catálogo de produtos, logo há pelo menos dois caminhos possíveis de execução.

Um cenário representa um desses caminhos (figura 14).

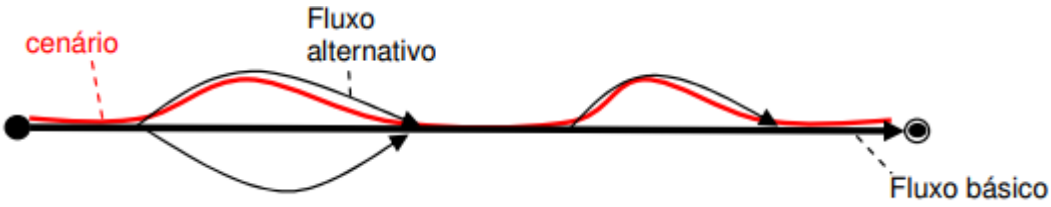

Figura 14: Representação esquemática de um cenário.

Cenários são importantes para definir casos de teste e para desenvolvedores pensarem sobre como o sistema será utilizado. Podem ser documentada adicionando-se informação às descrições dos casos de uso ou como parte da descrição dos testes. Não há necessidade de descrevê-los detalhadamente, basta nomeá-los e descrever o caminho a ser percorrido (por exemplo, fluxo básico, fluxo alternativo a1, fluxo básico).

Realizações de Casos de Uso

Um caso de uso pode ser realizado (projetado e implementado) de diferentes modos. Em UML há uma representação para realização de caso de uso como ilustra a figura 15. O intuito dessa representação é fazer uma ponte entre as descrições do sistema utilizadas pelas pessoas envolvidas na sua construção, mas que não participam do desenvolvimento em si, e as descrições do sistema utilizadas pela equipe de desenvolvimento.

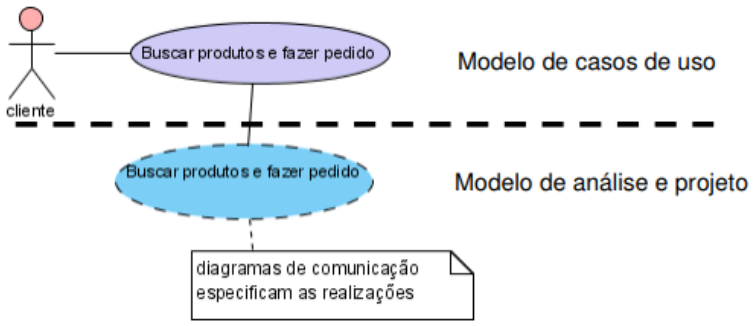

Figura 15: Realização de um caso de uso.

Diagramas de interação podem ser associados às realizações de casos de uso para especificar o fluxo de informações entre objetos que concretizam o caso. Porém, a representação de realização de caso não é muito utilizada.

Relações

Há vários tipos de relações possíveis num diagrama de casos de uso, porém é importante salientar que as relações:

não representam a ordem de execução dos casos;

devem melhorar a compreensão do que o sistema deve fazer (e não como projetá-lo). Em seguida, apresentam-se as relações mais comuns.

Associação

Associação é o tipo mais comum de relação. Pode ser utilizada entre dois atores ou entre um ator e um caso de uso. São representadas por uma linha cheia, com ou sem direção.

Ator x Ator

Relações associativas podem conectar atores para representar comunicação entre atores. A relação pode receber um nome que identifica o conteúdo da mensagem, documento ou objeto que trafega entre os atores. A figura 16 mostra uma associação entre o ator usuário de biblioteca que passa o

livro ao atendente que realiza o empréstimo ou a devolução. Como não há flechas, assume-se que o atendente devolve algo ao usuário da biblioteca, provavelmente um comprovante não representado no diagrama. Não é recomendável colocar este tipo de relação no diagrama de casos de uso.

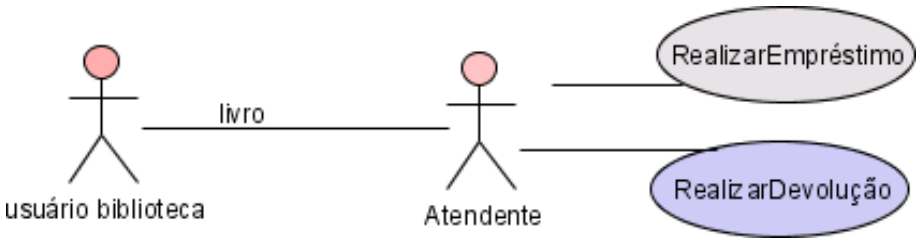

Figura 16: Exemplo de associação entre atores.

