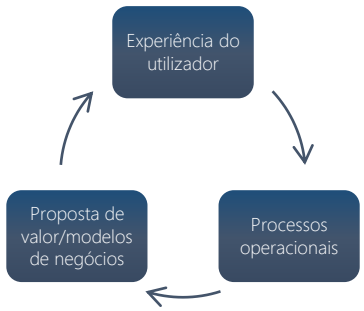
1. Introdução à Modelação

Desenvolvimento de sistemas de informação: impacto no negócio, sintomas e "terapeuta"



Sistemas

Transformação Digital: Para melhorias no desempenho de uma empresa, faz-se uso dos TIC.

Dados vs. Informação

Os **dados** são os factos em cru, ou seja, a observações ou as medidas sobre a realidade.
O **conhecimento (do domínio) e**xplica os padrões que ocorrem nos dados. É a compreensão das relações entre os dados.
A **informação** é obtida no processamento de dados. É a coleção de factos organizados de maneira a que apresentam mais valor, para além dos factos em si.

Experiência do cliente/utente	Processos operacionais da organização	Modelo de negócio (produtos/ serviços de base tecnológica)
Estudar o cliente	Desmaterialização de processos	Extensão do negócio para o digital
Mais tecnologia no momento de venda /compra	Maior flexibilidade no posto de trabalho	Novo modelo de negócio de base tecnológica
Canais de contacto com o cliente	Gestão informada do desempenho/ indicadores	Globalização das operações e parcerias

O que é um sistema de informação?

Um **Sistema é** um conjunto de componentes/partes que interagem para atingir uma finalidade. Tem uma estrutura interna que transforma inputs em outputs, para um fim específico.
Um **Sistema de Informação** é um conjunto de recursos interrelacionados (humanos e tecnológicos) para satisfazer as necessidades de informação de uma organização e dos seus processos de negócio.

Análise: O processo de caracterização do problema/ produto

Sintomas da indústria de engenharia de software

- Utilizadores descontentes
- Mau desempenho em condições de pico
- Equipas que não comunicam
- Dificuldades em controlar versões do software
- Necessidades sem respostas
- Requisitos que vão sendo alterados
- Módulos que não integram
- Dificuldades em manter e evoluir
- Erros revelados

Boas Práticas

- Desenvolver de forma iterativa
- Gestão explícita de requisitos
- Usar arquiteturas de componentes
- Usar modelos visuais
- Práticas de verificação contínua da qualidade
- Gestão explícita da mudança

Modelação: Linguagem visual de especificação

Os modelos ajudam a gerir a complexidade

Ajudar a visualizar um sistema, como se pretende que venha a ser.
Especificar a estrutura e o comportamento do sistema (antes de implementar).
Serve como referência para a construção ("planta").
Documentam as decisões (de desenho) que foram feitas.

Modelação visual ajuda no desenvolvimento

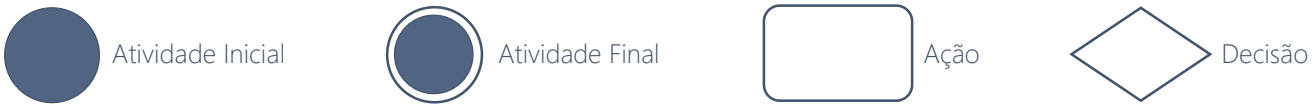
Linguagem de modelação normalizada: UML 2: Unified Modeling Language

Benefícios

- Promover a comunicação mais clara e sucinta.
- Manter o desenho (planeamento) e a implementação (construção) coerentes.
- Mostrar ou esconder diferentes níveis de detalhe, conforme apropriado.
- Pode suportar, em parte, processos de construção automática (gerar a solução a partir do modelo).

2. UML Diagrama de Atividades

Elementos do Diagrama de Atividades



Os diagramas de atividade mostram o fluxo de ações (e de dados)

Quando aplicar?

Modelar fluxos de trabalho/processos de negócio.

Descrever um algoritmo complexo.

Descrever a sequência de interações entre atores e o sistema sob especificação, num caso de atualização.

Pode ser usado para descrever os processos organizacionais existentes/novos:

Neutro em relação à programação.

Bom a captar papéis.

Pode captar o fluxo de dados também.

3. Análise por objetos e modelo do domínio

Análise por objetos: Objetos e Classes

Desenvolvimento por objetos

É uma estratégia para simplificar o espaço do problema, modularizando-o. É um esquema mental comum à análise de requisitos e programação. Facilita à reutilização de software.

Os objetos (em OO) modelam entidades do mundo real/espaço do problema:

Observáveis no mundo físico Exemplo: Aluno, Avião.

Conceitos Exemplo: Venda, Reserva.

Abstrações próprias do software Exemplo: Lista ligada, Vetor.

Objetos = **propósito** + **estado** + **comportamento**

Um objeto é uma entidade com uma fronteira bem definida, que encapsula **estado** e **comportamento**. Cada objeto tem a sua **própria identidade**, única, mesmo que o seu estado seja igual ao de outro objeto.

