Professor: Fabrício Varajão - varajao@gmail.com

Apostila

Revisado e atualizado em Jan/2013

ÍNDICE

INTRODUÇÃO

As metodologias para desenvolvimento de sistemas foram aceitas no meio tecnológico devido à necessidade de uma padronização do processo de análise e desenvolvimento. O rápido avanço tecnológico na parte de hardware demandou o desenvolvimento de softwares de grande porte, porém o porte intelectual dos desenvolvedores, até então, não estava preparado para isso. Com isso, nascem métodos para padronização e documentação destes sistemas que evoluíram com o passar do tempo, com as mudanças de padrões e necessidades de mercado.

O método pode ser definido como o caminho pelo qual se atinge um objetivo, ou seja, o caminho ordenado para chegar a um fim. Metodologia consiste na análise do estudo e avaliação dos vários métodos disponíveis pela aprovação das técnicas que serão aplicadas. Pode-se definir também como o estudo dos métodos, especialmente, os da ciência. No ramo da informática pode-se dizer que é uma pesquisa de métodos de programação e de exploração dos computadores e meios informáticos. A metodologia é o passo a passo para se chegar ao resultado desejado. Ela identifica as principais atividades a serem executadas e indica as pessoas envolvidas nas atividades e os papéis de cada uma. Geralmente descreve critérios de entrada saída e pontos de conferência para decisões. Já o método é uma abordagem técnica com um modelo para se realizar uma ou mais tarefas de uma metodologia. (OLIVEIRA, 1999)

Necessidade da metodologia nas organizações

Na primeira década do processamento de dados eletrônicos, muito dinheiro foi perdido, pois se acreditava que construir sistema era o mesmo que construir programas. Todo o desenvolvimento do sistema era com ênfase na resolução do problema proposto sem uma análise minuciosa do mesmo. Assim, era comum o desenvolvimento em três etapas: uma entrevista com o cliente, uma longa etapa de implementação e finalmente, uma longa etapa de alterações para implantação do sistema. O resultado disso, era a insatisfação do cliente e uma infinidade de críticas aos analistas de sistema. Com isso as empresas começaram a se preocupar com o processo de análise o que levou a criação de uma infinidade de métodos para desenvolvimento de sistemas. Inicialmente, surgiu o conceito de programação estruturada que eliminou as deficiências do método de programação linear, apelidado de código espaguete, e propôs uma abordagem estruturada para programas de computador. Essa nova forma de desenvolver sistemas, na década de 1980, promoveu avanços que padronizaram os métodos construtivos e difundiram uma base conceitual indispensável aos profissionais de informação. Durante mais de uma década, a metodologia estruturada incorporou banco de dados relacionais, ferramentas CASE, simplificações e melhorias. A programação orientada logo teve seus conceitos estabelecidos e passou a ser experimentada para testes nos meios acadêmicos. Alguns conceitos estruturados receberam modificações e tornaram-se híbridos. Com a rápida expansão da informática houve um aumento da demanda de profissionais para o desenvolvimento e manutenção de sistemas. Sem a preparação necessária, o resultado nem sempre atendia a necessidade do cliente. Na década de 90, com a expansão dos cursos superiores e o crescimento da oferta de mão de obra na área, o mercado passou a selecionar os profissionais minimizando o problema. (SILVA, 2003)

1960	**Programação Linear**
1965	Metodologia informal, a critério do analista.
1970	Técnicas de estrutura para arquivos.
1980	Metodologia estruturada Especificação do projeto. Ferramentas de software. Modelagem de dados. Prototipação
1985	Interface com o usuário Ferramentas de prototipação. Inicio das ferramentas CASE.
1990	Ferramentas de geração de código.
1994	Linguagem visual e orientada a objeto.
1996	Aprimoramento das ferramentas case
1998	Estudo aprimorado da metodologia orientada a objetos.
1990/atual	Avanços na metodologia Orientada a objetos.

Evolução das metodologias de desenvolvimento de sistemas. (OLIVEIRA, 1999, p. 25)

1 - O PAPEL DO ANALISTA E A HISTÓRIA DA ANÁLISE

1.1. Primeiros sistemas – até 70

• Aplicações não tinham grandes dimensões;
• Limitações das máquinas existentes;
• Análise sem métodos e formalismos;
• Praticamente a ferramenta era o fluxograma;
• Derivação da fase de análise para fase de projeto sem critérios.

1.2. Mais tarde – década de 70

• Conceito de Engenharia de Software surge em repulsa à crise de informática – 1968;
• Dijkstra escreve sobre a programação estruturada dando importância à complexidade dos sistemas;
• Codd descreve o modelo Relacional para banco de dados – 1978;
• Niklaus Wirth desenvolve o Pascal;
• A linguagem C e desenvolvida por Ritchie;
• A análise estruturada é popularizada por Tom Demarco;
• Passamos a conviver com Diagramas de Fluxo de Dados (DFD), Diagramas de Entidades e Relacionamento (DER) e outros naquilo que ficou conhecido como Análise Essencial.

A análise essencial foi importante para que o desenvolvimento de sistema fosse enxergado com mais esmero e com respeito a complexidade, levando a produtos de melhor qualidade. Excelentes sistemas foram desenvolvidos dentro da modelagem estruturada ou essencial.

Entretanto, uma questão continuava ainda sem uma solução adequada - a dificuldade em garantir a compatibilidade entre as fases de análise e projeto e desta para a de implementação. Alteração ou extensões dos modelos criados necessitam de grande esforço.

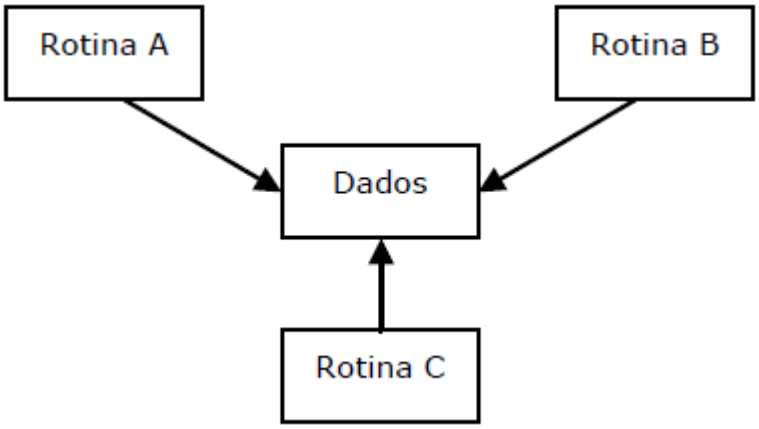
A comunicação entre desenvolvedores e usuários era difícil, os modelos fugiam à compreensão dos usuários. Validações não refletiam o universo dos requisitos do sistema. Outra questão: processos e dados eram vistos de forma independente.

Talvez estes problemas não fossem tão graves há 25 anos atrás: sistemas menores, com menos complexidade, ainda sem internet, usuários com pouca noção do que pedir,... Hoje se faz maior a necessidade de cumprir prazos, respeitar orçamentos, garantir a qualidade com menor custo e, se possível, oferecer um algo mais para o usuário.

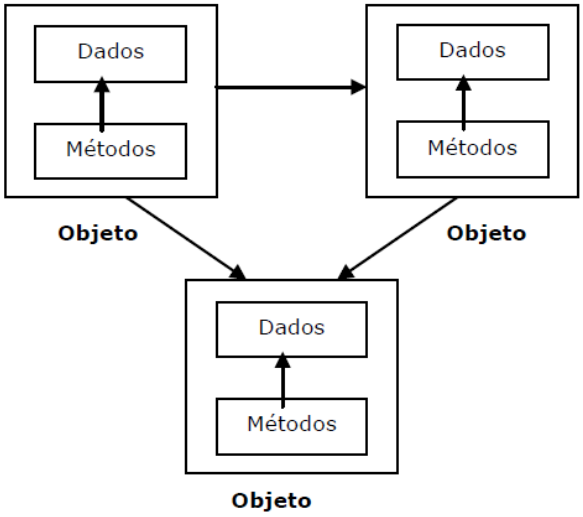
1.3. O início de um novo paradigma

Ainda na década de 70, métodos orientados a objetos começaram a surgir amadurecendo nova mudança de paradigma com o conceito de análise orientada a objetos. Chamou a atenção que neste novo cenário, os principais diagramas eram utilizados em todas as fases, mudando somente a visão deles. Neste conceito há a unificação da perspectiva funcional e de dados. E mais, melhorou a comunicação entre desenvolvedores e usuários, com estes conseguindo participar mais ativamente do desenvolvimento, pela análise e validação dos diagramas apresentados.

As linguagens tradicionais possuem foco nos programas. No paradigma estruturado – dados e processos – processos agem sobre os dados para alcançar o objetivo.



Nas linguagens orientadas a objeto os dados são protegidos por uma "cápsula" na qual residem procedimentos que mediam o acesso a eles. No paradigma da orientação a objetos, o objeto é uma unidade autônoma com seus dados sendo manipulados pelos processos definidos para o objeto. Cada objeto é responsável por realizar tarefas específicas e através da interação uma tarefa computacional é realizada.



2 - ANÁLISE DE SISTEMAS

2.1. Análise Estruturada

A análise estruturada é marcada pela construção de modelos que retratam o fluxo de informações utilizadas pelo sistema. Nela divide-se o sistema em camadas/partições funcionais e comportamentais e descreve-se a essência daquilo que será construído.

Ferramentas de Diagrama de Fluxo de Dados (DFD) são amplamente utilizadas como a abordagem preferida para a elaboração de um pré-projeto de *software*. O analista pode, ainda, criar modelos dos fluxos das informações através dos gráficos e com um dicionário de dados. O Diagrama de Fluxo de Dados é uma representação dos processos para uma macro visualização do sistema. Apresenta partes dos componentes do sistema e as interfaces entre eles. O dicionário de dados é um conjunto organizado das definições lógicas de todos os nomes de dados mostrados no DFD. A especificação de processos permite que o analista descreva a direção do negócio, representada por cada um dos processos de nível mais baixo dos diagramas detalhados do fluxo de dados (OLIVEIRA, 1999).

2.1.1. Modelo Ambiental

O modelo ambiental descreve o ambiente no qual o sistema se insere, ou seja, descreve o contexto do sistema, define a fronteira entre o sistema e o seu ambiente e respectivas interações.

Componentes do modelo ambiental:

- Definição de Objetivos: Finalidade de sistema;
- Lista de Eventos: Os acontecimentos que ocorrem no exterior e que interagem com o sistema;
- Lista de Respostas: As respostas aos "estímulos" que ocorrem no exterior e que o sistema deve responder.
- Lista de Entidades Externas: Entidades externas que geram fluxos de dados no sentido do sistema ou recebem fluxos de dados como respostas aos primeiros.
- Diagrama de Contexto: Representa o sistema como um único processo e as suas interações com o meio ambiente.

2.1.1.1. Definição de Objetivos

É uma declaração textual concisa e breve dos objetivos do sistema. Uma declaração mais detalhada deve ser deixada para quando do modelo comportamental.

Exemplo: “O propósito do Sistema de Processamento de Livros é manipular todos os detalhes dos pedidos de livros, bem como remessas, faturas e cobranças a clientes com faturas em atraso. Informações sobre pedidos de livros devem estar disponíveis para outros sistemas, tal como marketing, vendas e contabilidade. O propósito principal do sistema é reduzir o tempo necessário para processar um pedido de 3 para 1 dia".

2.1.1.2. Lista de Eventos

Consiste numa lista narrativa dos "estímulos" que ocorrem no exterior do sistema e aos quais o nosso sistema deve responder. Os eventos podem ser:

- Eventos Orientados por Fluxos
- Eventos Temporais

Exemplos:

- Cliente entrega pedido (F)
- Cliente cancela pedido (F)
- Direção necessita relatório de vendas semanalmente (T)
- Contabilidade precisa (mensalmente) do relatório de comissões de vendas (T)

2.1.1.3. Lista de Respostas

Consiste numa lista das respostas aos "estímulos" que ocorrem no exterior do sistema e aos quais o nosso sistema deve responder.

Exemplos:

- Fatura é enviada ao Cliente
- Relatório de vendas é enviado à Direção
- Relatório de comissões é enviado à Contabilidade

2.1.1.4. Lista de Entidades Externas

Consiste numa lista narrativa das entidades externas que geram fluxos de dados no sentido do sistema ou recebem fluxos de dados como respostas aos primeiros.

Exemplos:

- Fornecedor
- Direção
- Cliente
- Aluno

2.1.1.5 Diagrama de Contexto

É um caso especial de diagrama de fluxos de dados, no qual um único processo representa o sistema inteiro.

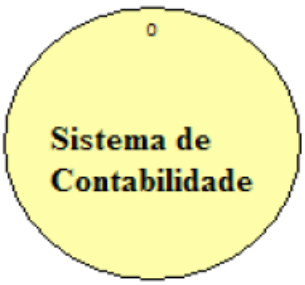
Componentes

- Processos
- Terminadores (Entidades Externas);
- Fluxos de dados;
- Um único processo que representa todo o sistema.

Processo

É a parte mais fácil do diagrama de contexto, consiste de um único círculo. O nome do processo é normalmente o nome do sistema.

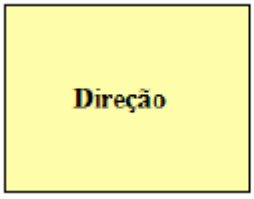
Exemplos

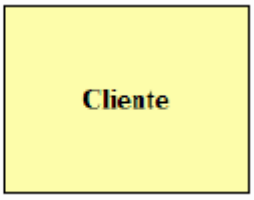


Terminadores

São representados por um retângulo. Os terminadores comunicam diretamente com o sistema através de fluxos de dados ou de fluxos de controle, ou através de depósitos de dados externos. Os terminadores não podem comunicar entre si.

Exemplos:



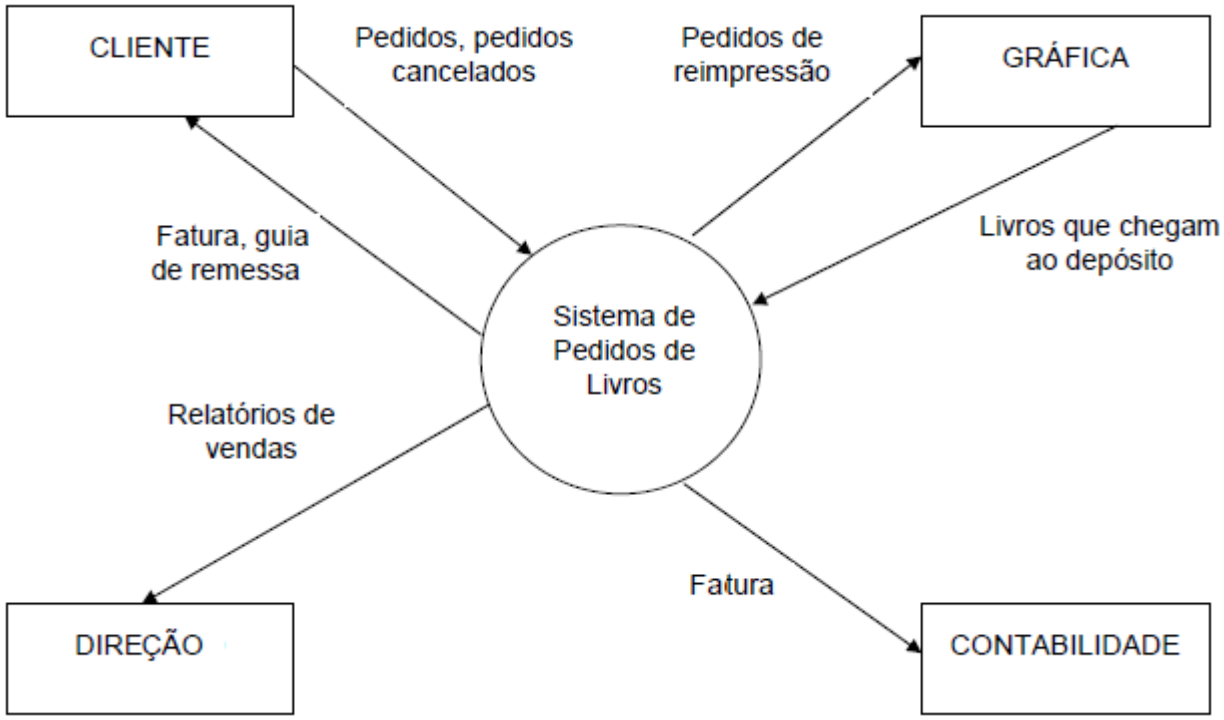


O diagrama de contexto deve ser construído de modo que as entradas sejam causadas e iniciadas pelos terminadores e que as saídas sejam causadas e iniciadas pelo sistema.

Fluxo de Dados

Um fluxo é graficamente representado por uma seta que entra ou sai de um sistema ou processo. O fluxo é uma seta utilizada para mostrar o movimento de fragmentos ou de pacotes de dados de um ponto para outro. Assim, o fluxo representa dados em movimento.

Exemplo: Diagrama de Contexto



Caso da Clínica Médica

Descrição do Sistema:

Considere os serviços prestados e os diferentes processos associados a uma clínica médica. Existe um conjunto de médicos que asseguram as diversas especialidades (oftalmologia, pediatria, dermatologia etc.). Cada médico assegura uma ou mais especialidades e tem consultório próprio. Cada consulta está associada a um determinado paciente e a um determinado médico.

Na primeira vez que um paciente se dirige à clínica para solicitar uma consulta tem de preencher na recepção um formulário de inscrição com os seus dados. Nas consultas posteriores, o paciente pode agendá-las por telefone ou presencialmente.

Após a consulta, os serviços administrativos passam a fatura de pagamento em função do perfil do paciente, que é confirmado com documentos comprovativos.

O processo de consulta termina com o pagamento feito pelo paciente contra emissão do respectivo recibo.

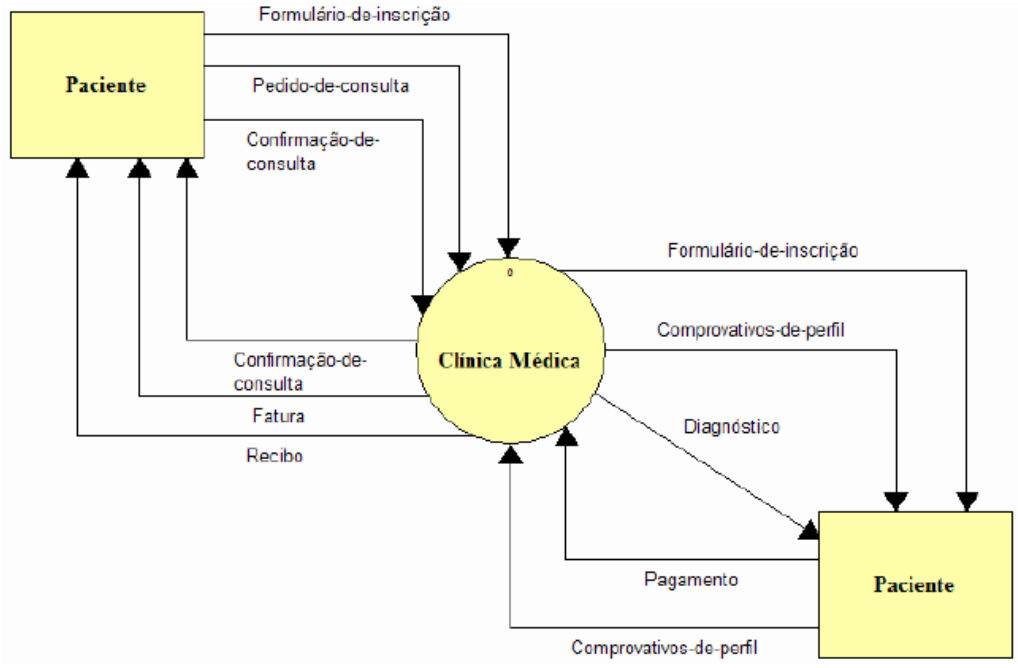
Periodicamente, no final de cada mês, efetuam-se diagnósticos que descrevem a evolução do tratamento e a situação atual de cada paciente da clínica. Estes são enviados por correio aos pacientes.

Lista de Eventos:

- Paciente solicita, cancela ou altera consulta (F)
- Paciente apresenta formulário de inscrição (F)
- Paciente apresenta-se a consulta (F)
- Paciente apresenta documentos comprovativos do seu perfil de apoio social ou de seguros de saúde (F)
- Paciente efetua pagamento (F)
- Final dos meses elaborar relatório para todos os pacientes (T)

Lista de Respostas

- Clínica entrega formulário de inscrição por preencher ao paciente
- Clínica confirma consulta ao paciente
- Clínica devolve comprovativos ao paciente
- Clínica entrega fatura ao paciente
- Clínica entrega recibo ao paciente
- Clínica envia diagnósticos aos pacientes



2.1.2. Modelo Comportamental

O modelo comportamental descreve as ações que o sistema deve realizar para responder da melhor forma aos eventos definidos no modelo ambiental. Descreve o comportamento do interior do sistema.

Técnicas utilizadas:

- Diagrama de Fluxos de Dados (DFD);
- Diagrama de Entidades e Relacionamentos (DER);
- Diagrama de Transição de Estados (DTE);
- Dicionário de Dados (DD);
- Especificação de Processos (EP).

2.1.2.1. Diagrama de Fluxo de Dados

Depois de modelado e validado o modelo ambiental é necessário passar para a modelação do comportamento do interior do sistema.

Geralmente o modelo comportamental segue uma pormenorização através de uma abordagem *top-down* (mas por vezes também pode haver a necessidade de uma generalização por meio de uma abordagem *bottom-up*).

A abordagem *top-down* envolve fundamentalmente a construção da primeira versão de um diagrama de fluxos de dados (DFD)

Desenha-se um processo, para cada evento da lista de eventos.

Os processos recebem um nome de acordo com a resposta que o sistema deve dar ao evento associado. (Ex: evento: cliente efetua pagamento-nome: atualizar contas a receberem vez de processar pagamentos de cliente porque não nos diz nada). Não devem ser associados processos a pessoas ou sistemas existentes.

Desenham-se entradas e saídas apropriadas de modo a que o processo seja capaz de emitir a resposta necessária e desenham-se depósitos de dados, como for mais adequado, para comunicação entre os processos.

O DFD resultante inicial é verificado em relação ao diagrama de contexto e à lista de eventos para que se confirme se está completo e consistente.

No DFD preliminar (0) não deve haver ligação entre processos porque eles representam respostas a eventos, sendo difícil que dois eventos ocorram no exterior simultaneamente. O que pode acontecer é que haja eventos interdependentes. Nesse caso o único modo de os sincronizar é através de um depósito de dados.

Como completar o modelo de processos?

O primeiro passo é reorganizar o DFD 0 ou preliminar que pode ser composto de vários processos. Então é necessário subdividir o DFD em níveis ascendentes. Existem três diretrizes a ter em consideração:

Cada agrupamento de processos deve envolver respostas estreitamente relacionadas.

Procurar oportunidades para ocultar dados armazenados que apareçam no nível inferior, quando há um grupo de processos no DFD preliminar relativo a um depósito, sem que outros processos se refiram a esse depósito.

Cada DFD deve conter no máximo 7 (+/- 2 processos) de modo a facilitar a sua leitura.