Ator x Caso

Há vários usos para associações entre atores e casos de uso:

Indica quem inicia a comunicação, o ator ou o caso de uso (sistema);

Indica o fluxo de informações, ou seja, quem fornece informações a quem.

Para documentar a escolha, pode-se atribuir um nome à associação. Na figura 16, há uma associação entre o atendente e o caso de uso realizar empréstimo. Observar que é bidirecional, portanto, o atendente inicia a execução do caso, fornece e recebe informações do mesmo.

Associações unidirecionais deixam os diagramas mais claros, embora não sejam obrigatórias. Por exemplo, a figura 17 ilustra um mesmo diagrama que representa os requisitos de um sistema de telefonia com relações bidirecionais e unidirecionais. No diagrama superior, pode-se deduzir que o emissor inicia a chamada telefônica e, no inferior, esta informação está explícita.

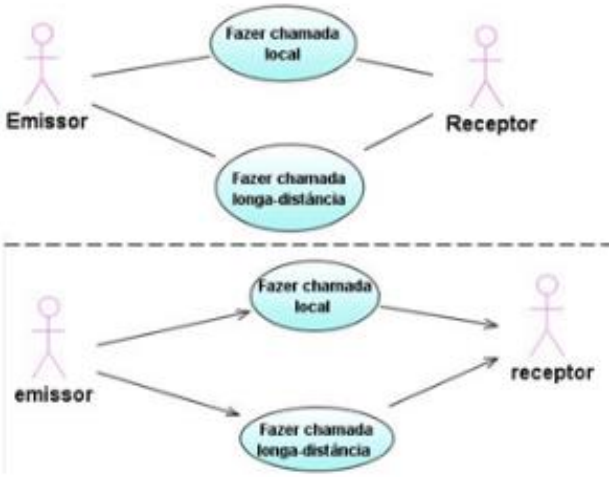

Figura 17: Associações bidirecionais e unidirecionais.

Inclusão

A relação de inclusão é utilizada entre dois casos, quando um deles inclui o outro na sua execução (subcaso). Um subcaso representa parte de um fluxo de eventos básico, isto é, um subfluxo que foi separado e representa um conjunto de ações atômico.

Ressalta-se que a existência de um subfluxo na descrição textual de um caso não implica sua representação no diagrama de casos de uso. A relação de inclusão (<<include>>) deve ser utilizada somente quando dois ou mais casos de uso apresentam partes idênticas nos seus fluxos de evento, caso contrário (se somente um caso de uso utilizar o subfluxo) não deve ser representado. Isto requer que os fluxos de evento sejam escritos antes de se colocar relações de inclusão no diagrama.

A figura 18 mostra um caso efetuar matrícula que possui subfluxos escolher disciplinas, alocar alunos às turmas e emitir boleto. Este último é compartilhado com o caso Efetuar inscrição curso opcional e, portanto, deve ser representado no diagrama. Um erro comum que adiciona complexidade ao diagrama é incluir os subfluxos escolher disciplinas e alocar alunos às turmas no diagrama.

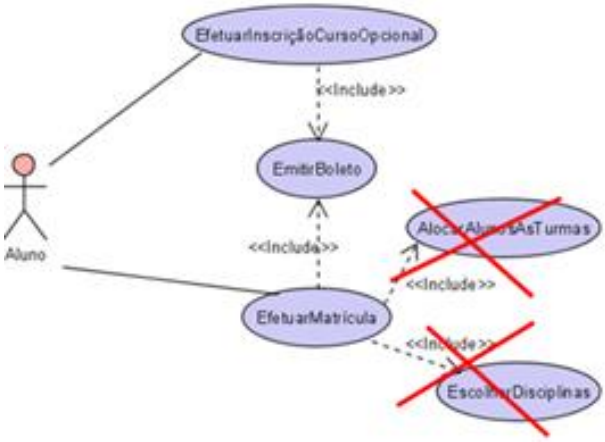

Figura 18: exemplo de inclusão de casos.