Subdivisão em níveis descendentes

Quando os processos identificados no DFD preliminar não são processos primitivos exigem subdivisão para baixo, em DFDs de níveis inferiores. Isto significa apenas que os processos iniciais, em que cada um dos quais é responsável pela produção da resposta a um evento, podem ser demasiadamente complexos para serem descritos numa especificação de processos.

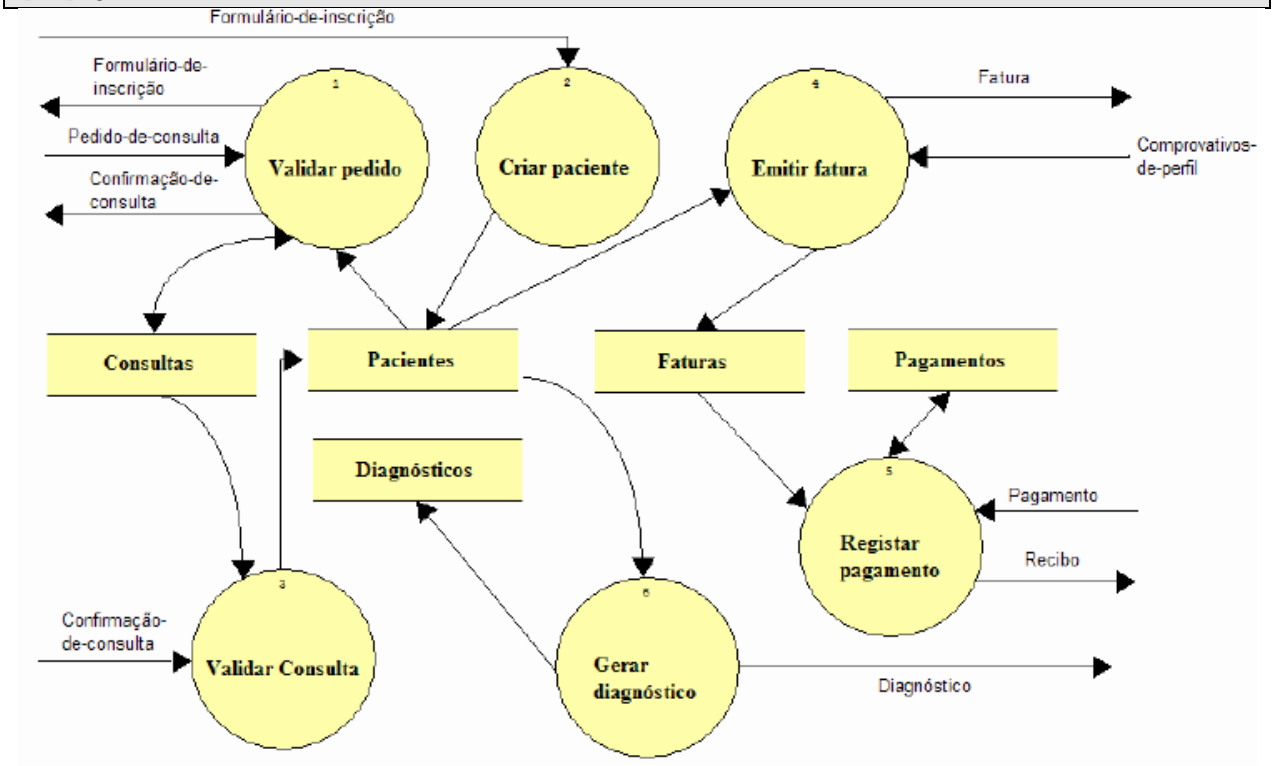
Em alguns casos, a abordagem de decomposição funcional pura é adequada. Isto é, se encontrar um processo complexo, tente identificar sub-funções, cada uma das quais podendo ser um processo de nível mais baixo.

Em outros casos, os fluxos de dados que chegam e que saem do processo darão melhor indicação para a subdivisão em níveis descendentes.

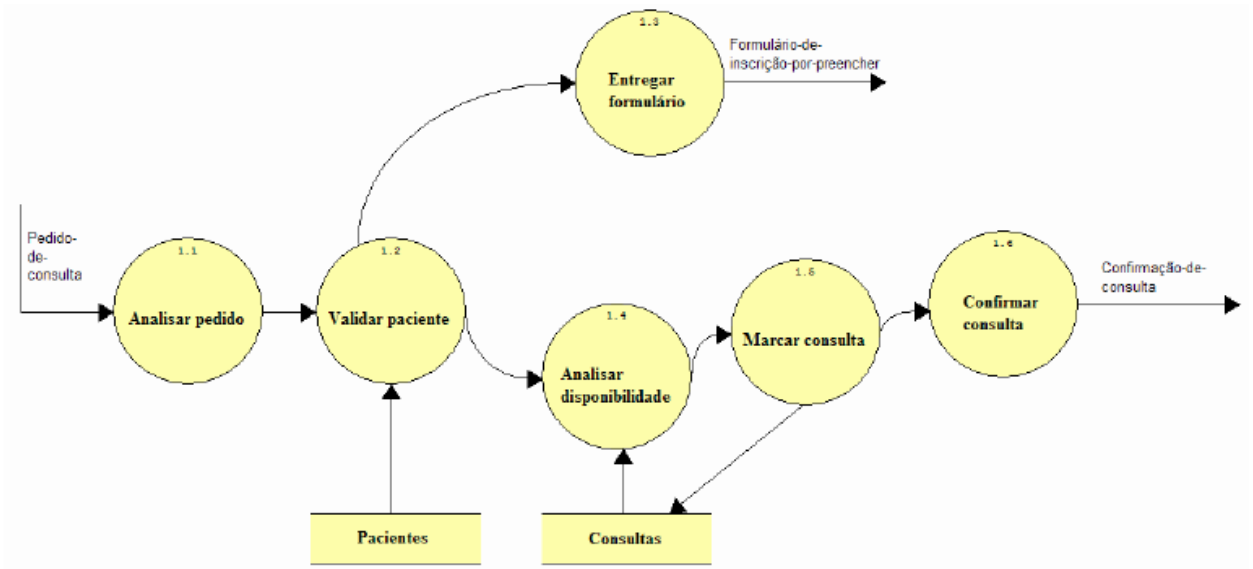
Enquanto estiver envolvido na atividade de subdividir os níveis de maneira ascendente ou descendente lembre-se da importância do equilíbrio. Isto é, é preciso verificar se as entradas e saídas de um processo de um determinado nível correspondem às entradas e saídas de um diagrama de nível imediatamente inferior.

Exemplo: Caso da Clínica Médica

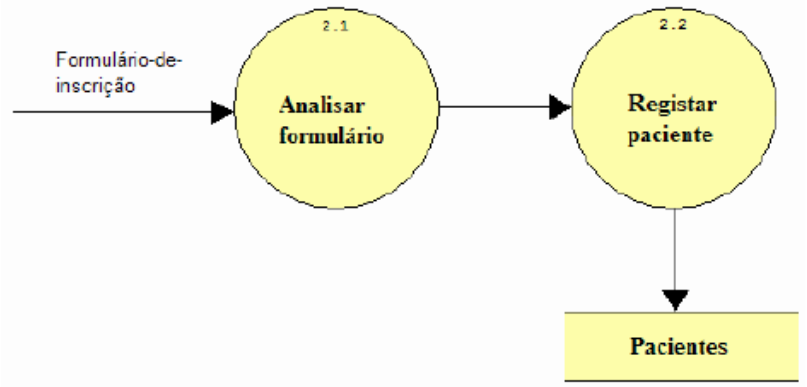
DFD 0



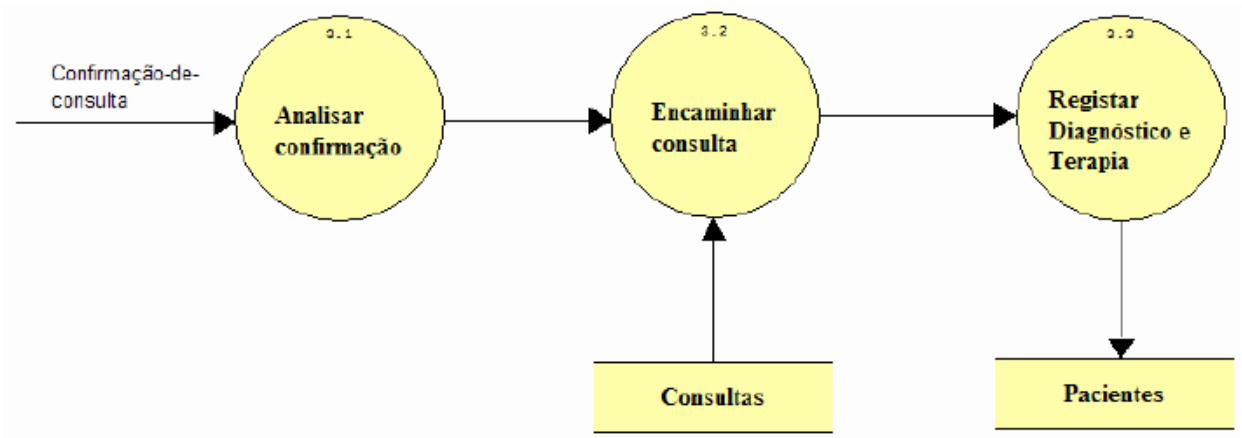
DFD 1



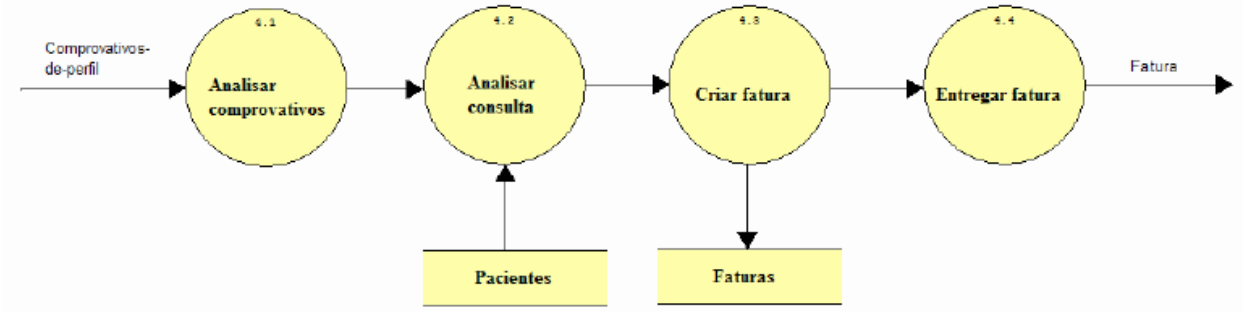
DFD 2



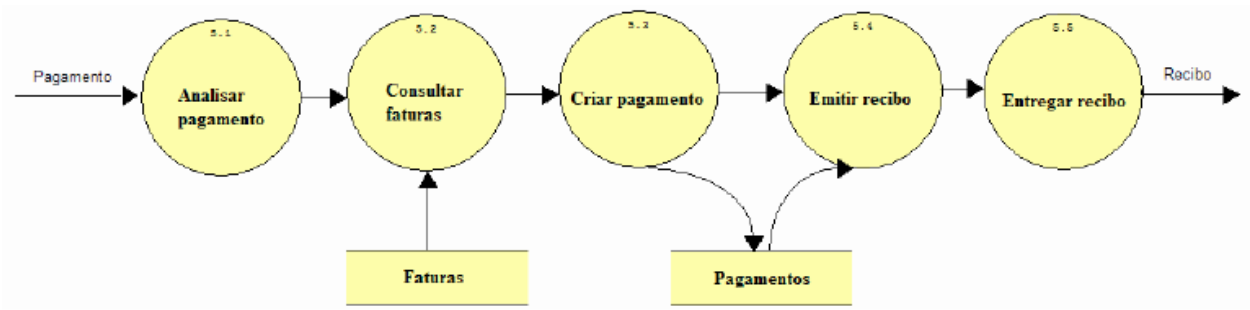
DFD 3



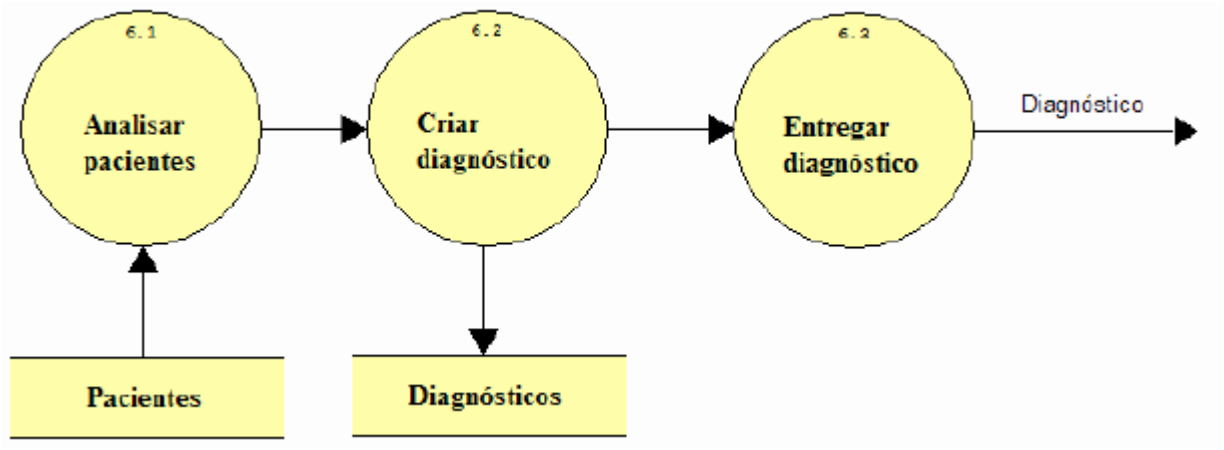
DFD 4



DFD 5



DFD 6



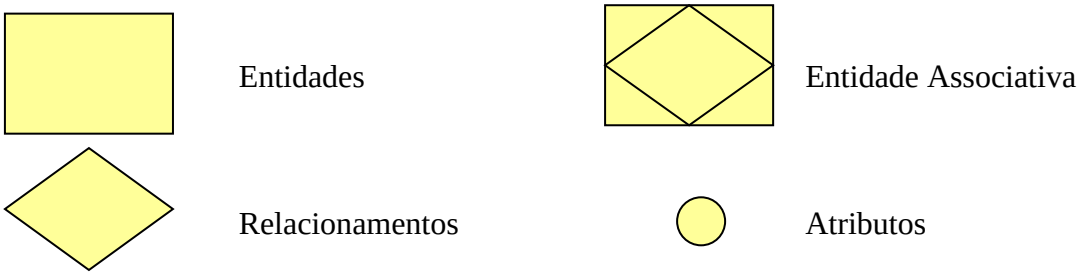
2.1.2.2. Diagrama de Entidades e Relacionamentos (DER)

Representa o modelo de dados do sistema.

Os elementos da modelagem conceitual de dados são:

- **Entidade**: conjunto de "coisas" que possuem características próprias;
- **Atributo**: características próprias de uma entidade;
- **Relacionamento**: vínculos entre entidades.

Simbologia básica:



2.1.2.3. Diagrama de Transição de Estados (DTE)

É uma técnica de modelação para descrever o comportamento do sistema (ou parte) dependente do tempo. Define as mudanças dinâmicas (de estado) que ocorrem na vida de uma entidade (ou sistema ou interface).

O DTE permite modelar:

- Os vários estados que a entidade pode ter (ESTADOS)
- As alterações de estado que pode sofrer (TRANSIÇÕES)
- As circunstâncias que levam a alteração de estado (CONDIÇÕES)
- As respostas a mudança de estado (AÇÕES)

O entendimento de diferentes estados, e as condições que provocam as mudanças de um estado para outro, representam módulos de programa (caso o objetivo seja construir software) que devem ser construídos para permitir a aplicação funcionar harmoniosamente com o mundo real. Por exemplo, a entidade "ordem-cliente" pode ter estados tais como, completa, devolvida, parcialmente satisfeita, aguarda entrega, em entrega, entregue, perdida, em atraso, em dívida, etc. Como os estados são posições naturais, e os eventos ou gatilhos ações sobre uma ocorrência de

uma entidade, as transições de um estado para outro representam os módulos de operação de um sistema.

O DTE mostra a sequência em que os eventos podem ocorrer e o efeito de eventos como uma função do estado do sistema. Os principais componentes de um DTE são os retângulos que representam os estados e as setas que representam as alterações de estado.

Estados

Um estado é uma situação em que o sistema se encontra e que pode durar por um determinado período de tempo.

Exemplos: Aguardando o próximo comando; Executando transação; Esperando a digitação da senha; Acelerando o motor; Em votação etc.

Em geral os estados apresentam situações em que o sistema aguarda pela ocorrência de um evento ou está fazendo algo.

Mudanças de Estado

São as transições de um estado para outro.

Indicam, para cada estado, os seus possíveis estados subsequentes.

Geralmente apontam os estados iniciais e finais.

O estado inicial normalmente é desenhado na parte de cima do diagrama. É identificado através de uma seta que lhe chega sem partir de outro estado.

Um estado final normalmente é desenhado na parte de baixo do diagrama e não possui setas que partem dele.

Um DTE pode ter vários estados finais.

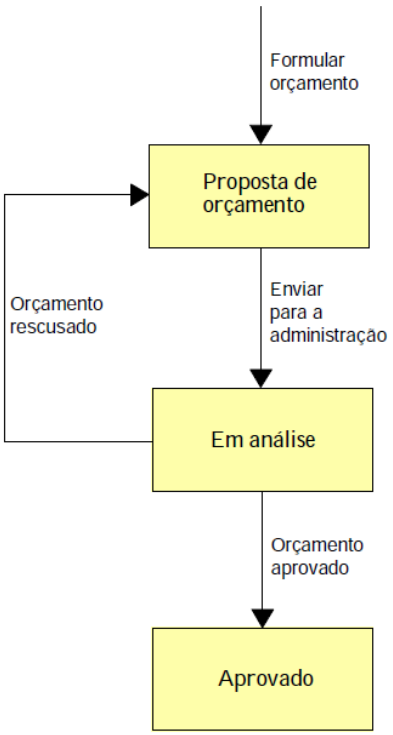
Condições e Ações

Num DTE também é possível incluir as condições que causam uma mudança de estado e as ações que o sistema empreende quando muda de estado.

São exibidas junto à seta que indica a mudança de estado (a condição acima e a ação abaixo, separadas por uma linha)

Exemplo de diagrama de transição de estados.

Orçamento



2.1.2.4. Dicionário de Dados (DD)

É uma listagem organizada de todos os elementos de dados pertinentes ao sistema, com definições precisas e rigorosas para que o utilizador e o analista de sistemas possam reconhecer todas as entradas, saídas, componentes de depósitos e cálculos intermédios.

- A forma narrativa é longa e sujeita a erros
- É necessário usar uma notação compacta e concisa

Elementos de dados são dados que não necessitam de decomposição.

Estrutura de dados são composições de elementos de dados e/ou de outras estruturas de dados.

A definição no DD é feita de forma *Top-Down*.

O dicionário de dados define os elementos de dados descrevendo:

O significado de **fluxos** e **depósitos**.

A composição de pacotes agregados de dados que se movimentam pelos **fluxos** (Ex: Endereço pode ser divido em itens elementares como cidade, estado etc.).

A composição dos pacotes de dados nos **depósitos**.

Os valores e unidades relevantes de partes elementares de informações dos **fluxos** e **depósitos**.

Os detalhes dos **relacionamentos** entre os **depósitos** realçados num DER.

Há vários esquemas de notação. Porém, o mais comum é o seguinte:

=	É composto de
+	E (concatenação)
( )	Opcional
{ }	Iteração
[ ]	Escolha de uma das opções alternativas
*	Delimitador de comentário
@	Identificador (campo chave) de um depósito
|	Separa opções alternativas na construção [ ]

Exemplo: definição de um nome (estrutura de dados)

nome =	* Nome completo do cliente * título-cortesia + primeiro-nome + (nome-intermédio) + último-nome
título-cortesia =	[Sr.|Srta.|Sra.|Sras.|Dr.|Professor]
primeiro-nome =	{carácter-válido}
nome-intermédio =	{carácter-válido}
último-nome =	{carácter-válido}
carácter-válido =	[A-Z|a-z|0-9|'|]

2.1.2.5. Especificação de Processos (EP)

Define o que deve ser feito para transformar entradas em saídas. É uma descrição detalhada mas concisa da realização de processos pelos utilizadores.

Quando os DFDs estiverem + ou - prontos deve-se começar a redigir as especificações de processos. Este trabalho é muitas vezes prolongado porque cada processo do nível mais baixo do DFD exige uma especificação.

Os passos lógicos de uma especificação de um processo podem ser mostrados em Linguagem Estruturada. Mas por vezes também podem ser usadas Condições Pré/Pós, Tabelas de Decisão e Árvores de Decisão como alternativas.

O vocabulário da **Linguagem Estruturada** consiste em:

- Verbos no imperativo (exemplos):
  - RECEBER (fluxo)
  - ENVIAR (fluxo)
  - MOVIMENTAR (dados)
  - ACEDER (depósito de dados)
- Palavras reservadas (estruturas de controlo)
- Termos definidos no Dicionário de Dados
- Termos locais
- Numerais e valores booleanos

Na linguagem estruturada, os passos lógicos são baseados nas construções base (primitivas) da programação estruturada: sequência, seleção e iteração.

- Sequência: uma ou mais sentenças executadas em sequência sem interrupção
- Seleção: SE e FAÇA-CASO
- Repetição: FAÇA-ENQUANTO e REPITA

Exemplo:

```
ACEITAR requisições-armazém
RECUPERAR balanço-stock EM Stock
SUBTRAIR quantidade A balanço-stock
SE balanço-stock < nível-reposição
     CRIAR requisição-compra
ESCREVER balanço-stock EM Stock
```

Condições Pré-Pós

São um modo prático de descrever um processo, sem que seja necessário detalhar o algoritmo que será utilizado.

Tem duas partes: pré-condições e pós-condições e é útil quando:

O utilizador tende a expressar o que é executada por um processo em termos de um algoritmo especial e particularizado (já conhecido).

O analista está razoavelmente seguro de que existem muitos algoritmos que podem ser usados.

O analista quer deixar que o programador explore alguns desses algoritmos.

Pré-condições

Definem o que deve ser verdade antes do início da execução do processo.

Podem ser consideradas como uma garantia do usuário.

Pós-condições

Descrevem o que deve ser verdadeiro quando o processo terminar.

Podem ser consideradas como uma garantia do sistema para o usuário.

Exemplos:

Pré-Condição 1

- O Cliente apresenta-se com um número-de-conta coincidente com um número de conta em Contas, cujo código-de-status está ajustado em "válido"

Pós-Condição 1

- A fatura é emitida contendo número-de-conta e valor-da-venda

Pré-Condição 2

- A pré-condição falha por algum motivo (o número-de-conta não é encontrado em Contas ou código-de-estados não é igual a válido

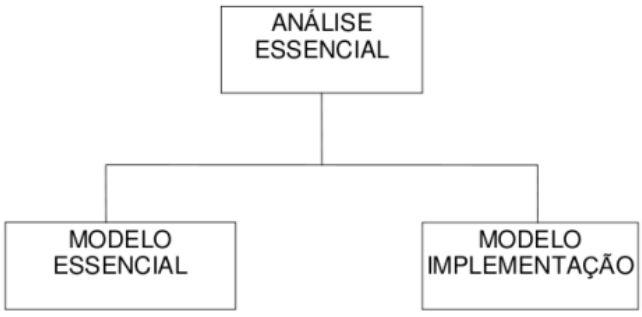
Pós-Condição 2

- É emitida uma mensagem de erro

2.2. Análise Essencial

A Análise Essencial propõe o particionamento do sistema por eventos. A rigor, o valor de um sistema está na sua capacidade de responder com eficácia a todos os estímulos a que for submetido. Assim, um sistema é construído para responder a estímulos. A cada estímulo, o sistema deve reagir produzindo uma resposta predeterminada.

No modelo essencial o analista deve cumprir os requisitos do cliente mencionando o mínimo possível sobre como o sistema será desenvolvido. Deve-se evitar a descrição dos detalhes específicos dos processos. No conceito de análise essencial, o sistema é visualizado em duas partes: os **dados** e as **funções**. O analista inicia a análise com a criação do "modelo essencial". Este apresenta um grau de abstração que não considera restrições tecnológicas. Posteriormente, será inserido o "modelo de implementação". Este é derivado do modelo essencial e contém especificações do sistema considerando restrições tecnológicas. O modelo essencial ainda é formado pelo "modelo ideal", que descreve os requisitos que o sistema irá atender sem considerar se isto será ou não incrementado no sistema. O "modelo ambiental" irá delimitar uma fronteira que divide o sistema e o mundo. O "modelo comportamental" define como irão funcionar os componentes internos uma vez que estes possuem interações diretas com o usuário. Neste modelo é utilizado o Diagrama de Fluxo de Dados para representação do sistema e o "dicionário de dados", um repositório de informações sobre os componentes do sistema. (POMPILHO, 1995)



2.2.1. Modelo Essencial

Apresenta o sistema num nível de abstração completamente independente de restrições tecnológicas. Antes que um sistema seja implementado, é necessário conhecer-se a sua verdadeira essência, não importando saber se sua implementação vai ser manual ou automatizada, e nem mesmo que tipo de hardware ou software vai ser usado. O Modelo Essencial é formado por:

- **Modelo Ambiental:** Define a fronteira entre o sistema e o resto do mundo
- **Modelo Comportamental:** Define o comportamento das partes internas do sistema necessário para interagir com o ambiente;

2.2.1.1. Modelo Ambiental

O Modelo Ambiental é o modelo que define a fronteira do sistema com o ambiente onde ele se situa, determinando o que é interno e o que é externo a ele; as interfaces entre o sistema e o ambiente externo, determinando que informações chegam ao sistema vindas do mundo exterior e vice-versa; os eventos do ambiente externo ao sistema aos quais este deve responder; ferramentas para definição do ambiente.

O Modelo Ambiental consiste de quatro componentes:

1. Declaração de Objetivos
2. Diagrama de Contexto
3. Lista de Eventos
4. Dicionário de Dados Preliminar (opcional)

Declaração dos Objetivos

Consiste de uma breve e concisa declaração dos objetivos do sistema.

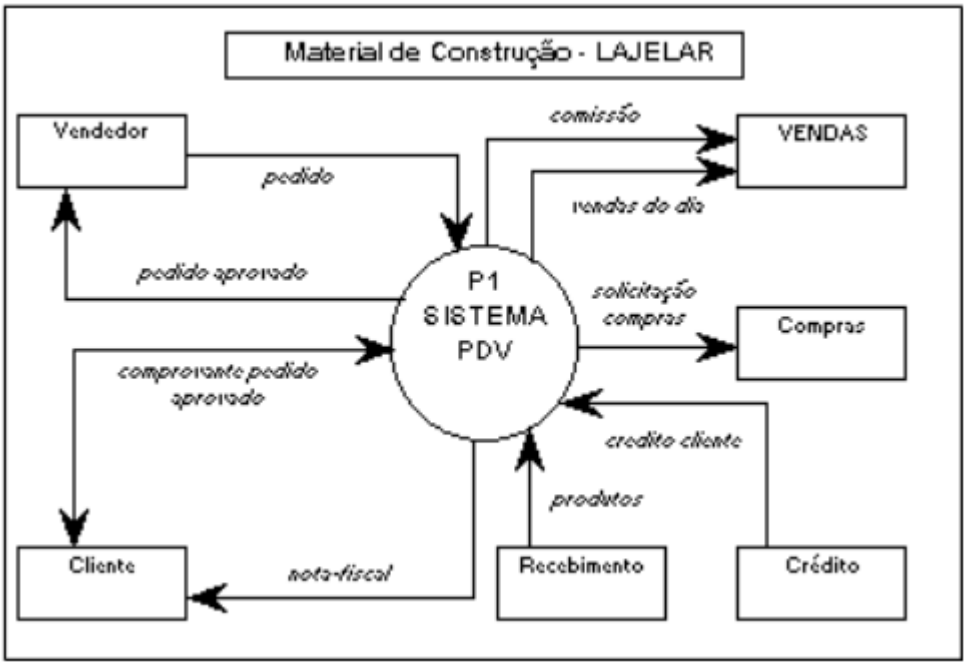
É dirigida para a alta gerência, gerência usuária ou outras pessoas não diretamente envolvidas no desenvolvimento do sistema.

Pode ter uma, duas ou várias sentenças, mas não deve ultrapassar um parágrafo.

Não deve pretender dar uma descrição detalhada do sistema.

Diagrama de Contexto

Apresenta uma visão geral das características importantes do sistema, as pessoas, organizações ou sistemas com os quais o sistema se comunica (Entidades Externas), os dados que o sistema recebe do mundo exterior e que de alguma forma devem ser processados, os dados produzidos pelo sistema e enviados ao mundo exterior e a fronteira entre o sistema e o resto do mundo.



Lista de eventos

É uma relação de estímulos que ocorrendo no mundo exterior implicam que o sistema de algum tipo de resposta.

Também pode ser definido informalmente como um acontecimento do mundo exterior que requer do sistema alguma resposta. É um ativador de uma função. É a forma como o evento age sobre o sistema. É a consequência do fato de ter ocorrido um evento externo. É a chegada de um estímulo que indica que o evento ocorreu e isto faz com que o sistema então ative uma função pré-determinada para produzir a resposta esperada.

UM ESTÍMULO: É um ativador de uma função. É a forma como o evento age sobre o sistema. É a consequência do fato de ter ocorrido um evento externo. É a chegada de um estímulo que indica que o evento ocorreu e isto faz com que o sistema então ative uma função pré-determinada para produzir a resposta esperada.

UMA RESPOSTA: É o resultado gerado pelo sistema devido à ocorrência de um evento. Uma resposta é sempre o resultado da execução de alguma função interna no sistema como consequência do reconhecimento pelo sistema de que um evento ocorreu.

2.2.1.2. Modelo Comportamental

Define o comportamento interno que o sistema deve ter para se relacionar adequadamente com o ambiente. Ou, o Modelo Comportamental é definido do ponto de vista interno, é o modelo interior do sistema. Descreve de que maneira o sistema, enquanto um conjunto de elementos inter-relacionados reage, internamente, como um todo organizado, aos estímulos do exterior.

Consiste de cinco componentes:

1. Diagrama de Fluxo de Dados (DFD) Particionado
2. Diagrama de Entidades e Relacionamentos (DER)
3. Diagrama de Transição de Estado (DTE)
4. Dicionário de Dados
5. Especificações de Processos (EP)

2.2.2. Modelo de Implementação

Tem como objetivo definir a forma de implementação do sistema em um ambiente técnico específico. Apresenta o sistema num nível de abstração completamente dependente de restrições tecnológicas.

Simbologia

Conjunto de artefatos gráficos que permitem a montagem de diagramas na análise essencial.

Processo: Conjunto de atividade que produzem, modificam ou atribuem qualidade às informações.

Depósito de Dados: Conjunto de informações armazenadas pelo processo para serem utilizadas por algum processo, a qualquer momento.

Entidade Externa: É algo situado fora do escopo do sistema, que é fonte ou destino das suas informações.

Fluxo de Dados: O nome deve expressar o significado do conjunto de informações que está fluindo.

2.3. Análise Orientada a Objetos

A análise Orientada a Objetos é a mais nova abordagem de análise de sistemas. É baseada na decomposição do sistema de acordo com os objetos que serão manipulados por este. Ela

oferece os principais benefícios: uma visão do sistema mais próximo do mundo real; uma modelagem do sistema baseada nos dados; e maior transparência da análise para o projeto. Este tipo de análise é subdividido em etapas. Serão abordadas três etapas: análise do problema, projeto da solução e a construção. Cada uma destas etapas é decomposta em atividades. A subdivisão cria um guia para o analista ter uma visão de onde ele está situado no projeto e decidir pelos próximos passos.

O fluxo de funcionamento das etapas utiliza da notação do diagrama de atividades da UML. Cada etapa exige que profissionais sejam designados para cumprir os papéis que serão exercidos. São os profissionais: analista de requisitos, analista de sistemas, arquiteto de *software*.

O objetivo da etapa de análise do problema na metodologia orientada a objetos é o mapear do problema, bem como propor alternativas de solução deste e apontar soluções e justificar a alternativa escolhida. Nesta fase, são identificadas duas macro atividades. A primeira é na área de engenharia de requisitos, a segunda na gerência de projetos. Na análise de requisitos são identificadas as ações que serão executadas pelo sistema para que este possa alcançar os objetivos esperados. Desta análise são extraídas as funcionalidades que o sistema deve ser capaz de executar e as interações com o ambiente computacional e humano. Estes são os requisitos funcionais de um sistema. Também são identificados os requisitos não funcionais que se referem às questões internas do software, não alteram sua funcionalidade, mas dão identidade ao produto do *software*.

Alguns dos requisitos não funcionais mais citados nas literaturas específicas:

- Usabilidade: o atendimento ao perfil das pessoas que irão utilizar o sistema. Isto influenciará na produtividade e aceitação do *software*;
- Confiabilidade: diz respeito aos resultados produzidos pelo sistema. Estes devem estar corretos;
- Desempenho: trata dos comprometimentos recursos que o sistema exige e tempo para execução em compatibilidade com suas funcionalidades;
- Segurança: Confidencialidade e proteção dos dados do usuário;
- Integridade: garantia de fidelidade dos dados contidos no sistema e garantia de que os erros serão recuperados.

Existem muitos outros, mas o que importa é o conjunto de requisitos assinalarem o comprometimento entre os desenvolvedores, clientes e usuários sobre o produto de software a ser desenvolvido. Antes do desenvolvimento do software, é necessário que haja o conhecimento de seu escopo, as necessidades de recursos e um cronograma de planejamento, além de muitas outras informações acerca do trabalho a ser realizado. Uma vez alcançado um nível de compreensão suficiente, é realizado atividades de preparação do plano de projeto. O plano de projeto se apoia nas atividades de engenharia de software da metodologia de organização. Cada parte do projeto deve ter estima das necessidades e recursos e outras especificações.

A programação orientada a objeto se difere da programação estruturada uma vez que as funções e os dados estão juntos e assim, formam o objeto. Já a orientação a objeto cria uma forma abstrata de analisar, ela utiliza conceitos do mundo e não conceitos computacionais. Porém, a análise orientada a objeto exige que o analista tenha conhecimento das notações utilizadas na orientação a objeto como diagrama de classe, interação, sequência. (SILVA, 2003)

2.4. Alguns comparativos entre as diferentes Análises

Técnicas	Abordagens	Ferramentas
Análise tradicional	Funcional	Textos Fluxogramas
Análise estruturada	Funcional Dados	DFD Diagrama estrutura de dados Mini especificações Normalização Dicionário de dados
Análise essencial	Funcional Dados Controle	Tabela de eventos DFD Diagrama de entidade Relacionamento Diagrama de transição de dados Diagrama de estrutura de dados Normalização Mini especificações Dicionário de dados

Quadro comparativo entre algumas análises.

3 - UML – CONCEITOS E SEUS DIAGRAMAS

3.1. Conceitos de Orientação a Objetos

Com o surgimento do paradigma da orientação a objetos no início da década de 90, várias metodologias de modelagem ficaram em voga naquilo que ficou conhecido como a "guerra dos métodos". De um modo geral, essas modelagens pecavam por tentar estender os métodos estruturados levando os usuários à insatisfação no uso do disponível.

Principais propostas de modelagem durante a década de 90.

Ano	Autor (Técnica)
1990	Shaler & Mellor
1991	Coad & Yourdon (OOAD – Object Oriented Analysis and Design)
1993	Grady Booch (Booch Method)
1993	Ivar Jacobson (OOSE – Object Oriented Software Engineering)
1995	James Rumbaugh et al (OMT – Object Modeling Technique)
1996	Wirfs-Brock (Responsibility Driven Design)

Com o passar do tempo, cada método ganhava uma fatia do mercado sob acirrada competição desnorteando os usuários. Esforços para unificação iniciaram em 1994 com James Rumbaugh e Grady Booch na *Rational Software* com intuito de unir os métodos Booch e OMT. Em 1995 eles lançaram um rascunho do Método Unificado versão 0.8. Nesta época, Ivar Jacobson se junta a eles, levando seu método OOSE, ficando esses autores conhecidos como os "três amigos". Em 1996, os três lançaram uma nova versão do Método Unificado, desta feita denominada UML – *Unified Modeling Language* versão 0.9, já sendo vista pelas empresas usuárias como uma ótima estratégia para seus negócios.

Ainda em 1996, um requerimento de proposta de padronização chamado RFP – *Request for Proposals* – emitido pela OMG – *Object Management Group* (consórcio internacional de empresas que define padrões na área de orientação a objetos) – gerou uma união de forças para produzir uma resposta a este RFP, gerando uma participação ativa da comunidade da engenharia de software no sentido de fortalecer a UML com alguns autores abrindo mão dos seus métodos em favor desse novo padrão.

No início de 1997, a Rational lançou a versão 1.0 da UML com resposta ao requerimento da OMG. O grupo original se expandiu incluindo outros participantes, que separadamente também haviam submetido resposta ao RFP para, juntos produzirem a UML versão 1.1 melhorando a semântica da versão 1.0. Em 14 de novembro de 1997, a UML, versão 1.1, foi adotada como padrão pela OMG.

A manutenção da UML passou a ser responsabilidade da RTF, pertencente a OMG, que tem como objetivo realizar revisão nas especificações referentes a erros, inconsistências, ambiguidades etc. Nos últimos anos, novas revisões foram editadas: em 1998 a UML 1.2 e 1.3, em 2001 a 1.4 e 1.5. Fiquemos atentos, pois já estamos nas atualizações da versão UML 2.x (http://www.omg.org/spec/UML/).

3.2. UML – A Linguagem de Modelagem Unificada

A UML é uma linguagem de modelagem, não um método. Métodos, em sua maioria, são compostos de uma linguagem de modelagem – notação gráfica – e de um processo – passos para elaboração de um projeto. A UML, através da sua estrutura, conduz à criação e leitura de seus modelos sem determinar quais e nem quando esses modelos precisam ser criados. Essa é uma responsabilidade do processo de desenvolvimento. Desta forma, pode-se usar qualquer processo com a UML, pois esta é independente de processo. A UML é uma linguagem para especificação, visualização, construção e documentação de artefatos de sistemas de software.

Por Booch, Rumbaugh e Jacobson:

"A UML proporciona uma forma padrão para a preparação de planos de arquitetura de projetos de sistemas, incluindo aspectos conceituais tais como processos de negócios e funções do sistema, além de itens concretos como as classes escritas em determinada linguagem de programação, esquemas de bancos de dados e componentes de software reutilizáveis"

A UML é uma linguagem visual. Logo, é constituída de elementos gráficos com objetivo de representar os conceitos do paradigma da orientação a objetos e permitindo construir diagramas que representam perspectivas de um dado sistema.

Os elementos gráficos têm **sintaxe** – forma predeterminada – evitando ambiguidades. Com isto evita que um analista modele uma classe como um retângulo e outro como um cubo. Elementos também possuem **semântica** – significado e função. Essas características permitem extensão, ou seja, podem ser "customizadas", adequadas, a um dado projeto incorporando suas especificidades.

A UML, por definição, é independente das linguagens de programação e de processos de desenvolvimento e alcançou dois aspectos: terminou com as diversas diferenças existentes entre os métodos de modelagem anteriores e unificou as perspectivas entre muitos sistemas de tipos diferentes (negócios X software), fases de desenvolvimento (análise de requisitos, projeto e implementação) e conceitos internos.

O projeto da UML objetivou uma linguagem que atingisse as seguintes metas:

• Prover à comunidade de uma linguagem de modelagem visual pronta para o uso e expressiva, possibilitando desenvolver e intercambiar modelos significativos.

• Fornecer extensibilidade e mecanismos de especialização pra estender os conceitos centrais.

• Suportar especificações que são independentes de processos de desenvolvimento e linguagens de programação particulares.

• Prover uma base formal para entendimento da linguagem de modelagem.

• Encorajar o crescimento do mercado de ferramentas de objetos.

• Suportar alto nível de conceitos de desenvolvimento como componentes, colaborações, estruturas e padrões.

• Integrar melhores práticas.

3.2.1. Visões de um sistema

No desenvolvimento de um sistema complexo, visões ou perspectivas diferentes precisam ser analisadas e os desenvolvedores da UML orientam que um sistema pode ser descrito por cinco seguintes visões:

• Visão de caso de uso: visão externa que descreve as interações entre o sistema e os agentes externos ao sistema com enfoque no comportamento do sistema. É uma visão inicial e que direciona o desenvolvimento das outras visões.

• Visão do projeto: destaca as características do sistema que dão suporte, tanto sob o ponto de vista estrutural quanto comportamental, às funcionalidades visíveis do sistema.

• Visão de implementação: atinge o gerenciamento de versões do sistema construídas através agrupamentos de módulos (componentes) e subsistemas.

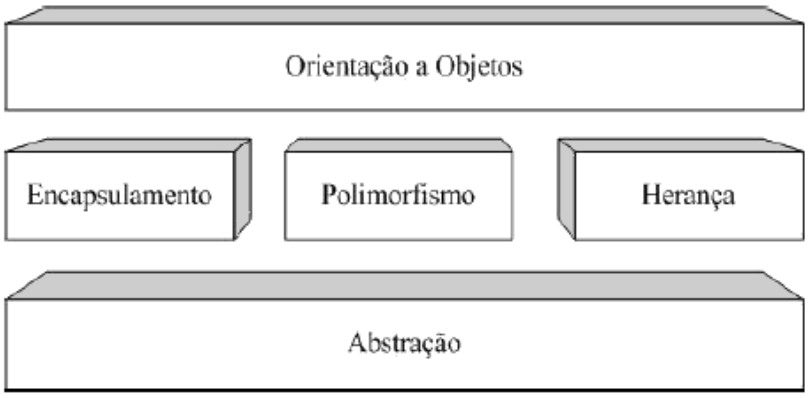
• Visão de implantação: mostra a organização dos subsistemas que compõem o sistema e a conexão entre eles.

• Visão de processo: enfatiza as características de concorrência (paralelismo), sincronização, desempenho, escalabilidade e rendimento do sistema.

Em decorrência da complexidades e características do sistema, nem todas as visões precisam ser criadas. Um sistema visando um ambiente de um único processador não necessita da visão de implantação. Um sistema de um único processo a visão de processo é irrelevante.

3.2.2. Princípio da Abstração

Este princípio se caracteriza pela observância dos aspectos mais importantes para o modelo, ignorando detalhes sem relevância para uma dada perspectiva. Com isto, facilita o gerenciamento da complexidade dando conta das características essenciais de um objeto. Abstração depende do observador.

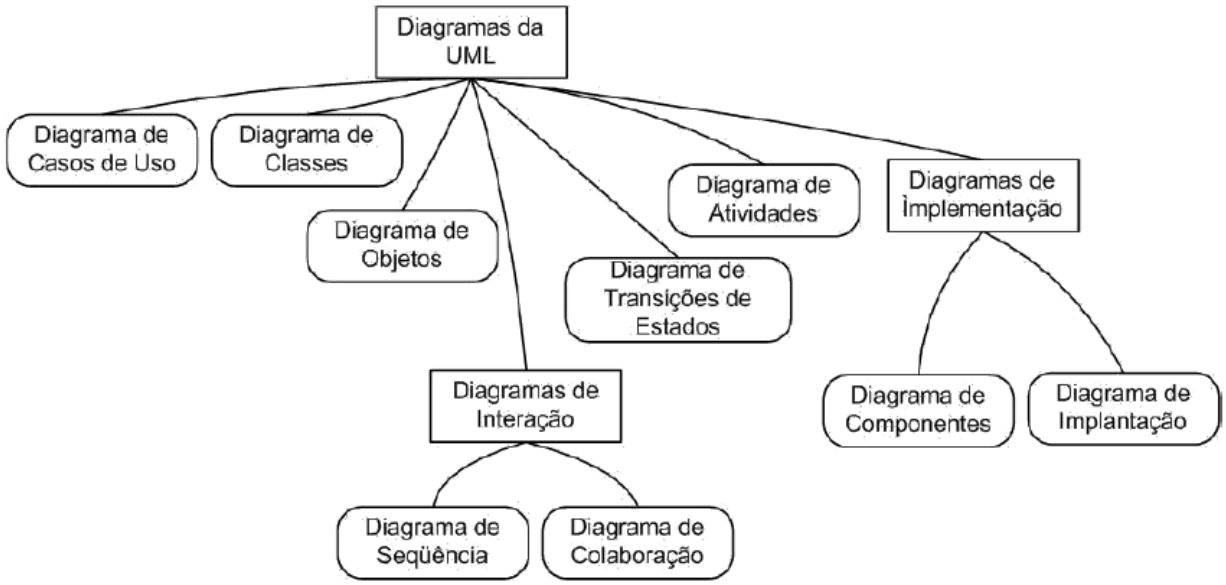


3.2.3. Diagramas da UML

A UML adota nove tipos de diagramas divididos em diagramas estruturais (estáticos) e dinâmicos conforme tabela abaixo:

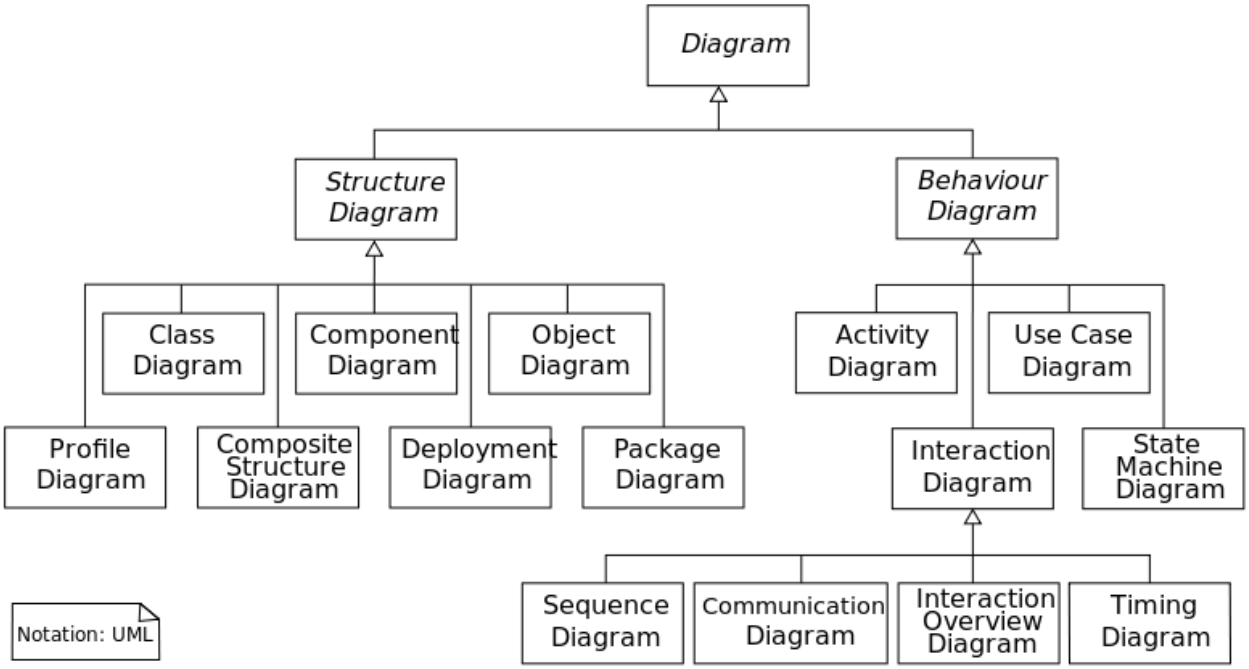
Diagramas Estáticos	Diagrama de Classes Diagrama de Objetos Diagrama de Componentes Diagrama de Implantação
Diagramas Dinâmicos	Diagrama de Casos de Uso Diagrama de Sequências Diagrama de Atividades Diagrama de Colaborações Diagrama de Gráficos de Estados

A modelagem através da UML envolve a utilização de vários documentos textuais ou gráficos. Esses documentos são denominados artefatos de software e compõem as visões do sistema. Os artefatos gráficos gerados durante o desenvolvimento de um sistema são definidos através da utilização dos diagramas da UML. Nova apresentação dos diagramas se segue:



Os retângulos com os cantos retos representam agrupamentos de diagramas da UML e os retângulos com cantos boleados, representam os diagramas em si. Cada um dos diagramas da UML fornece uma perspectiva parcial sobre o sistema consistente com as demais perspectivas.