É importante ressaltar que:

◊ um caso de uso nunca deve ser incluído apenas por um caso, ou seja, não utilizar <<include>> para decompor o diagrama em partes;

◊ um caso de uso que é incluído por vários outros não tem conhecimento sobre quem o inclui, portanto, podem ser incluídos por qualquer caso sem sofrer modificações;

◊ não utilizar a relação de inclusão para representar opções de menu, pois o caso que faz a inclusão seria um simples despachante, todo o comportamento estaria fragmentado nos casos incluídos.

Enfim, inclusão deve ser utilizada para administrar comportamentos comuns e não para estruturar funcionalmente o diagrama.

Extensão

Um caso pode estender outro quando se deseja inserir um comportamento opcional ou excepcional disparado por alguma condição (ex. um alarme ou condição especial de algum objeto). Situações que podem levar ao uso da relação de extensão (Bittner e Spencer, 2003):

◊ Descrições opcionais ao comportamento normal do sistema: por exemplo, módulos que podem ser comprados do desenvolvedor ou de terceiros.

◊ Descrições de tratamento de erros e exceções complexos: estes tratamentos podem ser extremamente longos e ofuscar o fluxo básico.

◊ Customização: fluxos alternativos que especificam como diferentes clientes tratam certas situações no dentro do mesmo caso de uso base.

◊ Administração de escopo e de release: comportamentos que serão incluídos futuramente. É importante ressaltar que:

◊ Um caso de uso de extensão não requer modificações no caso base (aquele que é estendido). O comportamento básico do caso base permanece intacto.

◊ Um caso de uso que estende um caso base conhece este último (não é muito comum um caso de uso estender mais de um caso base).

◊ Uma extensão nasce como um fluxo alternativo, mas nem todo fluxo alternativo vira uma extensão.

◊ Casos de uso que estendem assumem o controle no ponto de extensão e quando terminam devolvem o controle no mesmo ponto.

Aqui cabe uma distinção entre fluxos alternativos e casos de uso que estendem outros. Fluxos alternativos são parte do caso de uso base e têm acesso ao seu estado, pré-condições, outros fluxos existentes e pontos de extensão além daquele onde se inserem. Casos de uso que estendem conhecem apenas o ponto de extensão onde se inserem no caso estendido. Para saber se um fluxo alternativo é candidato a ser uma extensão, deve responder positivamente à questão: o sistema pode ser entregue sem a extensão?

Na figura 19, a emissão de histórico escolar é estendida pelo caso imprimir comprovante de término quando o aluno solicitante for formado. Observa-se que é um comportamento opcional que pode não ser oferecido sem prejuízo ao comportamento básico emitir histórico escolar.

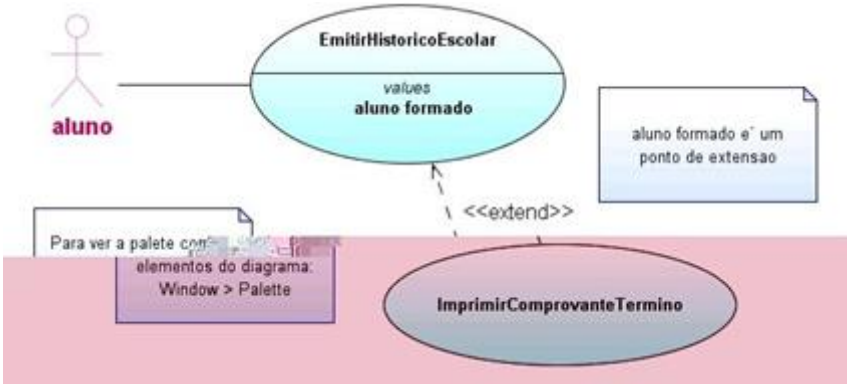

Figura 19: exemplo de relação de extensão entre casos de uso.

Nota-se na o ponto de extensão público denominado {aluno formado} onde o comportamento opcional imprimir comprovante término é inserido. É provável que existam outros pontos de extensão privados definidos nos fluxos de emitir histórico escolar, porém, no diagrama só os usados pelas extensões são listados. A figura 20 ilustra o diagrama de casos de uso para o exemplo buscar produto e fazer pedido.