Conforme o site http://www.omg.org/spec/UML, onde se pode acompanhar as atualizações e documentação sobre UML, na versão 2.4 de março de 2011 os diagramas são:



4 - CONCEITOS DE ORIENTAÇÃO A OBJETOS

4.1. Princípios da orientação a objetos

- Qualquer coisa é um objeto
- Objetos realizam tarefas através de requisições de serviços (métodos) a outros objetos
- Cada objeto pertence a uma determinada classe. Uma classe define objetos similares
- A classe define os comportamentos associados ao objeto
- Classes são organizadas em hierarquia

A orientação a objetos propicia aumento de produtividade, menor custos de desenvolvimento e de manutenção e, ainda, maior portabilidade e reutilização de código. Classes – de onde se originam os objetos – bem escritas reduzem tempo e custo de desenvolvimento.

4.1.1. Objeto

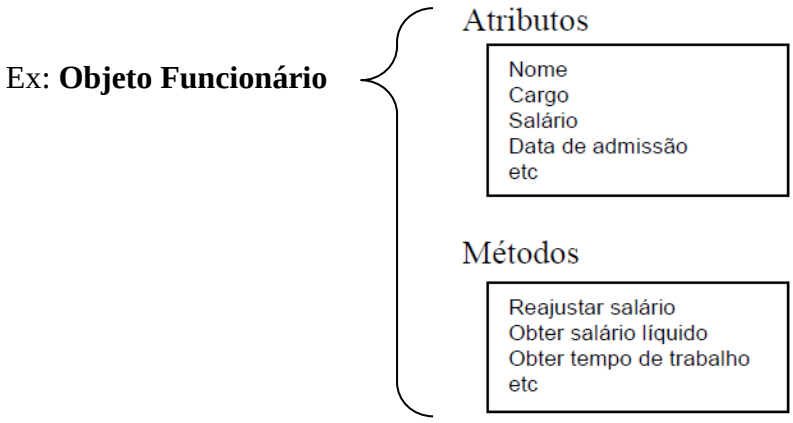
A grande vantagem da orientação a objetos é o fato de podermos abstrair das situações do dia-a-dia. Esta abstração é feita por representações do mundo real chamada objetos. Para isto, é necessário praticar o conhecimento que temos desde a infância, ou seja, identificar os objetos e seus comportamentos possibilitando categorizá-los. Ao olharmos uma caneta independente de seu formato ou cor de tinta conseguimos identificá-la.

Um objeto é qualquer coisa existente no mundo real seja concreto ou abstrato, ou ainda, que exista fisicamente ou apenas conceitualmente: aluno, professor, mesa, caneta, estoque, avaliação, janela do Windows, botão etc. Modelar um sistema baseado no paradigma da orientação a objetos e modelar os conceitos do nosso cotidiano.

Os objetos possuem **propriedades** – características – que são seus atributos e identificam o estado de um objeto. Um atributo é uma abstração do tipo de dados ou estado que os objetos de uma classe possuem.

Ex: **Objeto Maria**
Nome: Maria da Silva
Endereço: Rua A, 10
Sexo: Feminino
Data Nascimento: 20/05/1980
Altura: 1,68 m
Peso: 55 kg
Estado civil: casada
Cor dos olhos: verdes
Cor dos cabelos: castanho

Além dos atributos, os objetos possuem comportamentos que modificam seu estado ou prestam serviços a outros objetos. Para um funcionário que possua o atributo salário, este deve ser atualizado por operações do tipo Reajustar salário. Um **método** é a implementação de uma operação, ou seja, sua representação em código.



Os métodos de uma classe manipulam somente as estruturas de dados daquela classe. Uma classe tem conhecimento dos dados de outra por solicitação de serviços, ou seja, execução de operações. Esta solicitação é chamada de **mensagem**.

Ex: Informar a idade do objeto Maria chamando o método Calcular idade.

4.1.2. Limites de um objeto

Na modelagem, ao pensarmos em um objeto devemos fazê-lo dentro de um determinado contexto. Ao modelarmos o objeto Pessoa no papel de um aluno de uma escola de nível médio, não precisamos incluir atributos do tipo altura, peso, cor dos olhos e cor da pele. Entretanto se estivermos falando de aluno para uma escola de modelo e manequins, esses atributos já são relevantes.

Num retângulo podemos listar os atributos altura, largura, cor da linha, tipo da linha, cor de preenchimento, etc. Uma criança ao desenhar um retângulo pode apenas desenhar uma figura simples sem preenchimento ou pode se preocupar em colocar cores diferentes para a linha e seu interior. Um retângulo como objeto de um programa gráfico necessitaria de muito mais atributos que o do desenho de uma criança.

Este objeto retângulo poderia ter os métodos – operações – Desenhar, Pintar, Mudar Cor de Linha, Redimensionar, Mover etc.

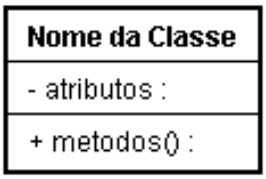
4.1.3. Identidade de objetos

Todos os objetos têm identidade e são diferentes – se distinguem por sua própria existência e não pelos valores de seus atributos. Uma fábrica produz 100 tipos de lápis com os mesmos atributos, mas que possuem identidades diferentes, pois fisicamente são diferentes.

5 - CLASSE

5.1. Classe

Quando identificamos atributos e métodos em objetos distintos estamos realizando sua classificação, ou seja: identificando classes.



Uma classe é a representação de um conjunto de objetos que compartilham a mesma estrutura de atributos, operações e relacionamentos dentro de um mesmo contexto – semântica. Uma classe especifica a estrutura de objeto sem se preocupar com valores.

Um objeto é uma instância – representação – de uma classe em um determinado momento.

Ex. 1: Carlos Ferreira é uma instância da classe Alunos da Universidade.

Ex. 2

Classe Pessoa

Nome Sexo Data de Nascimento Estado Civil

Objeto Maria

Nome	Maria da Silva
Sexo	Feminino
Data de Nascimento	20/05/1980
Estado Civil	Casada

Quando desenvolvemos um sistema, trabalhamos com as instâncias. Quando alguém preenche um formulário do imposto de renda, está usando uma instância. Assim sendo, temos:

Classe	Objetos
Funcionário	Ricardo Assunção Manoel da Veiga
Empresa	IBM do Brasil Ltda. Casas Bahia Ltda.
Veículo	Polo Corsa

5.2. Visibilidade

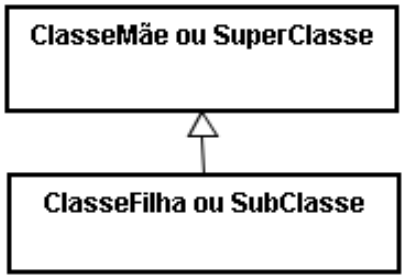
A visibilidade é utilizada para indicar o nível de acessibilidade de um determinado atributo ou método. Existem basicamente três modos de visibilidade: público, protegido e privado.

- A visibilidade **pública** é representada por um símbolo de adição (+), apresentado na frente da descrição do atributo ou método e significa que o atributo ou método pode ser utilizado por qualquer objeto de qualquer classe.
- A visibilidade **protegida** é representada por um símbolo de sustenido (#), e determina que somente objetos da classe possuidora do atributo ou método ou suas subclasses podem ter acesso ao mesmo.
- O atributo **privado** é representado por um símbolo de subtração (-) e significa que somente objetos da classe possuidora do atributo ou método poderão utilizá-lo.

6 - MODELAGEM ORIENTADA A OBJETOS

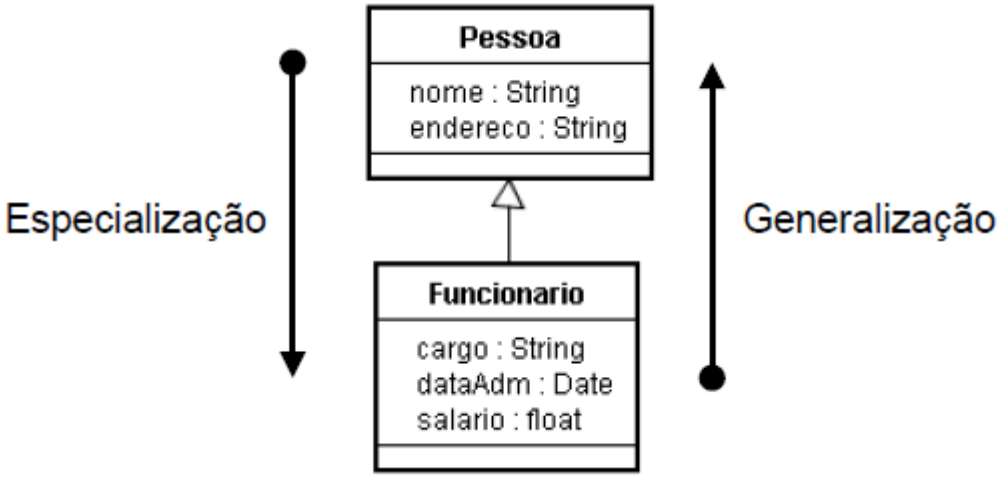
6.1. Herança

Pelo conceito de herança, são estabelecidas relações entre classes permitindo o compartilhamento de atributos e operações semelhantes dando a flexibilidade de criar nova classe incluindo somente as diferenças com relação à classe mais genérica.

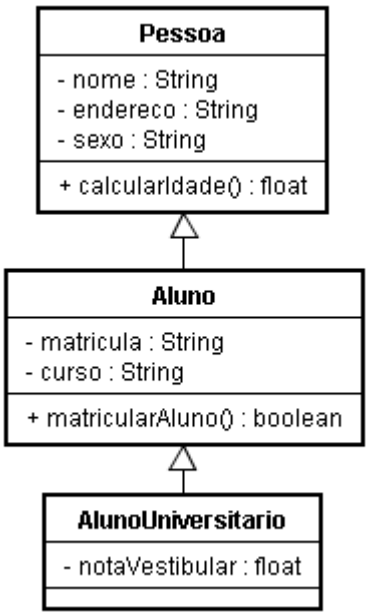


A classe-filha tem como base a classe-mãe. Por herança recebe todos os atributos e métodos da classe-mãe, sendo necessária somente a codificação dos atributos e operações específicos na classe-filha.

Suponhamos duas classes: a classe Pessoa com os atributos Nome e Endereço e a classe Funcionário com os atributos Nome, Endereço, Cargo, Data de admissão e Salário. As duas classes têm atributos comuns e seria redundância especificar esses atributos comuns para a classe Funcionário, daí:

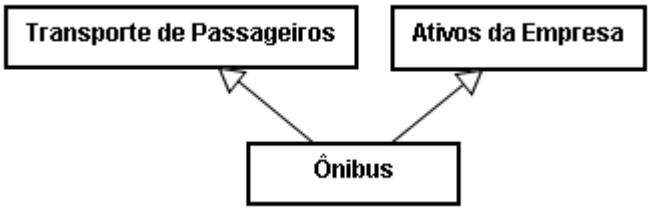


A classe Funcionário herda os atributos Nome e Endereço da classe Pessoa. Ao separarmos da classe Funcionário os atributos comuns, criando a classe Pessoa, estamos realizando uma generalização. De modo contrário, a partir da Pessoa separamos atributos mais específicos a serem colocados na classe Funcionário - estamos utilizando o conceito de especialização. Os dois conceitos apenas apresentam formas diferentes de enxergarmos o mesmo mecanismo – a herança.



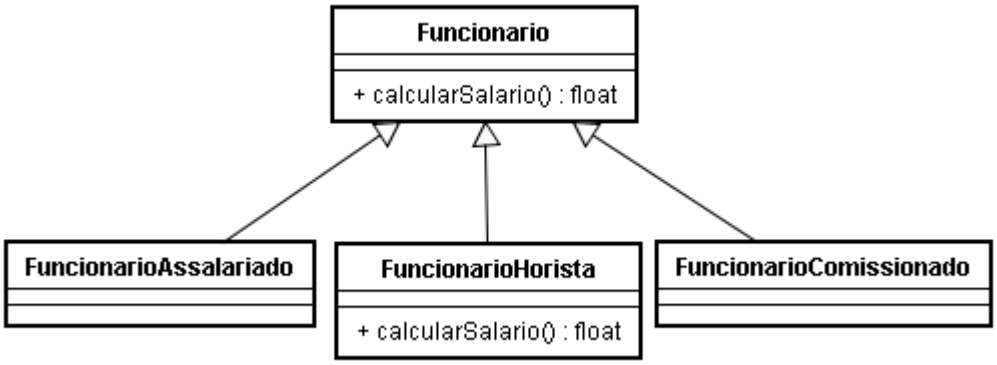
6.1.1. Herança Múltipla

Existe ainda o conceito de herança múltipla que permite que uma classe possua mais de uma superclasse herdando características de todas as ancestrais. Chama-se Herança Múltipla. Uma característica proveniente da mesma classe ancestral encontrada em mais de um caminho é herdada apenas uma vez. Uma classe com mais de uma superclasse é denominada classe de junção.



6.2. Polimorfismo

Um método pode ter implementações diferentes em diferentes classes de uma hierarquia de classes. Suponha um método que seja implementado nas classes-filha de forma diferente da classe-mãe. As operações das classes-filha prevalecerão sobre as da classe-mãe.



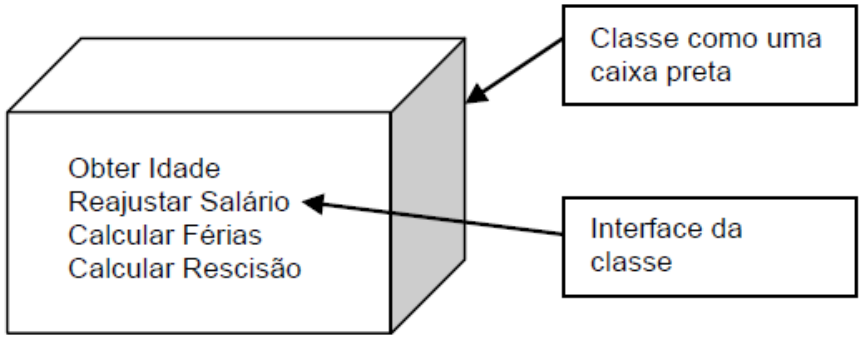
A operação Calcular Salário foi reescrita nas classes-filhas para que suas implementações possam ser modificadas. A operação da classe Funcionário é implementada considerando salário fixo que pode até ser igual para a classe-filha Funcionário Assalariado. Já na classe Funcionário Horista, para a mesma operação serão consideradas as horas-aula dadas, e para a classe Funcionário Comissionado a operação Calcular Salário será um percentual (comissão) sobre as vendas realizadas.

7 - ENCAPSULAMENTO, CLASSE ABSTRATA E INTERFACE

7.1. Encapsulamento

A utilização de um sistema orientado a objetos não deve depender de sua implementação interna e sim de sua interface, sendo isto uma das principais vantagens da orientação a objetos.

Projetada a classe, há a necessidade de proteger seus atributos e algumas operações. A interface – métodos - serve como intermediária entre a classe e o mundo externo. As alterações na classe são realizadas de forma transparente para o usuário.



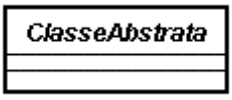
Devido ao conceito de encapsulamento, os atributos só podem ser acessados/atualizados pelas operações do objeto.

7.2. Classe Abstrata

São parcialmente implementadas e compostas por pelo menos 1 (em JAVA isso não é imposto) ou mais operações abstratas (operações sem implementação);

Tem como finalidade servir de base para a implementação do polimorfismo;

São identificadas pela palavra *abstract* e na UML são em itálico;



Não gera instâncias e existem apenas em hierarquia; Uma classe abstrata pode ter subclasses abstratas, mas nos níveis mais baixos (especializados) devem estar classes concretas;

Podem herdar de classes abstratas e não abstratas;

Podem possuir dados (atributos);

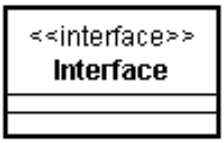
Uma subclasse de uma classe abstrata só pode ser instanciada se sobrescrever todos os métodos abstratos da superclasse e implementá-los.

7.3. Interface

É um conjunto de especificações de serviços (operações);

Um serviço tem especificação e implementação; A especificação define "*o que*" faz o serviço (operação);

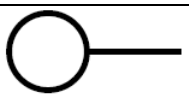
A implementação define "*como*" o serviço é feito (método); Uma interface pode ser representada na UML por uma pequena circunferência ligada por um segmento de reta sólido (não pontilhado) a uma classe.



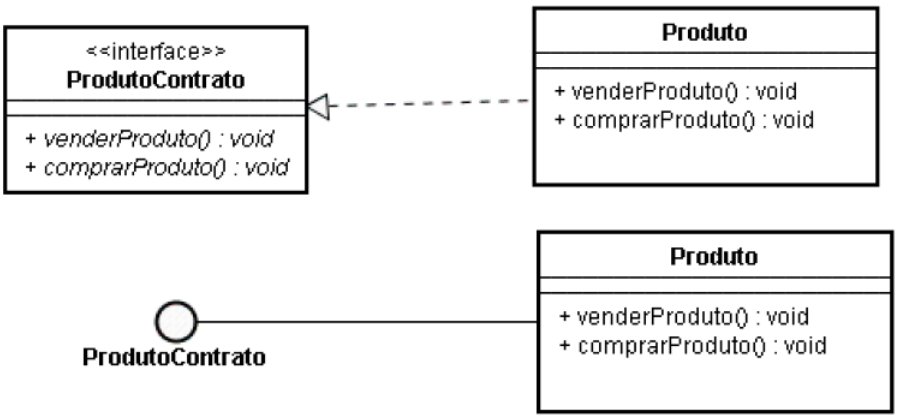
Uma interface é uma classe onde todos os serviços possuem apenas especificação (método com apenas assinaturas sem implementação).

Impõe especificação do que uma classe deve oferecer e implementar.

Há semelhanças entre Interface e Classe Abstrata.



Exemplos:


<<interface>>
ProdutoContrato
+ venderProduto() : void
+ comprarProduto() : void
Produto
+ venderProduto() : void
+ comprarProduto() : void
ProdutoContrato
Produto
+ venderProduto() : void
+ comprarProduto() : void

8 - INTRODUÇÃO AO CASO DE USO

8.1. Modelagem de Casos de Uso

O modelo de casos de uso, parte integrante da especificação de requisitos, é uma representação das funcionalidades externamente observáveis do sistema e dos elementos externos ao sistema que interagem com ele. Na modelagem de casos de uso é o diagrama de casos de uso e é importante, pois direciona diversas tarefas posteriores do ciclo de vida do sistema e por forçar os desenvolvedores moldarem o sistema de acordo com o usuário e não ao contrário.

8.1.1. Casos de Uso

Um caso de uso descreve uma sequência de interações entre um sistema e os agentes externos que utilizam esse sistema. Um caso de uso deve definir o uso de uma parte da funcionalidade de um sistema **sem revelar a estrutura e o comportamento internos desse sistema**. Um modelo de casos de uso típico contém vários diagramas de casos de uso.

O caso de uso, por expressar os requisitos do sistema – nada mais do que a essência deste – é utilizado durante todo o processo de desenvolvimento. Os requisitos além de serem a base para a criação dos modelos de análise, projeto e implementação também são usados como base para a criação de testes do sistema.

8.1.1.1. Formatos

1. Descrição contínua – Narrativa realizada através texto livre.

O Cliente chega ao caixa eletrônico e insere seu cartão. O Sistema requisita a senha do Cliente. Após o Cliente fornecer sua senha e esta ser validada, o Sistema exibe as opções de operações possíveis. O Cliente opta por realizar um saque. Então o Sistema requisita o total a ser sacado. O Sistema fornece a quantia desejada e imprime o recibo para o Cliente.

2. Descrição Numerada – Narrativa é descrita através de uma série de passos.

1- Cliente insere seu cartão no caixa eletrônico.
2- Sistema apresenta solicitação de senha.
3- Cliente digita senha.
4- Sistema exibe menu de operações disponíveis.
5- Cliente indica que deseja realizar um saque.
6- Sistema requisita quantia a ser sacada.
7- Cliente retira a quantia e recibo.

3. Descrição Particionada - Provê estrutura à descrição de casos de uso.

Cliente	Sistema
Insere seu cartão no caixa eletrônico.	
	Apresenta solicitação de senha.
Digita senha.	
	Exibe operações disponíveis.
Solicita realização de saque.	
	Requisita a quantia a ser sacada.
Retira a quantia e o recibo.	

8.2. Cenários

Normalmente, um caso de uso tem diversas maneiras de ser realizado. Um cenário, ou instância de um caso de uso, é a descrição de uma das maneiras pelas quais um caso de uso pode ser realizado.

Outro uso importante dos cenários está no esclarecimento e no entendimento dos casos de uso dos quais eles são instanciados. Frequentemente durante a construção de um cenário, novos detalhes de caso de uso são identificados ou mesmo novos casos de uso aparecem durante este processo. No cenário a seguir, o que aconteceria se o cliente desligar o telefone antes de realizar o pedido? E se o cartão de crédito do cliente não for aceito?

1 - Um Cliente telefona para a empresa.
2 - Um Vendedor atende ao telefone.
3 - Cliente declara seu desejo de fazer um pedido de compra.
4 - Vendedor pergunta a forma de pagamento.
5 - Cliente indica que vai pagar com cartão de crédito.
6 - Vendedor requisita o número do cartão, a data de expiração e o endereço de entrega.
7 - Vendedor pede as informações do primeiro item.
8 - Cliente fornece o primeiro item.
9 - Vendedor pede as informações do segundo item.
10 - Cliente fornece o segundo item
11 - Vendedor pede as informações do terceiro item
12 - Cliente e informa o terceiro item.
13 - Vendedor informa que o terceiro item está fora de estoque.
14 - Cliente pede para que O Vendedor feche o pedido somente com os dois primeiros itens.
15 - Vendedor fornece o valor total, a data de entrega e uma identificação do pedido.
16 - Cliente agradece e desliga o telefone.
17 - Vendedor contata a Transportadora para enviar o pedido de O Cliente.

Ao pensarmos nas ações realizadas numa rotina sem problemas, estamos lidando com um cenário perfeito, que será o **cenário principal** e descreve uma sequência de ações que serão executadas considerando que nada de errado ocorrerá durante a execução da sequência.

Quando consideramos que a sequência pode ser prejudicada de alguma forma – nada é perfeito – estaremos considerando **cenários alternativos**, que são apresentados como subitens do cenário principal.

6a - Se o cliente desligou, cessar esta sequência e esperar novo possível contato do cliente.
6b - Vendedor informa ao cliente sobre o problema com o cartão de crédito e solicita que o cliente retorne a ligação após solução do problema.

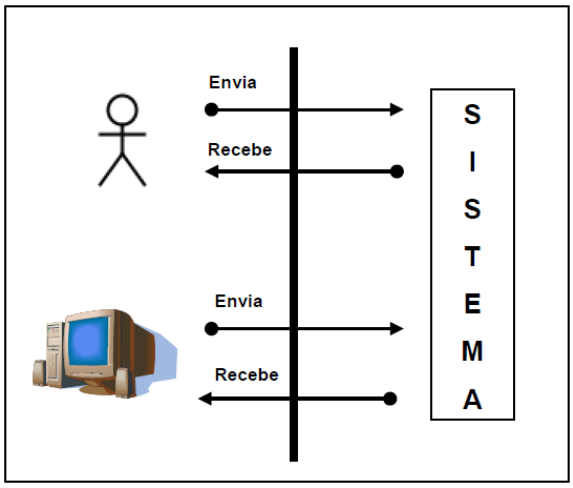
8.3. Atores

Na terminologia da UML, qualquer elemento externo que interage com o sistema é denominado **ator**. O termo "externo" indica que atores não fazem parte do sistema. O termo "interage" significa que um ator troca (envia/recebe) informações com o sistema. As categorias de atores, com exemplos, são listadas a seguir:

1. Pessoas (Empregado, Cliente, Gerente, Almoxarife, Vendedor, etc.)
2. Organizações (Empresa Fornecedora, Agência de Impostos, Administradora de Cartões, etc.)
3. Outros sistemas (Sistema de Cobrança, Sistema de Estoque de Produtos, etc.)
4. Equipamentos (Leitora de Código de Barras, Sensor, etc.)

Ao modelarmos um sistema, necessitamos saber até que ponto devemos nos preocupar. Estes pontos-limites são as fronteiras do sistema. Por exemplo: um sistema de controle de vendas emitirá em algum momento o faturamento semanal ou mensal de cada vendedor para o departamento do pessoal. Mas não é responsabilidade do sistema o que o departamento do pessoal fará com essa informação.

Os sistemas recebem e enviam informações para o mundo externo através de suas fronteiras e essas informações não podem cair num "buraco negro" na saída nem surgem por mágica na entrada. Na modelagem de dados esse papel é exercido pelo ator.



9 - DIAGRAMAS DE CASOS DE USO (DCU)

9.1. Diagramas de Casos de Uso - DCU

Utilizamos casos de uso para expressar a fronteira do sistema e/ou modelar requisitos do mesmo. Não é obrigatória a construção de diagramas de casos de uso, mas eles facilitam uma visão geral dos relacionamentos por estarem representados os atores, os casos de uso e os relacionamentos entre esses elementos com objetivo de apresentar um contexto mostrando os elementos externos e as formas que eles utilizam o sistema.

9.2. Atores no DCU

Um ator, independentemente do tipo de ator, é representado graficamente pela figura de um boneco (*stick man*). Outras representações para ator podem ser utilizadas, como um retângulo de classe com o estereótipo <<**ator**>> ou outros ícones que identifiquem mais precisamente o tipo de ator.

9.2.1. Tipos de Atores

Vimos que ator é qualquer elemento externo que interage com o sistema desempenhando um "papel". O nome de um ator deve lembrar o seu "papel", ou ainda, a sua classe.

Ex: Aluno, Correntista, Cliente etc.

Um ator pode participar de muitos casos de uso e um caso de uso pode envolver vários atores. Os atores são classificados em primários e secundários.

9.2.1.1. Atores primários

Atores primários iniciam uma sequência de interação com um caso de uso que lhe gera benefício direto. Basicamente, as funcionalidades de um sistema são definidas tendo em mente os objetivos dos atores primários.

9.2.1.2. Atores secundários

Atores secundários não são o objetivo das funcionalidades do sistema. Eles são responsáveis por operações de supervisão, operação de sistema, suporte na utilização do sistema, etc. Esses atores existem para viabilizar a utilização do sistema.

Ex.: De acordo com Bezerra (página 51: Na utilização de um browser de internet, para o usuário – ator primário – requisitar uma determinada página, o servidor Web – ator secundário – está sendo utilizado.)

9.3. Identificação dos elementos do modelo de casos de uso

Um caso de uso é representado por uma elipse com o nome do caso de uso representado ou dentro ou abaixo da elipse. É possível, ainda, uma elipse com compartimento referente a atributos, operações e pontos de extensão.

Os atores e os casos de uso são identificados a partir das informações coletadas na fase de levantamento de requisitos do sistema, onde também, os analistas devem identificar as atividades do processo de negócio relevantes ao sistema a ser construído.

9.4. Identificação de atores

Antes de começar a construir o modelo de caso de uso, todos os atores do sistema devem ser identificados. Para identificação dos atores é importante identificar quais as fontes de informação e seu destino depois de processadas pelo sistema. As fontes e os destinos serão atores em potencial.

Perguntas úteis neste processo:

• Que órgãos, empresas ou pessoas utilizarão o sistema?
• Que outros sistemas irão se comunicar com o sistema a ser construído?
• Alguém deve ser informado de alguma ocorrência no sistema?
• Quem está interessado em certo requisito funcional?

9.5. Identificação dos casos de uso

Na identificação dos casos de uso, estes devem ser separados em primários ou secundários.

9.5.1. Casos de uso primários

Representam os objetivos dos atores. Representam, por exemplo, os processos da empresa que estão sendo automatizados pelo sistema de software.

Perguntas úteis na identificação de casos de uso primários;

• Quais são as necessidades e objetivos da cada ator em relação ao sistema?
• Que informações o sistema deve produzir?
• O sistema deve realizar alguma ação que ocorre regularmente no tempo?
• Para cada requisito funcional, existe um ou mais caso de uso para atendê-lo?

9.5.2. Casos de uso secundários

É o caso de uso que não traz benefício direto para os atores, mas que é necessário para que o sistema funcione adequadamente. Esses casos de uso são categorizados:

• Manutenção de cadastro.
• Manutenção de usuários
• Manutenção de informações provenientes de outro sistema

Embora os casos de uso secundários devam ser considerados, relevância deve ser dada aos primários que representam os processos de negócio da empresa. O objetivo principal de um sistema é produzir algo de valor para o ambiente no qual será implantado.

10 - MODELO DE CASO DE USO

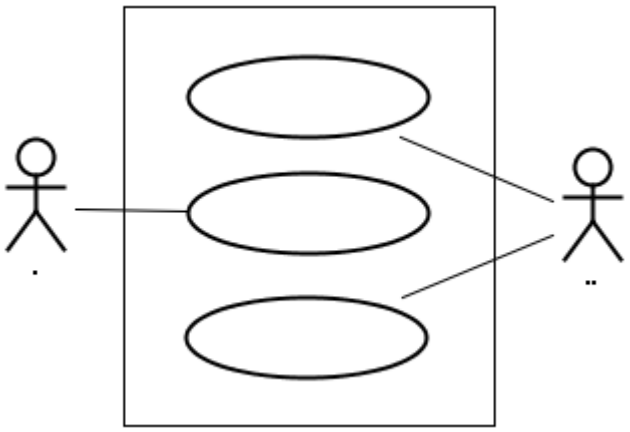
10.1. O modelo de casos de uso

O modelo de casos de uso é composto de:
• Diagrama de casos de uso - DCU.
• Documentação dos atores.
• Documentação dos casos de uso.

10.2. Construção do DCU

Uma das questões a ser resolvida pela UML era a comunicação entre os participantes do projeto em todas as suas fases. O modelo de casos de uso tem que ser uma ferramenta importante para que clientes, usuários finais e desenvolvedores discutam funcionalidades e comportamentos. Os DCU servem para dar suporte à parte escrita do modelo, daí o sucesso de um modelo começa pela facilidade de leitura do diagrama representando casos de uso.

Os casos de uso são desenhados dentro de um retângulo com os atores do lado de fora, dando ideia visual da fronteira do sistema.



10.2.1. Documentação dos atores

Breve descrição de cada ator deve ser incluída ao modelo de casos de uso. O nome de cada ator deve ser dado considerando seu papel no sistema modelado.

10.2.2. Documentação dos casos de uso

Não existe estrutura específica para descrição de um caso de uso. Os desenvolvedores devem utilizar itens de descrição que lhe forem realmente úteis. Segue uma proposta de itens.

Nome – o mesmo nome que aparece no diagrama de casos de uso que deve ser único.

Identificador – um código de referência nos diversos documentos do modelo. Por exemplo: CSU01, CSU02,...

Importância – é a categoria de importância dada ao caso de uso. São quatro categorias segundo Murray Cantor, 1998, como segue:

• Risco alto e prioridade alta.
• Risco alto e prioridade baixa.
• Risco baixo e prioridade alta.
• Risco baixo e prioridade baixa.

Sumário – pequena descrição do caso de uso.

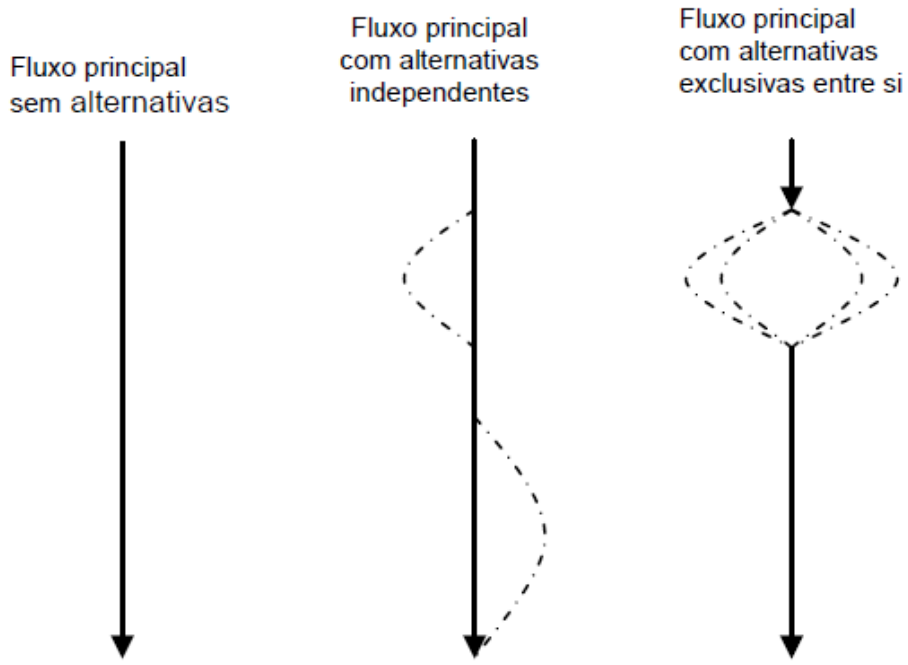
Ator primário – nome do ator que inicia o caso de uso.

Atores secundários – os nomes dos demais atores participantes do caso de uso, caso existam

Pré-condições – definição da(s) condição(ões) necessária(s) para que o caso de uso tenha início.

Fluxo principal – descrição do que normalmente acontece quando o caso de uso é realizado. Esta descrição deve ser escrita a partir do ponto de vista do usuário e usando a terminologia deste.

Fluxo Alternativos – descrição do que acontece quando o ator opta por uma escolha alternativa, diferente da descrita no fluxo principal, para alcançar o seu objetivo.



Fluxos alternativos também podem ser utilizados para descrever opções exclusivas entre si.

Fluxo de exceção – descrição de uma situação de exceção, ou seja, quando ocorre fato inesperado na interação entre ator e caso de uso. Ex.: usuário realiza alguma ação inválida.

A partir destas situações não usuais, o sistema pode se recuperar ou pode cancelar a realização da operação. Características de um fluxo de exceção:

• Representa um erro de operação durante o fluxo de caso de uso.
• Não tem sentido fora do contexto do caso de uso no qual ocorre.
• Deve indicar em que passo o caso de uso continua ou, conforme for, indicar explicitamente que o caso de uso termina.

Ex.: No caso de uso Realizar Pedido:
• E se o cartão de crédito excede o limite?
• E se a loja não tem a quantidade requisitada do produto?
• E se o cliente já tem um débito anterior?

Pós-condições – é um estado que o sistema alcança após o caso de uso ter sido realizado sem, entretanto, declarar como esse estado foi alcançado. Ex.: Informação X foi alterada.

Regras de negócio – A descrição de um caso de uso pode fazer referência a uma ou mais regras de negócio.

Histórico – são informações sobre o autor do caso de uso, modificações em seu conteúdo e datas principalmente.

Notas de implementação – na descrição dos fluxos de um caso de uso, o objetivo é manter a narrativa em um alto nível e utilizar a terminologia do domínio do negócio. Porém, o autor pode precisar colocar algumas considerações relativas à implementação do caso de uso que passam pela cabeça do autor enquanto o está descrevendo.

Quanto à complexidade

Sistemas de baixa complexidade são sistemas com até 12 casos de uso. Para estes, podemos ter só um diagrama de casos de uso.

Para sistemas complexos, tentar representá-lo em um único diagrama muito provavelmente prejudicará sua legibilidade. A alternativa passa pela criação de vários diagramas de casos de uso de acordo com a necessidade de visualização.

Ex.:

• Diagramas com um caso de uso e seus relacionamentos.

• Diagramas com todos os casos de uso para um determinado ator.

• Diagramas exibindo todos os casos de uso a serem implementados em um ciclo de desenvolvimento.

O modelador deve sempre tentar maximizar a legibilidade do DCU. Se necessário, mais de um diagrama de casos de uso devem ser criados.

10.3. Documentação associada ao modelo de casos de uso

O modelo de casos de uso tem como base os requisitos funcionais e força o desenvolvedor a pensar na interação entre os agentes externos e o sistema. Outros tipos de requisitos: desempenho, interface, segurança, regras do negócio, etc, que fazem parte do documento de requisitos de um sistema não são considerados pelo modelo de casos de uso. Alguns destes requisitos serão salientados.

10.4. Regras do Negócio (RN)

São políticas, condições ou restrições que devem ser consideradas na execução dos processos existentes em uma organização. Constituem uma parte importante dos processos organizacionais, pois descrevem como a organização funciona. Estas regras de negócio são identificadas durante o levantamento de requisitos. Da mesma forma que caso de uso, cada regra de negócio tem seu identificador: RN01, RN02, .., permitindo sua referência em outras partes do modelo.

Exemplos de regras de negócio:

• O valor total de um pedido é igual à soma dos totais dos itens do pedido acrescido de 10% de taxa de entrega.

• Os pedidos para um cliente não-especial devem ser pagos antecipadamente.

• Um aluno deve ter a matrícula cancelada se obtiver dois conceitos D no curso

• Um professor só deve estar lecionando disciplinas para as quais esteja habilitado.

Exemplo de formato para documentação de regras de negócio:

Identificação (Identificador)	
Descrição	Texto sumário descrevendo a regra de negócio.
Fonte	Fonte de informação que permitiu definir a regra.
Histórico	Histórico da evolução da regra: data da identificação, data da última atualização etc.

10.5. Requisitos Funcionais (RF)

Descrevem explicitamente as funcionalidades e serviços do sistema.

Deve documentar como o sistema deve reagir a entradas específicas, como deve se comportar em determinadas situações, o que o sistema não deve fazer.

Exemplo:

- O usuário pode pesquisar todo ou um subconjunto do banco de dados;
- O sistema deve oferecer telas apropriadas para o usuário ler documentos armazenados;
- Cada pedido deve ser associado a um identificador único (PID), o qual o usuário pode copiar para a área de armazenamento permanente da conta.

A imprecisão na especificação de requisitos é motivo de vários problemas, pois o desenvolvedor tende a interpretar o requisito da maneira mais fácil de implementar.

Exemplo: “O sistema deve oferecer telas apropriadas...”

O que são telas apropriadas?

10.6. Requisitos Não-Funcionais (RNF)

Definem propriedades e restrições do sistema e podem ser do sistema todo ou de partes do sistema.

Exemplos: segurança, desempenho, espaço em disco.

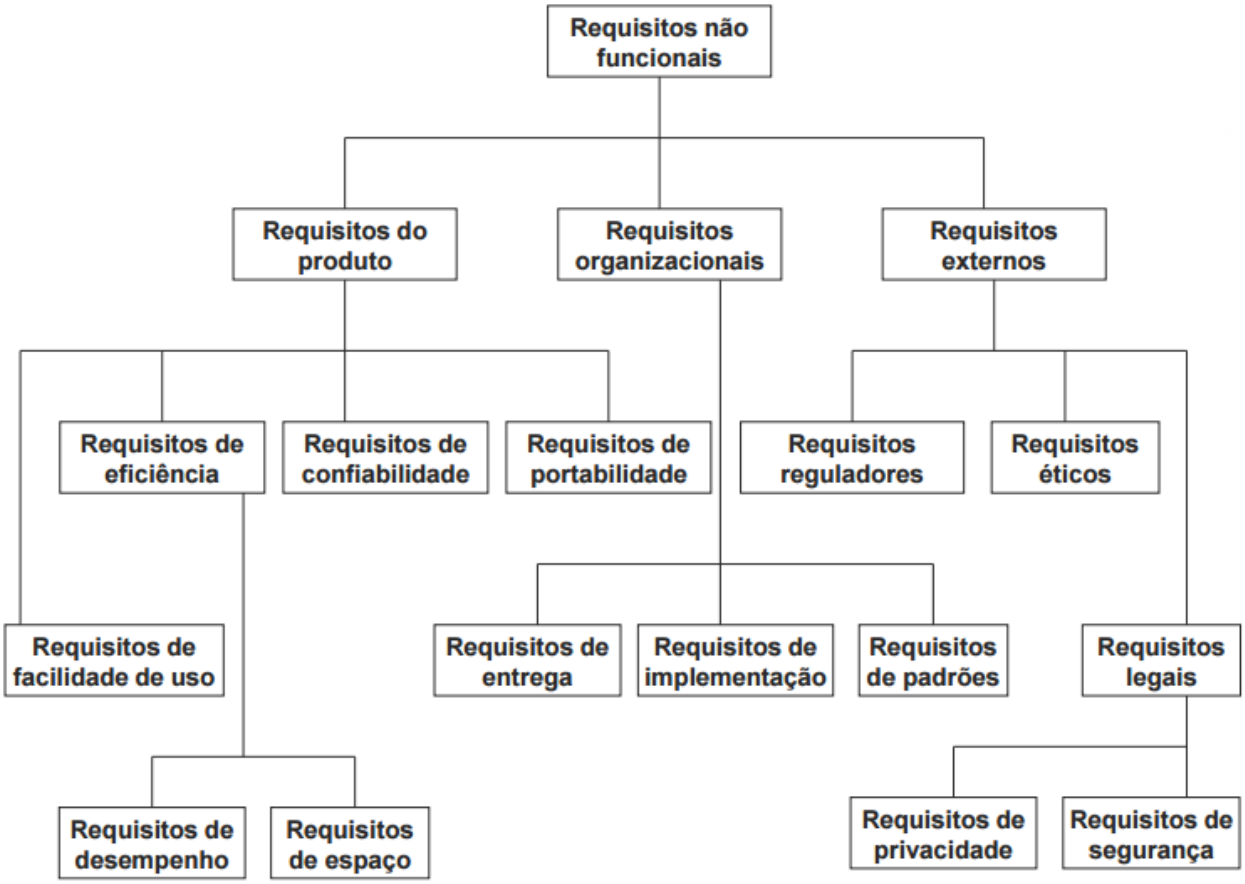
Os RNFs podem ser classificados em:

- *Requisitos do Produto*: Especificam o comportamento do software (ex.: desempenho). Exemplo: A interface do usuário deve ser implementada como simples HTML.

- *Requisitos Organizacionais*: Consequência de políticas e procedimentos das empresas (ex.: padrões do cliente). Exemplo: Todos os documentos entregues devem seguir o padrão de relatórios XYZ-00

- *Requisitos Externos*: Derivados do ambiente ou fatores externos ao sistema (ex.: legislação). Exemplo: Informações pessoais dos usuários não podem ser vistas pelos operadores do sistema.

A classificação completa dos requisitos não-funcionais é apresentada a seguir:



EXEMPLO: Documentação do modelo de casos de uso

Regras de negócio

Quantidade máxima de inscrições por semestre letivo (RN01)	
Descrição	Em um semestre letivo, um aluno não pode se inscrever em uma quantidade de disciplinas cuja soma de créditos ultrapasse a 20.

Quantidade de alunos possível (RN02)	
Descrição	Uma oferta de disciplina não pode ter mais de 40 alunos inscritos.

Pré-requisitos para uma disciplina (RN03)	
Descrição	Um aluno não pode se inscrever em uma disciplina para a qual não possua os pré-requisitos necessários.

Habilitação para lecionar disciplina (RN04)	
Descrição	Um professor só pode lecionar disciplinas para as quais esteja habilitado.

Quantidade máxima de inscrições por semestre letivo (RN05)	
Descrição	Em um semestre letivo, um aluno não pode se inscrever em uma quantidade de disciplinas cuja soma de créditos ultrapasse a 20.

Política de avaliação de alunos (RN06)	
Descrição	A nota de um aluno em uma disciplina – 0 a 10 – é obtida pela média de duas avaliações durante o semestre, Nota1 e Nota2, ou pela frequência nas aulas. • Se o aluno obtém nota maior ou igual a 7.0, está aprovado. • Se o aluno obtém nota maior ou igual a 5.0 e menor que 7.0, deve fazer a avaliação final. • Se o aluno obtém nota menor do que 5.0, está reprovado. • Se o aluno tiver uma frequência menor do que 75% em uma turma, está automaticamente reprovado.

Identificação de atores

Aluno – indivíduo que está matriculado na faculdade e que tem interesse em se inscrever em disciplinas do curso.

Professor – indivíduo que leciona disciplinas na faculdade.

Coordenador – pessoa interessada em agendar as alocações de turmas e professores, e visualizar o andamento de inscrições dos alunos.