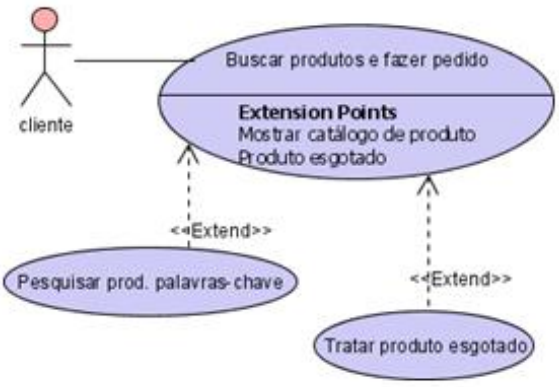

Figura 20: pontos de extensão para o caso buscar produtos e fazer pedido.

Generalização/Especialização

A relação de generalização/especialização pode ocorrer entre casos de uso ou entre atores.

Caso x Caso

Generalização permite especificar comportamentos genéricos que podem ser especializados para atenderem necessidades específicas. Normalmente é utilizado quando se quer descrever famílias de sistemas.

Por exemplo, uma empresa que desenvolve software pare terminais bancários de auto-atendimento quer expandir seus negócios para outras áreas, tais como pagamento direto em bombas de gasolina.

NOME: Realizar transação **(caso abstrato)**

DESCRIÇÃO: Permite ao usuário comprar mercadorias de um terminal automático sendo que o valor das mercadorias descontado de uma conta bancária.

PRÉ-CONDIÇÕES: (e) o cliente possui um cartão bancário; a conexão com o banco está ativa; o terminal deve ter mercadoria.

PÓS-CONDIÇÕES: (ou) O terminal retornou o cartão bancário ao cliente, entregou a mercadoria ao cliente e debitou o valor de sua conta. O terminal retornou o cartão bancário ao cliente, não entregou nenhuma mercadoria e nenhum valor foi debitado da sua conta.

FLUXO BÁSICO DE EVENTOS

1. O ator cliente insere o cartão bancário no terminal.
2. O sistema lê as informações da conta do cliente no cartão bancário
3. O sistema solicita ao cliente a senha
4. O cliente fornece a senha
5. O sistema verifica se a senha fornecida pelo cliente é idêntica à lida do cartão bancário
6. O sistema contata com o Sistema Bancário para verificar se as informações da conta do cliente são válidas
7. O sistema solicita o valor da transação
8. O sistema contata o Sistema Bancário para verificar se o cliente tem saldo para cobrir a solicitação.

{Cliente realiza a transação}

9. O sistema registra o valor da transação.
10. O sistema comunica ao Sistema Bancário que a transação foi efetuada.
11. O sistema grava no log os dados da transação: data, hora, valor e conta.
12. Término do caso de uso.

NOME: Sacar **(caso concreto)**

DESCRIÇÃO: Especializa o caso de uso realizar transação para permitir ao cliente retirar dinheiro de um terminal de auto-atendimento bancário.

PRÉ-CONDIÇÕES: (e) o cliente possui um cartão bancário; a conexão com o banco está ativa; o terminal deve ter dinheiro.

PÓS-CONDIÇÕES: (ou) O terminal retornou o cartão bancário ao cliente, entregou o dinheiro ao cliente e debitou o valor de sua conta. O terminal retornou o cartão bancário ao cliente, não entregou dinheiro e nenhum valor foi debitado da sua conta.

FLUXO BÁSICO DE EVENTOS

Em {Cliente realiza transação}

1. O sistema verifica se tem dinheiro suficiente em relação ao montante solicitado pelo cliente.
2. O sistema entrega o montante solicitado.
3. O sistema solicita que retire o dinheiro do terminal.
4. O cliente pega o dinheiro.

5. Retoma-se o caso de uso abstrato em **{Cliente realiza transação}**

NOME: Abastecer veículo **(caso concreto)**

DESCRIÇÃO: Especializa o caso de uso *realizar transação* para permitir ao cliente obter combustível de uma bomba debitando o valor de sua conta.

PRÉ-CONDIÇÕES: (e) o cliente possui um cartão bancário; a conexão com o banco está ativa; a bomba tem combustível.