Departamento de Registro Escolar – departamento interessado em manter informações sobre alunos matriculados e sobre seu histórico escolar

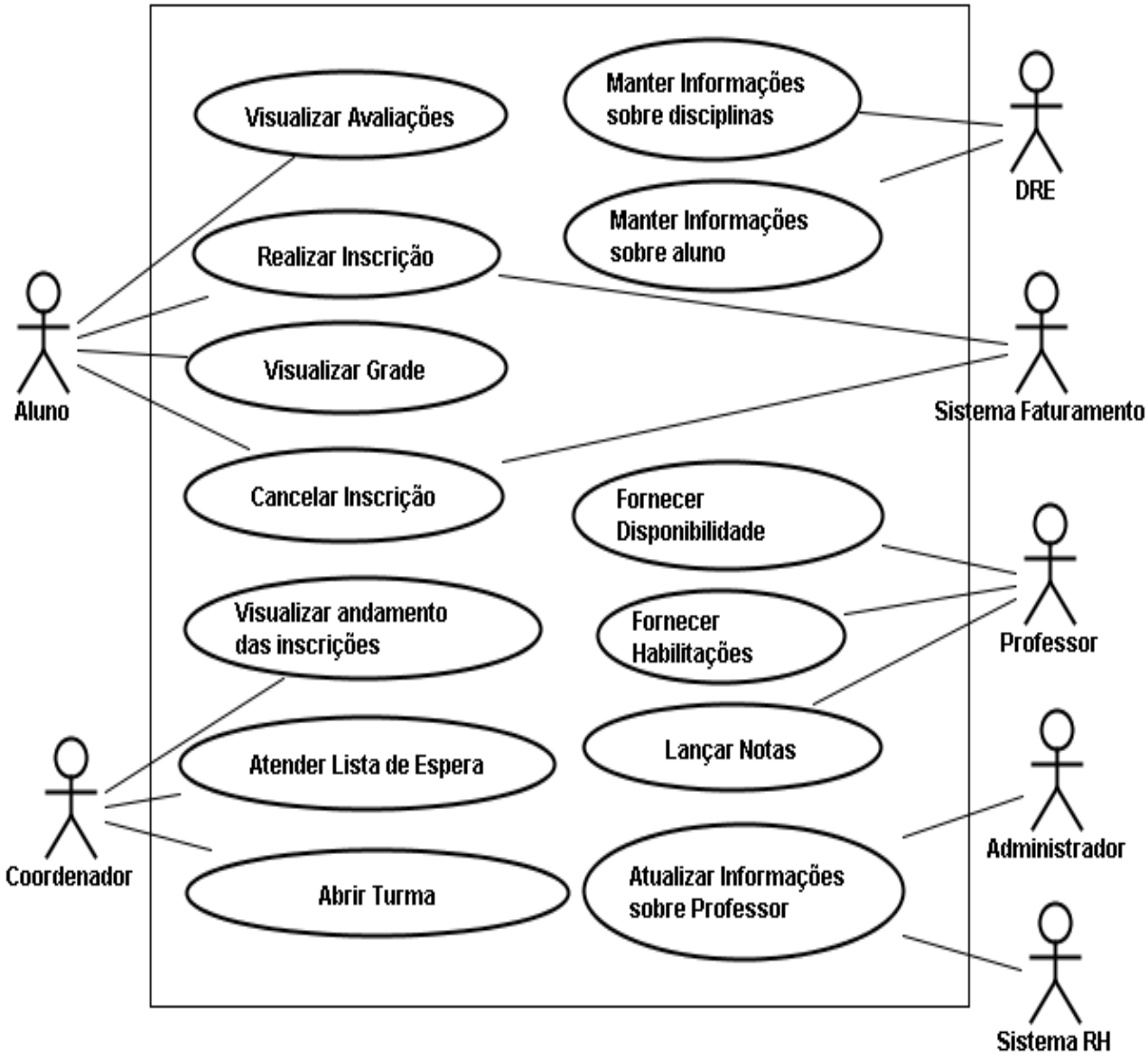
Sistema de RH – sistema responsável por fornecer informações cadastrais sobre os professores.

Sistema de Faturamento – sistema com interesse em obter informações sobre inscrições dos alunos para realizar o controle de pagamento de mensalidades.

Administrador – indivíduo que poderá atualizar informações sobre professores.

Diagrama de casos de uso

SISTEMA DE CONTROLE ACADÊMICO



Documentação dos Casos de uso

Serão apresentadas somente algumas descrições de casos de uso neste exemplo.

Realizar inscrições (CSU01)

Sumário: Aluno usa o sistema para realizar inscrição em disciplinas.
Ator primário: Aluno
Atores secundários: Sistema de Faturamento
Pré-condições: O aluno está identificado pelo sistema

Fluxo principal:
1.O aluno solicita a realização de inscrição.
2.O sistema apresenta as disciplinas disponíveis para o semestre corrente e para as quais o aluno tem pré-requisitos.
3.O aluno seleciona as disciplinas desejadas e as submete para inscrição.
4. Para cada disciplina selecionada, o sistema aloca o aluno em uma turma que apresente uma oferta para tal disciplina.
5.O sistema informa as turmas nas quais o aluno foi alocado. Para cada alocação, o sistema informa o professor, os horários e os respectivos locais das aulas de cada disciplina.
6.O aluno confere as informações fornecidas.
7.O sistema envia os dados sobre a inscrição do aluno para o Sistema de Faturamento e o caso de uso termina.

Fluxo alternativo (4): Inclusão em lista de espera.
a. Se não há oferta disponível para alguma disciplina selecionada pelo aluno, o sistema reporta o fato e fornece a possibilidade de inserir o aluno em uma lista de espera.
b. Se o aluno aceitar, o sistema o insere na lista de espera e apresenta a posição na qual o aluno foi inserido na lista. O caso de uso retorna ao passo 4.
c. Se o aluno não aceitar, o caso de uso prossegue a partir do passo 4.

Fluxo de exceção (4): Violação de RN01
a. Se o aluno atingiu a quantidade máxima de inscrições (RN01), o sistema informa ao aluno a quantidade de disciplinas que ele pode selecionar, e o caso de uso retorna ao passo 2.

Pós-condições: O aluno foi inscrito em uma das turmas de cada uma das disciplinas desejadas, ou foi adicionado a uma ou mais listas de espera.

Regras de negócio: RN01, RN02, RN03.

Visualizar avaliações (CSU02)

Sumário: Aluno visualiza avaliação que recebeu (notas e frequência) nas turmas de um semestre letivo.
Ator primário: Aluno
Pré-condições: O aluno está identificado pelo sistema

Fluxo principal:
1.O aluno solicita a visualização das avaliações para as ofertas de disciplinas em que participou.
2.O sistema exibe os semestres letivos nos quais o aluno se inscreveu em pelo menos uma oferta de disciplina.
3.O aluno seleciona os semestres letivos cujas avaliações deseja visualizar.
4.O sistema exibe uma lista de avaliações agrupadas por semestres letivos selecionados e por turma.
5.O aluno visualiza as avaliações e o caso de uso termina.

Fluxo de exceção (2): Aluno sem inscrição
a. Não há semestre letivo no qual o aluno tenha participado em alguma oferta de disciplina: o sistema reporta o fato e o caso de uso termina.

Pós-condições: O aluno obteve as avaliações que desejava visualizar.

Fornecer disponibilidades (CSU03)

Sumário: Professor fornece a sua grade de disponibilidade (dias e horários) e disciplinas que deseja lecionar no próximo semestre letivo.
Ator primário: Professor
Pré-condições: O professor está identificado pelo sistema

Fluxo principal:
6.O professor indica o desejo de fornecer sua disponibilidade para o próximo semestre letivo.
7.O sistema apresenta a lista de disciplinas disponíveis para as quais o professor está habilitado e uma grade de dias e horários da semana em branco.
8.O professor seleciona as disciplinas que deseja lecionar.
9.O professor preenche a grade com sua disponibilidade.
10. O sistema registra as informações fornecidas e o caso de uso termina.

Fluxo alternativo (3): Modificações na grade atual.
d.O professor solicita que o sistema apresente a mesma configuração de disponibilidade do semestre atual.
e. O sistema apresenta a configuração requisitada.
f. O professor realiza as modificações que deseja na configuração.
g.O sistema registra a informação e o caso de uso termina.

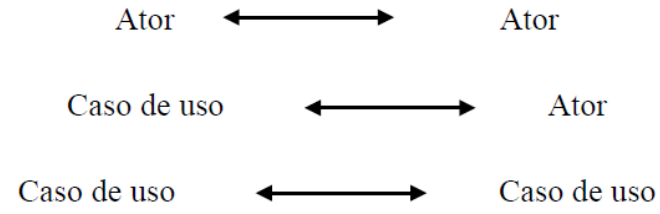
Fluxo de exceção (4): Disciplinas não selecionadas.
b. Se o professor não selecionou disciplina alguma ou não preencheu a grade de disponibilidade, o sistema reporta o fato e o caso de uso continua a partir do passo 2..

Pós-condições: O sistema registrou a disponibilidade do professor para o próximo semestre letivo.

11 - CASO DE USO E SEUS RELACIONAMENTOS / ASSOCIAÇÃO

11.1. Relacionamentos

Casos de uso e atores não existem sozinhos. Os casos de uso representam conjuntos bem definidos de funcionalidades do sistema, que não podem trabalhar sozinhas no contexto do sistema. Portanto, esses casos de uso precisam se relacionar com outros casos de uso e com atores que enviarão e receberão mensagens destes. Então, temos relacionamentos entre:

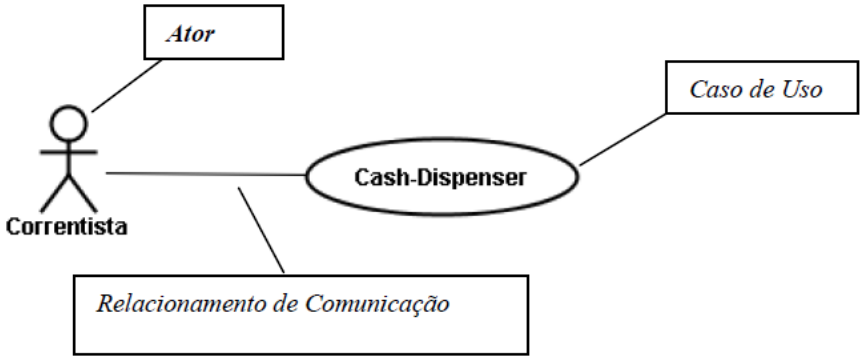


Os relacionamentos entre casos de uso são: inclusão, extensão e generalização. Já o relacionamento entre caso de uso e ator definimos de comunicação ou associação. Entre atores só existe o relacionamento de generalização.

11.1.1. Relacionamento de Comunicação (ou associação)

Representa a interação do ator com o caso de uso por meio do envio e recebimento de mensagens. O ator pode se relacionar com mais de um caso de uso e este tipo de relação é sempre binária, ou seja, envolvem sempre dois elementos.

Ex.: O correntista envia e recebe mensagens do sistema cash-dispenser do caixa automático de um banco.



11.1.2. Relacionamento de Inclusão

Existe somente entre casos de uso. Indica a utilização de um determinado caso de uso no cenário de outro caso de uso.

Ex.: o caso de uso Validar Matrícula é útil para casos de uso com renovar matrícula de aluno, solicitar histórico escolar, lançar notas de provas etc.

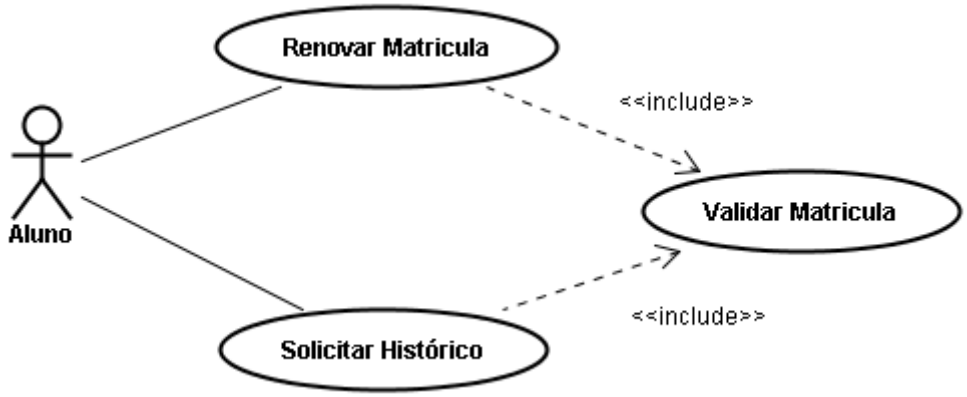
Textualmente, um relacionamento de inclusão é representado com o termo **Inclui** (**Caso de uso**).

Ex.: Cenário principal do caso de uso Solicitar Historio Escolar

1. O aluno digita sua matrícula.
2. O sistema verifica se a matrícula é válida e ativa Include (Validar Matrícula)

Este tipo de relacionamento também pode ser entendido pela técnica de definição de rotinas em linguagens de programação. Quando dois ou mais casos de uso têm uma sequência comum de interações, essa sequência comum pode ser descrita em um caso de uso comum. A

partir daí os casos de uso anteriores podem usar (incluir) este caso de uso evitando a descrição da uma sequência de uso mais de uma vez.

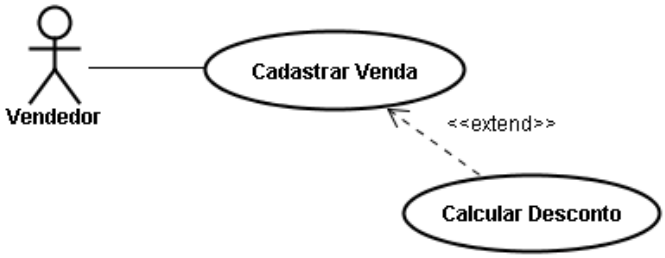


11.1.3. Relacionamento de Extensão

Ocorre entre casos de uso e é utilizado para modelar situações em que diferentes sequências de interações podem ser inseridas em um caso de uso caracterizando caso de uso estendido. Representa um comportamento opcional que só ocorre sob certas condições ou depende da escolha de um determinado ator. O caso de uso base é chamado de **caso de uso estendido** e o caso de uso opcional e o **caso de uso extensor**.

Ex.: Na sequência de interações do caso de uso de um cadastro de vendas, a operação de desconto só pode ser executada pelo gerente. Essa rotina pode ser transferida para um caso de uso de extensão.

Os pontos de extensão são representados pelo termo **Estende (Caso de Uso)**.



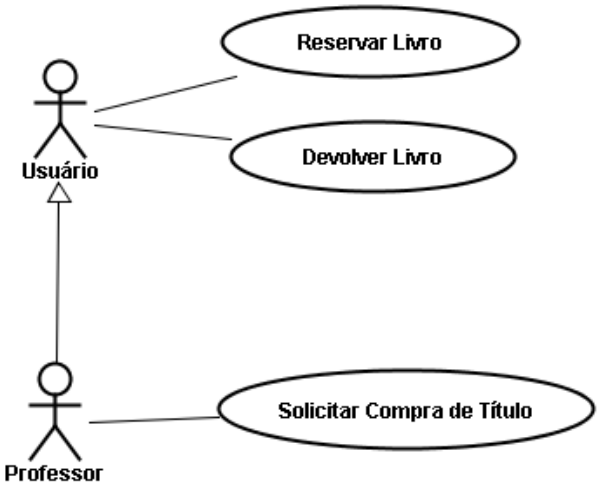
11.1.4. Relacionamento por Generalização

Ocorre entre casos de uso ou entre atores. Este relacionamento permite que um caso de uso, ou ator, herde características de outro caso de uso, ou ator, mais genérico. O herdeiro do relacionamento pode especializar o comportamento do base.

Na generalização entre casos de uso, A e B, as sequências de operação de A valem para B e este participa de qualquer relacionamento no qual A participa. Somente o comportamento diferente para o caso de uso herdeiro, B, precisa ser definido. Ex: Validar usuário, validar senha e validar retina.

No caso de generalização entre atores, o ator herdeiro possui o mesmo comportamento do ator base e, ainda, o herdeiro participa de todo relacionamento que o ator base participa.

Ex.: numa biblioteca existem dois tipos de usuário: alunos e professores. Ambos podem realizar empréstimos e reservas de títulos, mas somente o professor pode requisitar a compra de livros.



11.2. Comparações entre relacionamentos

Os relacionamentos de inclusão, extensão e generalização possibilitam manter a descrição dos casos de uso mais simples devido à fatoração de sequências de interações comuns. Mas não se deve usar em excesso a fatoração para não gerar dificuldade no entendimento do modelo.

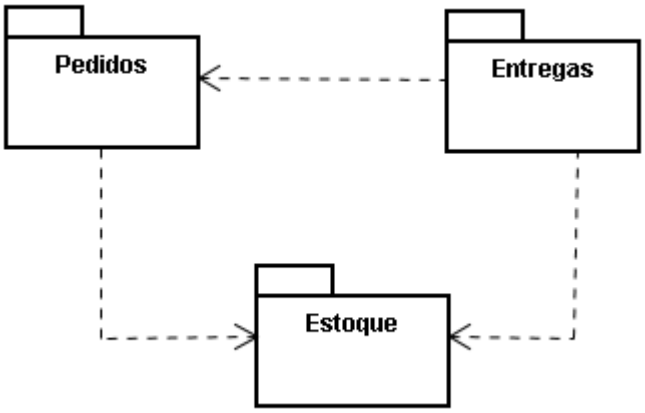
12 - OUTRAS CARACTERÍSTICAS GERAIS DA UML

12.1. Notas

Comentário usado em um diagrama. É exibida como um retângulo com o canto direito superior dobrado.

12.2. Pacotes

Em sistemas complexos, muitos casos de uso, o modelador pode decidir por formar grupos de casos de uso logicamente relacionados e que podem ser visualizados de uma só vez. Pacote é um mecanismo de agrupamento definido pela UML onde elementos semanticamente relacionados podem ser agrupados



Chama-se de dependência os relacionamentos entre pacotes, ou seja, um pacote depende de outro. Para nomear essas dependências, o desenvolvedor pode criar seus próprios estereótipos.

No contexto dos casos de uso, os pacotes podem ser utilizados com diversos objetivos como:

- Estruturar o modelo de casos de uso para refletir os tipos de usuários do sistema.
- Definir a ordem na qual os casos de uso serão desenvolvidos.
- Definir o grau de correlação entre casos de uso.

A **UML** podemos fazer uma modelagem visual de maneira que os relacionamentos entre os componentes do sistema sejam melhor visualizados, compreendidos e documentados.

Um diagrama de pacotes mostra pacotes e relações entre pacotes

Na realidade, não existem propriamente diagramas de pacotes em UML; em vez disso, pacotes e relações entre pacotes aparecem noutros diagramas, de acordo com o tipo de pacote:

- Pacotes de classes (pacotes lógicos) - em diagramas de classes
- Pacotes de componentes – em diagramas de componentes
- Pacotes de nós – em diagramas de distribuição
- Pacotes de casos de utilização – em diagramas de casos de utilização

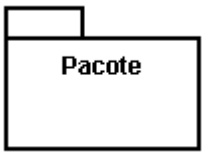
A UML é uma linguagem para especificar, construir, visualizar e documentar um sistema de software que surgiu com a fusão das metodologias já anteriormente usadas e tem como objetivo:

1. Modelar sistemas usando os conceitos da orientação a objetos

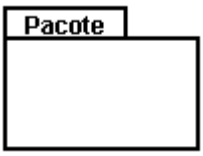
2. Estabelecer um elo explicito entre os artefatos conceituais e os executáveis
3. Tratar questões de representar sistemas complexos de missão crítica
4. Criar um linguagem de modelagem que possa ser usada por homens e por máquinas

O pacote é uma das notações mais usadas pela linguagem UML:

Pacote - Organiza as classes de objetos em grupos. É representado como um retângulo com uma aba no canto superior esquerdo.

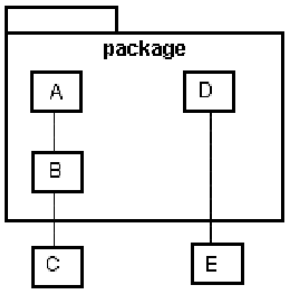


As classes são definidas dentro do pacote
• Pacotes diferentes podem ter classes com nomes iguais
• Para referenciar uma classe de um pacote usamos a sintaxe:
• **NomedoPacote :: NomeDaClasse**



Visibilidade do Pacote:

1. **Privado** - Só o pacote que define determinadas classes tem acesso a elas.
2. **Protegido** - Só os pacotes gerados a partir do pacote podem acessar suas classes.
3. **Público** - O conteúdo do pacote pode ser acessado por outros elementos
4. **Implementação** - Idêntico a definição do pacote privado com algumas restrições para o uso dos elementos do pacote.



Atributo e Método em A	*Podem ser acessados por*
Público (+)	A,B,C,D,E
Protegido (#)	A,B,C,D
Pacote (default) (Java)	A,B,D
Privado (-)	A (Maior Encapsulamento)

12.3. Estereótipos

É um elemento de modelagem que rotula tipos de classes de objeto sendo que uma Classe de objetos pode ter um ou mais tipos de estereótipos. É descrito entre os sinais << e >> ou como um ícone no canto superior direito.

• «system» - pacote que representa o sistema completo que será modelado (incluindo todos os modelos e elementos dos modelos);

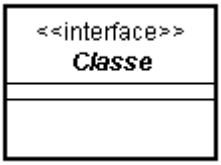
• «subsystem» - pacote que representa uma parte independente de sistema completo que será modelado; corresponde normalmente a um corte "vertical";

• «facade» (fachada) - pacote que constitui uma vista sobre outro pacote (não acrescenta funcionalidades, apenas apresenta de forma diferente);

• «framework» (infra-estrutura aplicacional) - pacote que representa um conjunto de classes abstratas e concretas concebido para ser estendido, implementando a funcionalidade típica de um determinado domínio de aplicação;

• «stub» - pacote que serve como *proxy* para o conteúdo público de outro pacote;

• «layer» - pacote que representa uma camada horizontal de um sistema.



13 - MODELAGEM DE CLASSE

13.1. Modelagem de Classe

No paradigma da orientação a objeto, nada mais razoável do que documentar os objetos encontrados nos requisitos do sistema. Após extrair os objetos dos requisitos, será necessário separar e classificar as suas características desses objetos, modelando as classes desses objetos

O modelo de casos de uso de um sistema é construído para formar a visão de casos de uso do sistema que fornece uma perspectiva do sistema a partir de um ponto de vista externo. A funcionalidade externa de um sistema orientado a objetos é fornecida através de colaborações entre objetos. Externamente ao sistema, os atores visualizam resultados de cálculos, relatórios, confirmações de requisições etc. Internamente, os objetos colaboram uns com os outros para produzir os resultados que são vistos externamente. Essas colaborações podem ser vistas sobre os aspectos: Estrutural Estático e Dinâmico.

13.1.1. Estrutural Estático

Que descreve como o sistema está estruturado internamente para que as funcionalidades sejam produzidas. Não apresenta informações de como os objetos interagem no decorrer do tempo. É estrutural porque a estrutura das classes de objetos e as relações entre elas são representadas.

13.1.2. Dinâmicos

Que descreve a troca de mensagens entre objetos e a reação destes, a eventos que ocorram no sistema.

Esses aspectos não são independentes. Um serve para adicionar detalhes no outro.

13.2. Modelo de classes

Deverá ser notado que um modelo de classes evolui durante as iterações do desenvolvimento do sistema. À medida que o sistema é desenvolvido, o modelo de classes é incrementado com novos detalhes estabelecendo três níveis sucessivos de abstração: domínio, especificação e implementação.