PÓS-CONDIÇÕES: (ou) O terminal retornou o cartão bancário ao cliente, liberou o combustível ao cliente e debitou o valor de sua conta. O terminal retornou o cartão bancário ao cliente, não liberou combustível e nenhum valor foi debitado da sua conta.

FLUXO BÁSICO DE EVENTOS

Em {Cliente realiza transação}

1. O sistema solicita ao cliente para tirar o bico de abastecimento e libera o fornecimento de combustível.
2. O cliente enche o tanque até atingir o valor informado ou até que o tanque esteja cheio.
3. O cliente recoloca o bico na bomba.
4. Retoma-se o caso de uso abstrato em **{Cliente realiza transação}**

Diagrama de casos de uso correspondente aos casos acima é ilustrado na figura 21.

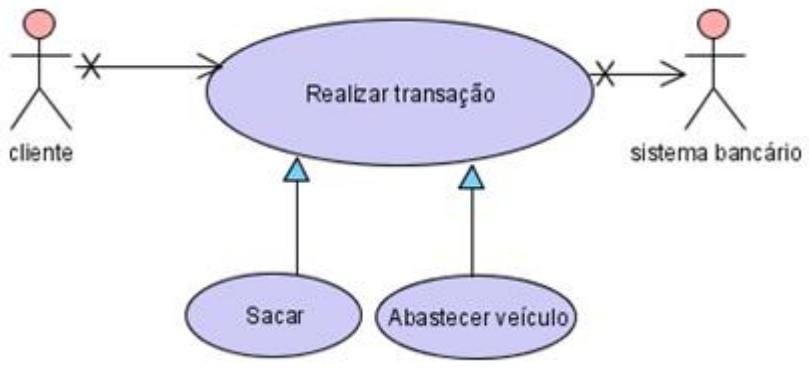

Figura 21: generalização/especialização de casos de uso.

Ressalta-se que nesta situação, são executados os casos de usos especializados. Eles apenas reusam partes do caso geral. A tabela 3 mostra as diferenças entre especialização e extensão de casos de uso.

Tabela 3: comparativo entre especialização e extensão de casos de uso.

Especialização	Extensão
O caso de uso especializado é executado	O caso de uso base é executado
caso de uso base não precisa ser completo e com sentido. Há várias lacunas preenchidas somente nas especializações.	O caso de uso base deve ser completo e com sentido.
comportamento de uma execução depende unicamente do caso específico.	comportamento de uma execução depende do caso de uso base e de todas as extensões que são executadas.

Ator x Ator

Especialização de atores representa que um conjunto deles possui responsabilidades ou características em comum. Algumas dicas para evitar modelagens desnecessárias:

◊ Não utilizar atores para representar permissões de acesso.

◊ Não utilizar atores para representar organogramas (hierarquias) de cargos de uma empresa.

◊ Utilizar atores somente para definir papéis em relação ao sistema.

Por exemplo, se num sistema de matrículas de uma universidade há casos de uso especiais para alunos de ciências exatas e para alunos de humanas, então é sinal que estes alunos são especializações de um ator genérico aluno. A figura 22 ilustra a notação UML para este caso. Observar que alunos de ciências exatas e de humanas herdam todas as associações do ator aluno.

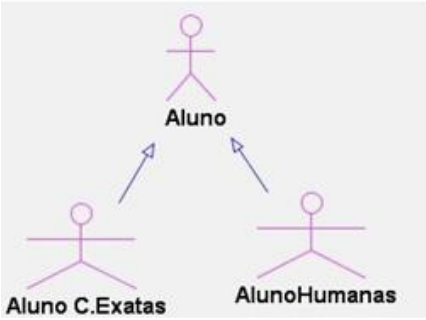

Figura 22: Especialização de atores.

Modelagem

Dicas

Casos de uso auxiliares

◊ Casos de uso auxiliares são frequentemente esquecidos, pois não são essenciais à funcionalidade do sistema. Porém, esquecê-los completamente pode conduzir a um sistema difícil de ser utilizado.

◊ Lembrar de colocar casos de uso para executar, manter e configurar o sistema, tais como: lançar e parar o sistema, incluir novos usuários, fazer backup das informações, incluir novos relatórios e realizar configurações.