13.2.1. Modelo de classes de domínio

Representa as classes no domínio do negócio em questão. Construído na fase de análise, não leva em consideração restrições inerentes à tecnologia a ser utilizada na solução de um problema.

13.2.2. Modelo de classes de especificação

Construído na fase de desenvolvimento, é uma extensão do modelo anterior com a adição de detalhes específicos conforme a solução de software escolhida. Neste modelo são definidas, caso necessário, novas classes para desenvolver a solução do problema. Descreve a solução em nível alto de abstração.

13.2.3. Modelo de classes de implementação

Extensão do modelo de especificação, da conta da implementação das classes em alguma linguagem de programação orientada a objetos. É construído na fase de implementação.

13.3. Nomenclatura

Uma equipe de desenvolvimento pode escolher qualquer estilo de nomenclatura que desejar, porém após escolhido, é importante que seja seguido de forma consistente. Usaremos a notação definida no livro-texto.

Para nomes de classes e de relacionamentos, as palavras componentes do nome são escritas começando por letra maiúscula. Ex.: Cliente, ItemPedido, Pedido, Aluno, Reside, Realiza etc.

Para nomes de atributos e de operações, a primeira palavra do nome deve ser escrita com letra minúscula e as palavras subsequentes iniciadas com letra maiúscula. Siglas são mantidas inalteradas. Ex.: quantidade, CNPF, dataNascimento, obterTotal etc.

13.4. Introdução ao Diagrama de Classes

O diagrama de classes é utilizado na construção do modelo de classes desde o nível de análise até o nível de especificação. É o diagrama mais rico em termos de notação.

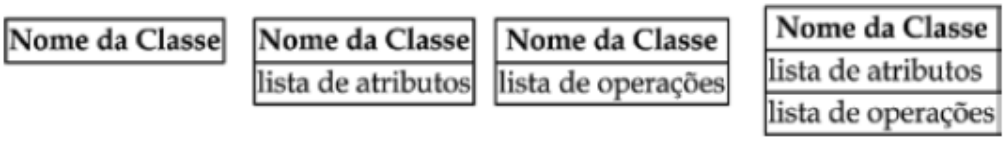
Uma classe é representada através de um retângulo com, no máximo, três compartimentos.

No primeiro, o de cima, vai exibido o nome da classe. Este nome deve ser no singular e com as palavras componentes começando por letra maiúscula.

No segundo, são declarados os atributos correspondem às informações que um objeto armazena.

No terceiro compartimento são declaradas as operações – métodos – que correspondem às ações que o objeto pode realizar.

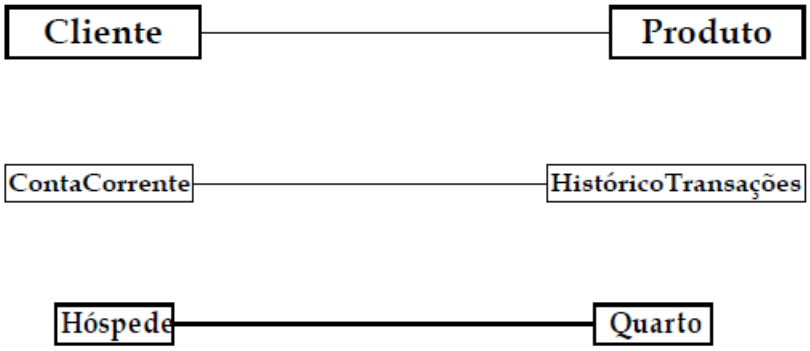
As possíveis notações para representar classes são representadas abaixo:



Ao contrário dos atributos de objetos – instâncias de uma classe – para os quais cada objeto tem seu próprio valor (dado), objetos de uma classe compartilham as mesmas operações – métodos. O nome de uma operação normalmente contém um verbo e um complemento, e terminam com um par de parênteses.

13.5. Relacionamentos

Novamente, um objeto é uma ocorrência ou uma instância de uma classe. Para representar o fato de que objetos podem se relacionar entre si existe a associação. Uma associação é representada no diagrama de classes através de um segmento de reta ligando as classes às quais os objetos relacionados pertencem. Exemplos de associações entre objetos:



13.6. Multiplicidades

Nas associações é possível representar a informação dos limites inferior e superior da quantidade de objetos aos quais outro objeto pode estar associado. Estes limites são denominados multiplicidades. Cada associação de um diagrama possui duas multiplicidades, uma em cada extremo da linha de associação.

Nome	Simbologia
Apenas Um	1..1 (ou 1)
Zero ou Muitos	0..* (ou *)
Um ou Muitos	1..*
Zero ou Um	0..1
Intervalo Específico	Li..Ls

Ex.:

A leitura informa que um cliente está associado a nenhum ou a vários pedidos e um pedido está associado a um e somente um cliente. Nos símbolos em que aparece o " * ", este denota que não há um limite superior predefinido para a quantidade máxima de objetos com os quais um outro objeto pode se associar. No exemplo anterior, não há um limite – pelo menos na teoria – para a quantidade de pedidos que um cliente pode realizar.

Os símbolos Li e Ls devem ser substituídos por valores correspondentes aos limites inferior e superior do intervalo que se quer representar.

Ex.: Supondo uma corrida em que deve haver no máximo 6 velocistas participantes:

Uma lista de intervalos também pode ser especificada na multiplicidade de uma associação. Por exemplo, a multiplicidade "1, 3, 5..9, 11" é válida e significa que um objeto pode se associar com uma quantidade de objetos que esteja no conjunto {1, 3, 5, 6, 7, 8, 9, 11}. Os valores devem estar representados em ordem crescente e lidos da esquerda para a direita.

13.7. Conectividade

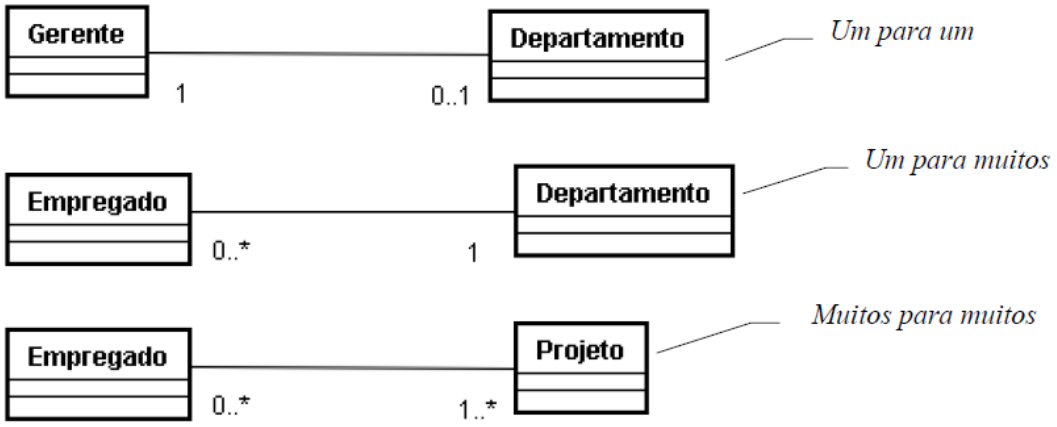
Existem infinitas possibilidades de associação entre objetos – todas as combinações possíveis entre os números inteiros. Essas associações podem ser agrupadas em três grandes tipos:

- Um para um
- Um para muitos
- Muitos para muitos

Denomina-se conectividade o tipo de associação entre duas classes. O tipo de conectividade da associação depende dos símbolos da multiplicidade que são utilizados na associação.

Conectividade	**Em um extremo**	**No outro extremo**
Um para um	0..1 1	0..1 1
Um para muitos	0..1 1	* 1..* 0..*
Muitos para muitos	* 1..* 0..*	* 1..* 0..*

Ex.: Tipos de conectividades



Neste exemplo, o primeiro caso indica que um gerente pode gerenciar nenhum ou 1 departamento. No segundo um empregado está lotado em um único departamento e um departamento pode possuir diversos empregados. O terceiro indica que um projeto pode ter diversos empregados e um empregado pode trabalhar em diversos departamentos.

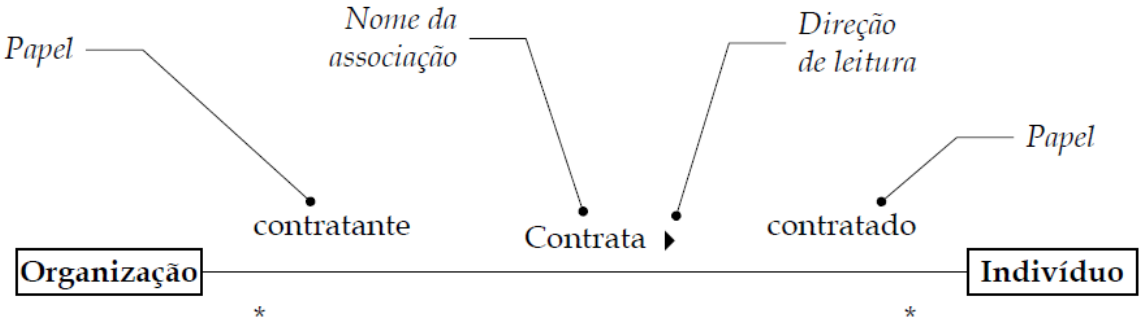
13.8. Nome da associação, direção de leitura e papéis

O nome da associação é posicionado, normalmente, acima da linha de associação e entre as classes envolvidas. Este nome deve ser escolhido de forma a dar significado semântico à mesma.

A direção de leitura indica como a associação deve ser lida e é representada por um pequeno triângulo próximo ao início ou final do nome da associação.

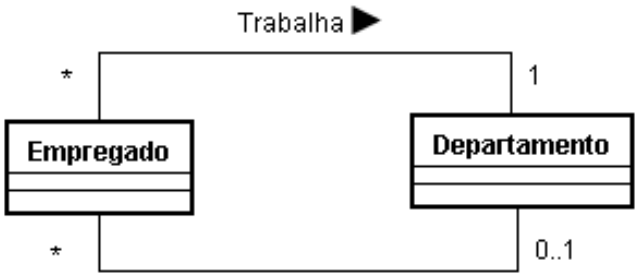
Uma característica complementar é a indicação de papéis para cada uma das classes participantes.

Sempre que o significado de uma associação não for fácil de inferir, é recomendável representar papéis ou pelo menos o nome da associação evitando que haja interpretações erradas do significado da associação e aumentando a sua legibilidade. Quando o significado for intuitivo, a utilização de papéis só serve para "carregar" o diagrama.



Pode haver diversas associações definidas entre duas classes em um diagrama de classes.

Ex.: Suponha um departamento que precisa saber quais são seus empregados e quem é seu gerente. Neste caso há duas associações diferentes.



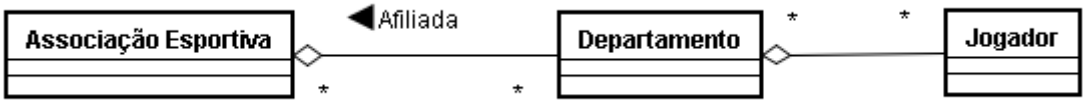
13.9. Agregação

Agregações são casos específicos de associações. Agregações são utilizadas para representar conexões entre objetos que guardam uma relação todo-parte entre si. A diferença entre agregação e associação é puramente semântica.

Pode ser aplicada a seguinte regra para verificar uma agregação. Sejam duas classes associadas, X e Y. Se uma das perguntas a seguir for respondida com um sim, provavelmente há uma agregação onde X é todo e Y é parte.

- X tem um ou mais Y ?
- Y á parte de X ?

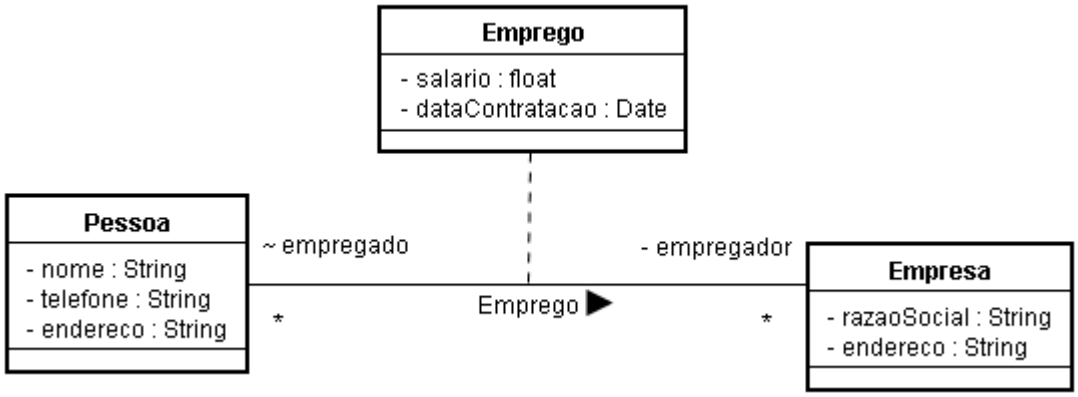
Uma agregação é representada por uma linha conectando as classes relacionadas, com um losango branco perto da classe que representa o todo.



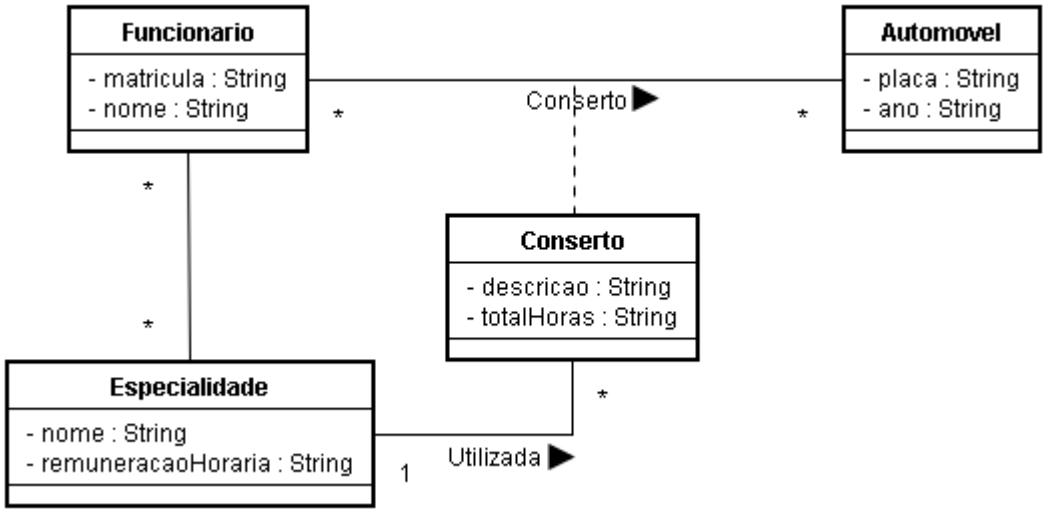
14 - DIAGRAMA DE CLASSE

14.1. Classes associativas

São classes que estão ligadas a associações em vez de estarem ligadas a outras classes. Este tipo de classe aparece normalmente quando duas classes estão associadas e precisa-se manter informações sobre a associação. Embora mais comumente encontrada em associações com conectividade muito para muitos, pode haver para qualquer tipo de conectividade. A ligação da classe associativa é feita através de uma linha tracejada.

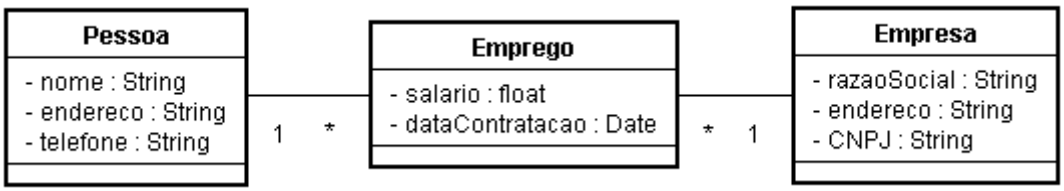


Uma classe associativa pode participar de outros relacionamentos além do qual ela faz parte como classe associativa.



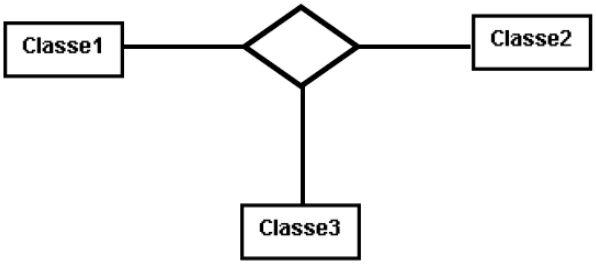
Podemos notar que uma classe associativa é um elemento híbrido pois tem característica de uma classe e de uma associação. Um diagrama de classes que contém uma classe associativa pode ser modificado para retirá-la sem perda de informação. A classe associativa é substituída por uma classe ordinária que deve estar associada no formato um para muitos com as classes que antes eram conectadas pela associação que continha a classe associativa.

Ex.:

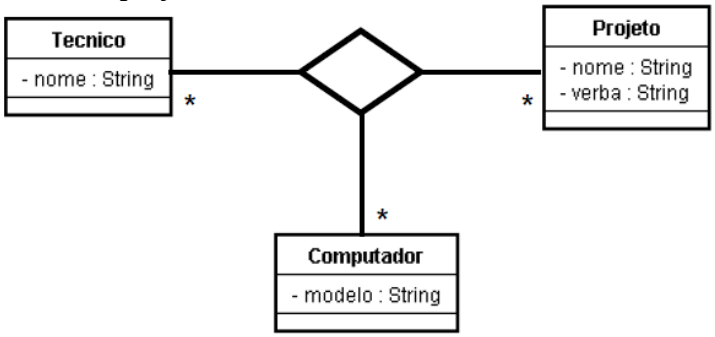


14.1.1. Associações Ternárias

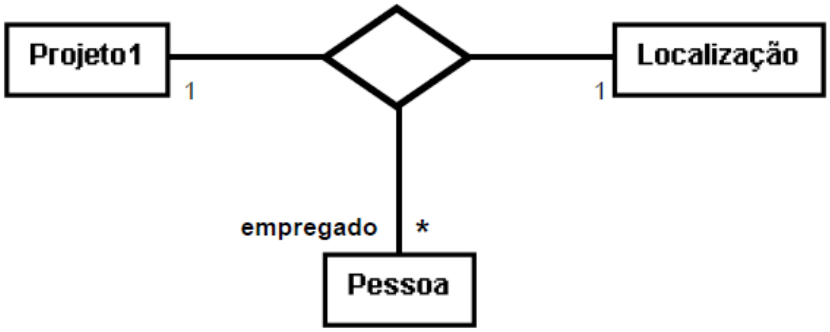
São associações necessárias quando é preciso associar objetos de três classes distintas. A notação UML para associação ternária é:



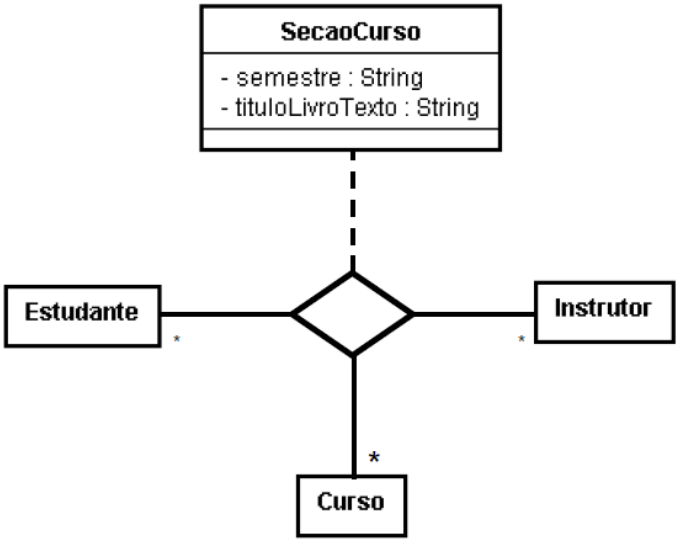
Ex.: O exemplo de associação ternária a seguir ilustra o fato que um técnico utiliza exatamente um computador para cada projeto em que trabalha. Cada computador pertence a um técnico para cada projeto. Um técnico pode trabalhar em muitos projetos e utilizar diferentes computadores para diferentes projetos.



Ex.: Outro exemplo de ternária registra que cada empregado associado a um projeto trabalha em somente uma localização, mas pode estar em diferentes localizações para projetos diferentes. Em uma localização particular pode haver muitos empregados associados a um dado projeto.



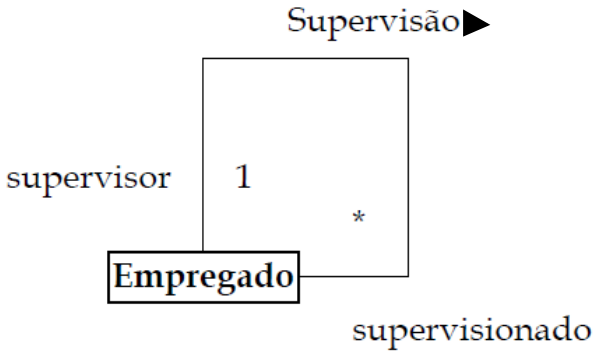
Estes conceitos podem ser utilizados em conjunto dando origem ao conceito de classe associativa ternária.



14.1.2. Associações reflexivas

Uma associação reflexiva ou auto-associação associa objetos da mesma classe. Cada objeto tem um papel distinto nesta associação. Neste tipo de associação, a utilização de papéis é bastante importante para evitar ambiguidades na leitura da associação. Um objeto não faz associação reflexiva com ele próprio, mas com outro da mesma classe dele.

Ex.: Considere uma associação reflexiva entre empregados. Nesta associação há objetos que assumem papel de supervisor e outros que assumem papel de supervisionado. No exemplo, um supervisor não é supervisor dele próprio, certo?



14.2. Identificando as Classes Iniciais

Um ambiente de desenvolvimento de software OO é composto de uma coleção de objetos que colaboram para realizar as tarefas desse sistema. Também sabemos que todo objeto pertence a uma classe. Daí, quando se fala na identificação das classes, o objetivo é descobrir quais objetos comporão o sistema.

A identificação das classes se divide em duas atividades:

1 – Identificar as classes candidatas, ou seja, entidades que podem se tornar classes. Identificadas, alguns princípios são aplicados para eliminar classes candidatas desnecessárias.

2 – Identificar as classes que comporão o sistema.

Estas atividades não são sequenciais. Elas se intercalam em ciclos repetitivos até a definição das classes do sistema.

Há dois métodos principais para realizar a identificação das classes de domínio de um sistema:

1 – Método dirigido a dados – a ênfase é dada na identificação dos conceitos relevantes para um domínio de negócio resultando em um modelo conceitual de sistema.

2 – Método dirigido a responsabilidades – a ênfase é dada na identificação das classes a partir dos seus comportamentos relevantes para o sistema. "O método dirigido a responsabilidades enfatiza o encapsulamento da estrutura e do comportamento dos objetos." (Wirfs-Brock e Wilkelson, 1989). Esta metodologia utiliza o princípio do encapsulamento e as responsabilidades sob foco são aquelas úteis externamente à classe. Este será o método a trabalhar em nosso curso.

14.3. Responsabilidades e Colaboradores

Uma responsabilidade é uma obrigação que um objeto tem para com o sistema no qual está inserido. É uma coisa que o objeto conhece ou faz sozinho ou com ajuda. Através de suas responsabilidades, um objeto colabora com outros objetos para que os objetivos do sistema sejam alcançados.