Decomposição funcional

◊ Não é necessário detalhar em excesso os casos de uso. Muitos detalhes levam a decomposição dos casos em funções. O objetivo é compreender o sistema através de cenários de utilização.

◊ Casos de uso não são feitos para analisar (no sentido de decompor) os requisitos em requisitos menores. É um processo de síntese ou elaboração (e não de análise) no qual o problema não é totalmente conhecido. Quebrá-lo em partes menores (análise) dificulta a obtenção de uma visão geral.

◊ Em equipes com forte competência em análise estruturada, há tendência em encontrar funções ao invés de casos de uso. Por exemplo, fazer pedido, revisar pedido, cancelar pedido e atender pedido podem parecer bons casos de uso. No fundo, todas estas funções estão relacionadas ao caso de uso realizar pedido.

◊ Decomposição funcional pode levar a um número intratável de casos, mesmo para pequenos sistemas, e à perda de foco no que realmente é importante no sistema (o que traz valor aos atores).

◊ Casos de uso não chamam outros casos de uso ou se comunicam com outros casos.

Estrutura e detalhamento

◊ Não estruturar demais o diagrama de casos de uso, isto é, não inclua relações entre casos de uso a não ser que sejam extremamente necessárias. O uso em excesso destas relações pode fragmentar

diagrama, diminuindo a compreensão global.

◊ O modelo deve ser o mais simples e direto possível.

◊ Não descrever o que ocorre fora do sistema. Por exemplo, interação entre atores pode ser importante para o negócio, mas se o sistema não facilita esta interação, deixe-a fora. Se for necessário esclarecer estes pontos faça um modelo do negócio e não um modelo de casos de uso.

◊ Não fazer casos tais como incluir, consultar, alterar e excluir (ICAE). Casos de uso que descreve comportamentos ICAE não adicionam muito valor à descrição da funcionalidade do sistema. Para estes tipos de comportamentos, os requisitos são bastante claros e não se deve perder tempo especificando-os. A maior parte destes comportamentos tem um padrão mostrar campos, usuário entra com os dados, sistema valida, usuário corrige erros,... A validação, a parte mais importante dos comportamentos ICAE, pode ser capturada no domínio do modelo (por meio de relações, cardinalidade e restrições) ou textualmente no glossário de dados.

Modelo de casos de uso é mais que um diagrama

◊ O diagrama de casos de uso é uma visão panorâmica dos casos de uso e, portanto, apenas um dos componentes do modelo de casos de uso. A descrição textual dos mesmos é a parte mais importante do modelo. São elementos do modelo de casos de uso: glossário, modelo do domínio e diagramas de atividades.

◊ Um glossário e um modelo do domínio podem evitar o excesso de detalhes nas descrições dos casos de uso. Por exemplo, ao invés de descrever a validação da entrada de dados, os campos podem ser definidos no glossário com os respectivos valores possíveis.

◊ Um modelo do domínio ajuda a entender as relações existentes entre as entidades do domínio e as restrições sobre as relações.

Passos

Para elaborar um modelo de casos de uso, os seguintes passos podem ser seguidos:

◊ Recapitular a visão do sistema (estudo de viabilidade) aos envolvidos.

◊ Elaborar se necessário um diagrama de contexto para definir os limites do sistema (o que está dentro e o que está fora).

◊ Identificar os atores do sistema.

◊ Identificar os casos de uso: descrevê-los e rascunhar o fluxo de eventos. Não se preocupe com fluxos alternativos. Não gaste mais do que 10 minutos para descrever cada caso de uso.

◊ Esboçar o diagrama de casos de uso mostrando associações entre atores e casos de uso.

◊ Verificar os casos de uso:

Há casos de uso sem conexão com requisitos funcionais? Caso haja, pode haver casos em excesso ou requisitos que não foram identificados!

Há requisitos funcionais sem casos? Alguns casos podem ter sido esquecidos.

Todos os requisitos não-funcionais foram tratados (associados aos casos de uso quando são específicos)?

Todos os tipos de usuários foram mapeados para um ator ao menos?

◊ Descrever os casos detalhadamente

◊ Representar os subfluxos (<<include>>) e fluxos alternativos (<<extend>>) considerados importantes no diagrama

◊ Verificá-los:

É possível eliminar os casos incluídos ou as extensões e ainda ser capaz de entender o que o sistema faz para as partes interessadas?