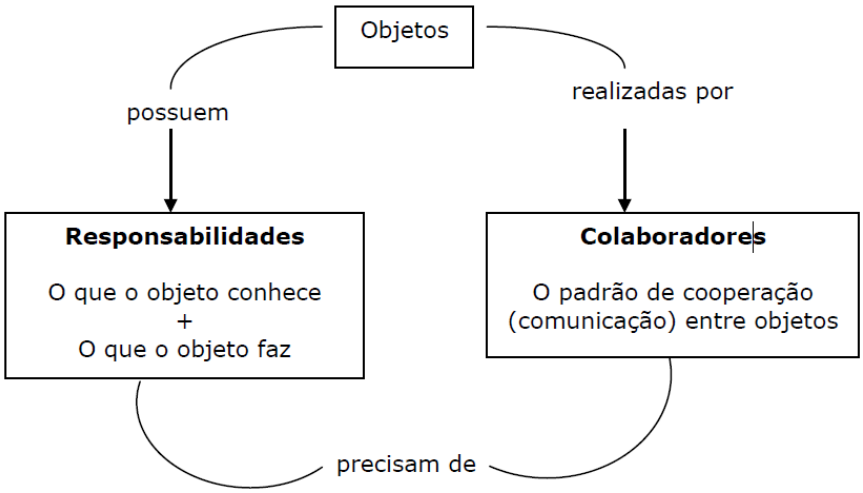
Ex.: Objeto Cliente – este objeto deve conhecer seu nome, endereço, telefone.

Objeto Pedido – conhece sua data de realização e deve fazer o cálculo de seu valor total.

Quando um objeto não pode cumprir uma ou mais responsabilidades sozinho, ele requisita colaborações de outros objetos para cumprir com sua(s) responsabilidade(s). Quando a impressão de fatura de um pedido é requisitada, um objeto Pedido pode ter a responsabilidade de fornecer o seu valor total, um objeto Cliente fornece seu nome, cada ItemPedido informa a

quantidade do produto correspondente e o valor subtotal, e os objetos Produto também colaboram fornecendo seu nome e preço unitário.

Cada objeto tem um conjunto de responsabilidades dentro do sistema. Algumas, o objeto cumpre sozinho, outras, necessitam da colaboração de outros objetos – os colaboradores.



14.4. Categorias de responsabilidades

Categorias são definidas de acordo com os tipos de responsabilidades atribuídas aos objetos em uma técnica chamada Análise de Robustez (Jacobson et al, 1992). Sob esta técnica os objetos são divididos em objetos de entidade, objetos de controle e objetos de fronteira.

14.4.1. Objetos de entidade

Um objeto de entidade é um depósito para alguma informação manipulada e representam conceitos do domínio do negócio:

A. normalmente existem várias instâncias da uma mesma classe de entidade coexistindo no sistema, como da classe Produto;

B. armazenam informações persistentes do sistema;

C. os atores não têm acesso direto a esses objetos, pois eles se comunicam com o exterior por meio de outros objetos;

D. normalmente participam de vários casos de uso. Um objeto Pedido pode participar dos casos de uso Realizar Pedido e Atualizar Estoque;

E. uma vez criado, esse objeto pode existir por diversos anos ou mesmo quanto o próprio sistema.

Responsabilidades de fazer típicas:

- Informar valores de seus atributos a objetos de controle;
- Realizar cálculos simples, normalmente com a colaboração de objetos de entidade associados através de agregação;
- Criar e destruir objetos-parte, em sendo um objeto-todo de uma agregação.

Responsabilidades de conhecer:

- São fáceis de identificar, pois são normalmente fornecidas por atores quando da criação desses objetos, por exemplo: nome, telefone e endereço do Cliente.

14.4.2. Objetos de fronteira

Objetos de interface com os atores (outros sistemas, equipamentos ou seres humanos). Traduzem os eventos gerados por um ator em eventos relevantes ao sistema. Responsáveis por apresentar resultados de interação dos objetos internos em algo que possa ser entendido pelo Ator. Por consequência, são altamente dependentes do ambiente.

Tipos de classes de fronteiras:

- Interface com usuário (atores humanos)
- Interface com outros sistemas
- Interface com dispositivos atrelados ao sistema (equipamentos)

Responsabilidades de fazer típicas:

- Notificar aos objetos de controle os eventos gerados externamente aos sistemas;
- Notificar aos atores o resultado de interações entre os objetos internos.

Responsabilidades de conhecer:

- Frequentemente representam informação manipulada através da interface com o usuário;
- Para interfaces com outros sistemas e equipamentos, as responsabilidades representam propriedades de interface de comunicação.

14.4.3. Objetos de controle

Objeto de controle ou controlador servem como uma ponte entre objetos de fronteira e objetos de entidade. São responsáveis por controlar a lógica de execução correspondente a um caso de uso. Servem como "gerentes" dos outros objetos para a realização de um ou mais cenários de um caso de uso. Representam a lógica de um caso de uso, traduzindo eventos (clique do mouse) em operações que devem ser realizadas pelos demais objetos (de fronteira ou de entidade).

Ao contrário de objeto de fronteira e de entidade, objetos de controle são tipicamente ativos – consultam informações e requisitam serviços de outros objetos. Como os objetos de fronteira, os de controle têm vida curta, normalmente existem somente durante a realização de um caso de uso. São bem mais efêmeros que os objetos de entidade, normalmente participando de um caso de uso.

Responsabilidades de fazer típicas:

• realizar monitorações a fim de responder a eventos externos ao sistema (gerados pelos objetos de fronteira);

• coordenar a realização de um caso de uso através do envio de mensagem a objetos de fronteira e objetos de entidade;

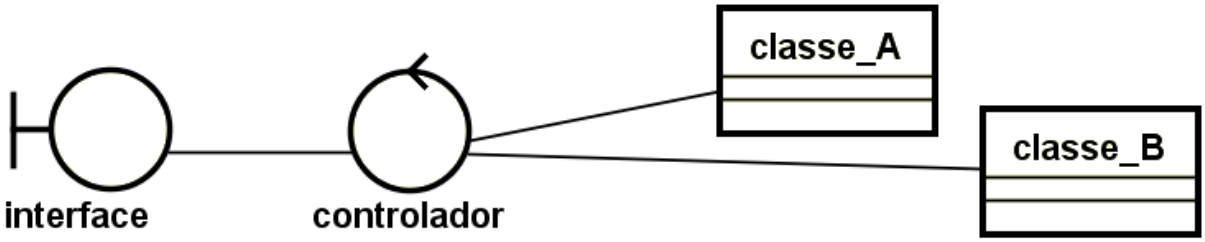
• assegura que as regras de negócio estão sendo seguidas corretamente;

• coordenar a criação de associações entre objetos de entidade.

Responsabilidades de conhecer:

• manter valores acumulados ou derivados durante a realização de um caso de uso;

• manter o estado de realização do caso de uso.

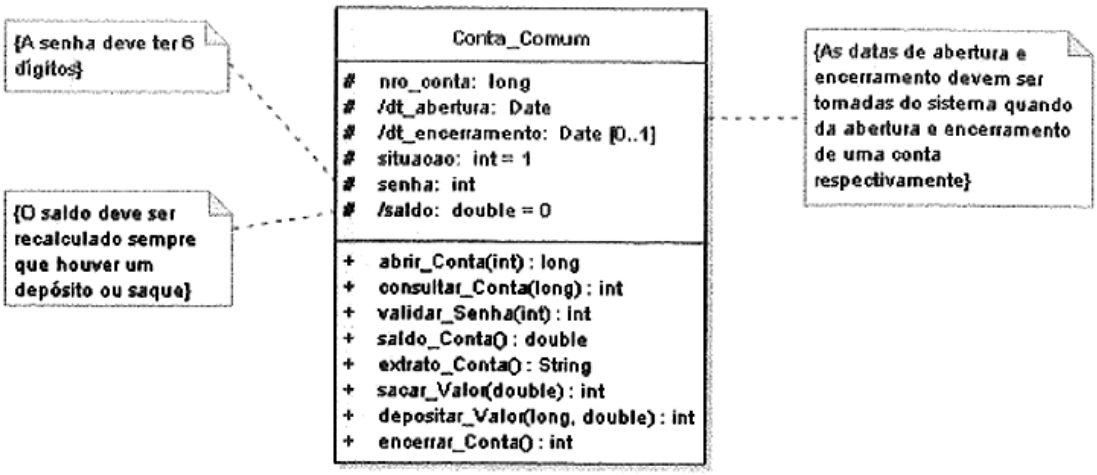


Exemplo de Diagrama de Classes com objetos de Fronteira e Controle

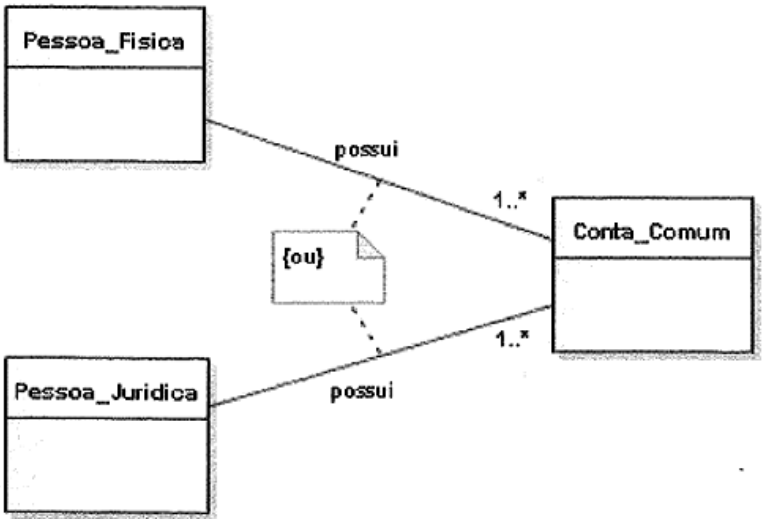
14.5. Restrições

Constituem-se em informações adicionais que definem condições a serem validadas durante a implementação dos métodos de uma classe, das associações entre as classes ou mesmo de seus atributos. Elas são representadas por textos limitados por chaves e podem ser utilizadas para detalhas requisitos funcionais, incluindo regras de negócio.

São alguns exemplos:



A barra "/" que antecede os atributos dt_abertura, dt_encerramento e saldo, indica que os mesmos sofrem algum tipo de cálculo.



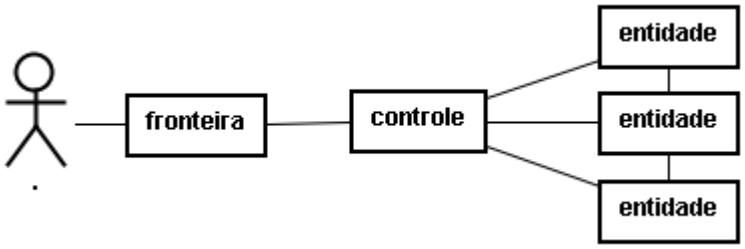
14.6. Divisão de Responsabilidades

A categorização estudada implica que um objeto é especialista em executar um dos três tipos de tarefa. A importância da categorização está relacionada à capacidade de adaptação do sistema a eventuais mudanças. Se cada objeto tem funções específicas dentro do sistema, possíveis mudanças no sistema podem ser menos complexas e mais localizadas.

Tipo de Mudança	Onde Mudar
Mudanças na interface do sistema, ou em relação à interface gráfica, ou em relação à comunicação com outros sistemas.	Objetos de fronteira
Mudanças nas informações manipuladas pelo sistema	Objetos de entidade
Mudanças em funcionalidades complexas (que envolvem a coordenação de vários objetos)	Objetos de controle

Além de concorrer para o desacoplamento entre elementos facilitando mudanças, a separação de responsabilidades facilita também o reuso dos objetos em sistemas de software semelhantes.

"Uma boa modelagem de classes deve separar a informação contida no domínio de negócios (objetos de entidade), as várias maneiras de visualizar uma informação (objetos de fronteiras) e a lógica da aplicação (objetos de controle)". (Bezerra, 2002)



14.7. Identificando Classes

Na análise dos casos de uso, a descrição textual de cada um deles é estudada para se identificar classes candidatas. Todos os fluxos de caso de uso: principal, alternativos e de exceção, são analisados. A partir desta análise, as classes vão sendo identificadas.

Na identificação de classes, os substantivos e locuções equivalentes a substantivos são considerados, sinônimos removidos, permanecendo o nome mais significativo no domínio do negócio. Os nomes que permanecem correspondem a classes candidatas. Nome de ator deve ser removido da lista de classes candidatas se não for necessário que o sistema mantenha informações sobre ele.

Para responsabilidades de fazer devem-se identificar os verbos que representam ações. A classe na qual essa responsabilidade deve ser definida pode ser escolhida através do sujeito da frase em que o verbo é utilizado. Informações necessárias à realização dessa responsabilidade, geralmente o complemento da frase, são normalmente outras classes (colaboradores) ou são responsabilidades de conhecer (atributos).

A vantagem da análise dos casos de uso é que esta abordagem é bastante simples. Para cada caso de uso, nomes são destacados para detectar classes candidatas.

A desvantagem é que abordagem depende da descrição do caso de uso estar completa e do estilo linguístico que pode gerar muitas classes candidatas ou não deixar explícita uma classe importante para o sistema.

Para contornar, uma solução em dois passos é sugerida:

1 – Procede-se a abordagem acima;

2 – Aplica-se a modelagem CRC.

14.8. Modelagem CRC

A modelagem CRC – Classes, Responsabilidades e Colaboradores, que não faz parte da UML, se baseia no paradigma OO em que objetos cooperam uns com os outros para que uma tarefa do sistema seja realizada. Sua simplicidade de notação a tornou particularmente interessante na identificação de classes de domínio.

Nome da classe (especialidade)	
Responsabilidades	**Colaboradores**
1ª responsabilidade 2ª responsabilidade ... n-ésima responsabilidade	1º colaborador 2º colaborador .. n-ésimo colaborador

Os nomes escolhidos para as classes devem estar no singular, já que uma classe representa vários objetos, e iniciando com letra maiúscula. Todas as classes devem receber nomes provenientes do domínio do negócio. Um controlador deve ter um nome que lembre o caso de uso pelo qual ele é responsável (Ex.: ControladorInscrição, ControladorReservasCarros,...). Já os nomes de classes de fronteiras e de entidades não devem conter verbos nem sugerir ações (Ex.: Pedido, ItemPedido, LeitoraCartões,...).

ContaBancária (entidade)	
Responsabilidades	**Colaboradores**
▪ Conhecer o seu cliente. ▪ Conhecer o seu número. ▪ Conhecer o seu saldo. ▪ Manter um histórico de transações. ▪ Aceitar saques e depósitos.	Cliente HistóricoTransações

As responsabilidades atribuídas a cada classe também devem receber nomes provenientes do domínio do negócio – verbo + complemento.

Para CRC, resumimos:

• selecionar um conjunto de cenários de casos de uso;

• para um dos cenários:

i. examinar a sua sequência de passos para identificar as responsabilidades do sistema em relação a cada um dos passos;

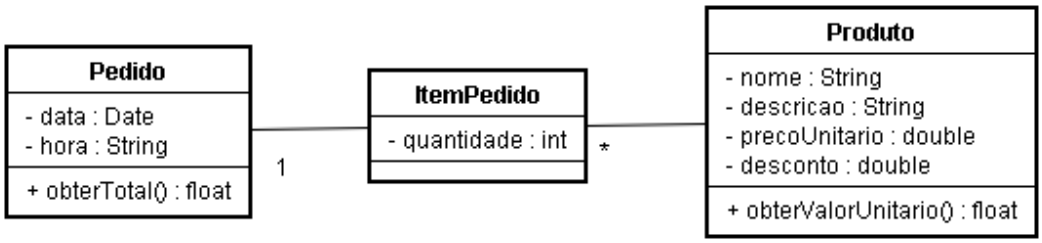
ii. identificar classes relevantes que devem cumprir com as responsabilidades;

• repetir o passo 2 para o próximo cenário e modificar a alocação de responsabilidades e a definição de classes.

Dependendo da especialidade da classe: fronteira, controle ou entidade, há um conjunto típico de responsabilidades com as quais ela deve cumprir.

A inteligência do sistema deve ser uniformemente distribuída, ou seja, o conjunto de responsabilidades do sistema deve ser distribuído o mais uniformemente possível pelas classes.

Ex: O objeto Pedido deve conhecer seu valor total. Para isso, cada itemPedido fica responsável por calcular o seu total (sub) e informar ao objeto Pedido para que lê faça a totalização. Por sua vez, o item Pedido pede ajuda a Produto para que esta informe o seu valor unitário.



A situação pode ser resumida da seguinte forma:

- Pedido sabe seu valor total
- ItemPedido sabe seu subtotal
- Produto sabe o seu valor unitário

As responsabilidades conceitualmente relacionadas devem ser mantidas em uma classe: informações pertinentes a um cliente devem estar definidas em classe Cliente. É mais adequado criar as classes Pedido e Fatura em vez de uma única classe para manter as informações sobre os dois conceitos.

Responsabilidades redundantes não devem ser criadas. Se duas ou mais classes precisam de alguma informação, deve-se verificar a opção de criar uma nova classe ou escolher uma das classes preexistentes para manter a informação. A informação deve ficar em lugar único e os outros objetos devem enviar mensagens ao objeto que possui informação para obtê-la.

14.9. Construção do Modelo de Classes de Domínio

Após a identificação de classes e atribuições de responsabilidades, o modelador deve realizar a verificação da consistência entre as classes para eliminar incoerências e redundâncias.

Para cada classe o modelador deve estar apto a declarar as razões de existência de cada classe identificada e, para cada classe, argumentar sobre a existência de cada uma das suas responsabilidades e de cada um de seus colaboradores.

14.10. Atributos de uma classe

Normalmente uma propriedade de conhecer é mapeada para um ou mais atributos. Os atributos de uma classe estão relacionados com o tipo de responsabilidade (fronteira, controle ou entidade) atribuídos a ela. Após o mapeamento, deve-se verificar se cada atributo possui as seguintes características:

- Guarda valor atômico: RS10.00, 100, 5.35, “MSIS”.
- Não contém estrutura interna. Se uma responsabilidade de conhecer possui uma estrutura interna relevante para o sistema, e se pertence ao domínio do negócio, considere modela-la como uma classe.
- Aplica-se a todos os objetos de uma classe: cada classe deve fazer sentido para todo e qualquer objeto dessa classe. A classe Pessoa deve ter o atributo Salário?

Um dos erros mais comuns é mapear uma responsabilidade de conhecer como atributo, quando melhor seria como associação. Por exemplo, o atributo nomeCliente na classe Pedido.

Uma mesma classe pode atributos diferentes em sistemas diferentes. Por exemplo, quais seriam os atributos para a classe Pessoa em um sistema para suspeitos de crimes e para um sistema de cadastro de funcionários?

14.11. Definição de associações e agregações

Se uma classe possui colaboradores, é porque existem relacionamentos entre eles. Um objeto precisa conhecer o outro para lhe fazer requisições. Para criar associações, verifique os colaboradores de uma classe. Para associações reflexivas, ternárias e agregações o modo é o mesmo. Na agregação, a classe é o todo e o colaborador é a parte.

Classes associativas normalmente surgem a partir de responsabilidade de conhecer que o modelador não conseguiu atribuir a uma classe. A solução é criar uma classe associativa para que ela cumpra a tal responsabilidade. Por exemplo, suponha que pessoas trabalham em projeto formando uma conectividade muitos para muitos. Na necessidade de se saber quantas horas semanais cada pessoa trabalha em cada um dos projetos, onde ficaria essa informação (atributo)?

14.12. Documentando o modelo de classes

Após classes, responsabilidades e colaboradores terem sido mapeados para elementos do modelo de classe, esses elementos devem ser organizados em um diagrama e documentados, resultando no modelo de classes de domínio.

Na diagramação, as classes devem ser posicionadas de tal forma que as associações sejam lidas da esquerda para a direita.

No diagrama de classes, também podem ser associados estereótipos predefinidos na UML às classes identificadas de acordo com a categoria: <<fronteira>>, <<controle>> e <<entidade>>.

A construção de um único diagrama de classes para todo o sistema pode resultar em um diagrama bastante complexo. Uma alternativa é criar uma **visão de classes participantes** (VCP) para cada caso de uso. Nessa forma, a elaboração de cada caso de uso resulta em uma VCP.

Além do diagrama, as classes devem ser documentadas textualmente. Inicialmente pode-se utilizar o próprio formato CRC como documentação com uma definição - duas ou três frases - para cada classe, assim como para cada atributo cujo significado não seja fácil de verificar pelo nome dado ao atributo.

Da mesma forma que para casos de uso, é útil adicionar uma seção de implementação. Embora detalhes de implementação não façam parte e nem devam ser considerados no modelo de classes de domínio, esses detalhes inevitavelmente surgem na mente dos analistas enquanto o modelo está sendo construído. Esta seção funciona como um depósito de ideias e soluções relativas ao modelo de classes que podem ser utilizadas nas atividades futuras do desenvolvimento.

Conclusão

As metodologias de desenvolvimento de sistemas são essenciais para a construção de um sistema que realmente atenda às necessidades e os requisitos estabelecidos pelo cliente. A escolha da metodologia adequada à realidade da equipe de análise e desenvolvimento irá determinar, em parte, o sucesso do projeto. Por isso, conhecer todas as técnicas e métodos de análise disponíveis e se atualizar no que há de novo no mercado tecnológico é imprescindível ao analista de sistemas. O processo de análise vai além de uma simples entrevista e uma posterior montagem do sistema. A documentação é o item mais importante em qualquer método de análise, pois, nela, estará o resultado de todo o trabalho de análise e todas as instruções para a construção do software. Por isso, a má documentação acarretará um software que não atenderá os requisitos do cliente e isto ocasionará o fracasso do sistema. E ainda, uma metodologia inadequada também levará a equipe a não se comunicar de forma correta, formando desta forma um software que não atenderá às necessidades do cliente. Cabe ao analista conhecer e saber aplicar corretamente uma metodologia para que seu projeto seja um sucesso e toda a equipe consiga desenvolver o sistema com eficiência e qualidade.

Referência bibliográfica

BEZERRA, Eduardo. Princípios de Análise e Projeto de Sistemas com UML, 2/E. Rio de Janeiro: Editora Campus, 2006.

CIOTI, Leonardo & ROSA, Aline. Apostila de Introdução ao Estudo de Análise. Rio de Janeiro, 2010.

De MARCO, T. Análise Estruturada e Especificação de Sistemas. Rio de Janeiro: Campus, 2001.

GANE, Chris. Análise Estruturada de Sistemas. RJ: LTC. 1999.

GILLEANES, T. A. Guedes. UML - Uma Abordagem Prática. 3. ed. Editora Novatec, 2008.

OLIVEIRA, Jayr Figueiredo de. Metodologia para desenvolvimento de Projeto de Sistemas. 4ª edição, São Paulo: Érica, 1999.

POMPILHO, S. - Análise Essencial - Guia Prático de Análise de Sistemas - IBPI Press, 1995.

SILVA, Alex de Araujo. Metodologia e projeto de software orientados a objetos. 1ª edição, São Paulo, Érica, 2003

YOURDON, E. Análise Estruturada Moderna. Rio de Janeiro: Campus, 1990.

YOURDON, E. e CONSTANTINE, Larry L. - Projeto Estruturado de Sistemas. Rio de Janeiro: Editora Campus, 1990.