O estado é representado através de atributos e relacionamentos. O estado de um objeto corresponde a uma das condições/ configurações a que é possível o objeto apresentar-se. É normal o estado do objeto mudar ao longo do tempo.

O comportamento é representado através de operações. O comportamento define como é que o objeto age/reage. O comportamento visível/exposto é modelado pelo conjunto de mensagens a que responde (operações).

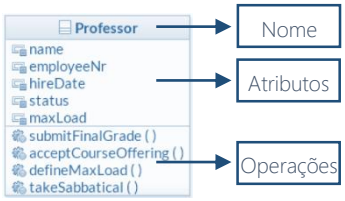
Objetos = **estado** + **operações**

Sabe os seus atributos e os seus relacionamentos. Sabe o seu comportamento, ativado através de operações.

O objeto é uma instância (ocorrência) de uma classe

Uma classe é uma categoria de objetos semelhantes, que partilham os mesmos atributos, operações, relacionamentos e semântica. É uma abstração, categoriza objetos semelhantes e enfatiza as características de interesse (e suprime outras).

Representação de classes em UML



A relação entre classes e objetos

Uma classe é uma definição abstrata de um objeto, define a estrutura e comportamento de cada objeto daquela classe/categoria. Funciona como um molde para criar objetos. **As classes não são coleções de objetos**.

Um atributo é uma propriedade de uma classe que descreve a gama de valores que as instâncias podem deter. Uma classe pode ter vários ou nenhum atributo.

Uma operação é a implementação de um serviço/ação que pode ser pedida a qualquer objeto de uma classe. É o que a classe sabe fazer. Uma classe pode ter várias ou nenhuma operação. Comuns: comandos e interrogações.

Uma associação é a relação semântica entre dois ou mais classificadores que especifica as ligações existentes entre as suas instâncias. Uma relação estrutura que mostra que um tipo de objetos estão ligados a outro tipo de objetos.

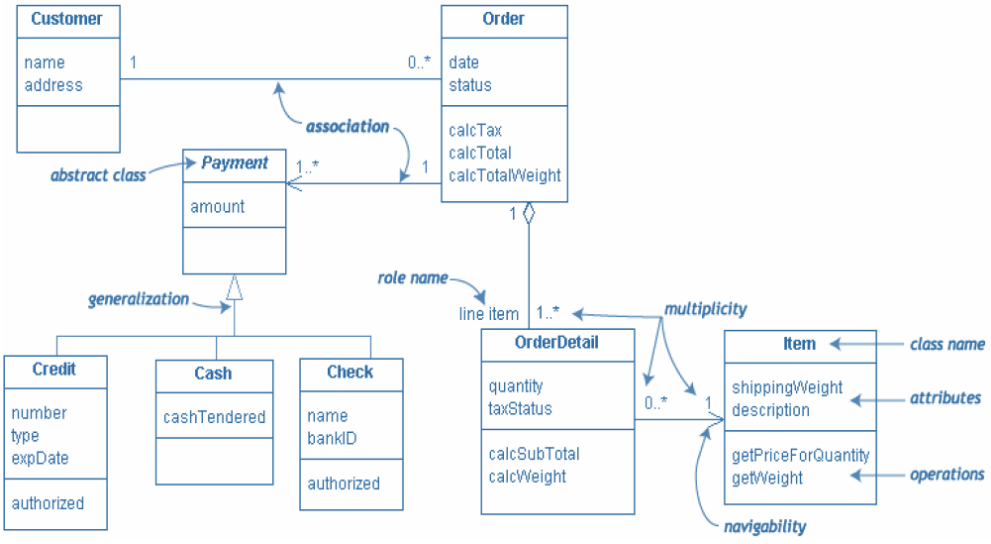
Multiplicidade é o número de instâncias de uma classe que se relacionam com uma instância da outra. Para cada associação, avaliar cada um dos extremos.

Agregação é uma forma especial de associação que modela uma relação de todo-parte, entre o agregador e as suas partes constituintes. Pode ler-se "**É parte de**...". A multiplicidade é usada como em outras associações.

Negavilidade indica a possibilidade de navegar de uma classe de partida para uma classe de chegada, usando a associação.

A relação entre classes em que uma especializa a estrutura e/ou comportamento de outra, partilhando todas as características designa-se generalização. Define uma hierarquia em que a subclasse herda das características da superclass (A subclasse pode sempre ser usada onde a superclasse é usada, mas não ao contrário). Pode ler-se "**é um tipo de**".
A subclasse herda os atributos, operações e relacionamentos da superclasse. A subclasse pode adicionar mais atributos, operações e relacionamentos à base herdada. Redefine as operações da superclasse. A herança põe em evidência as características comuns entre classes.

Síntese da notação do diagrama de classes



Mapa de conceitos de um problema: Modelo do Domínio

O modelo do domínio é um mapa para os objetos:

Mostra os conceitos de um problema (**dicionário visual**). Tem uma perspetiva do cartógrafo, mostrar que existe usando os nomes que a população utiliza. Não é um software.

O modelo de domínio em UML é representado com um diagrama de classes **sem operações**: não tem implementação. Pode representar objetos/conceitos do domínio, associações entre esses conceitos e atributos.

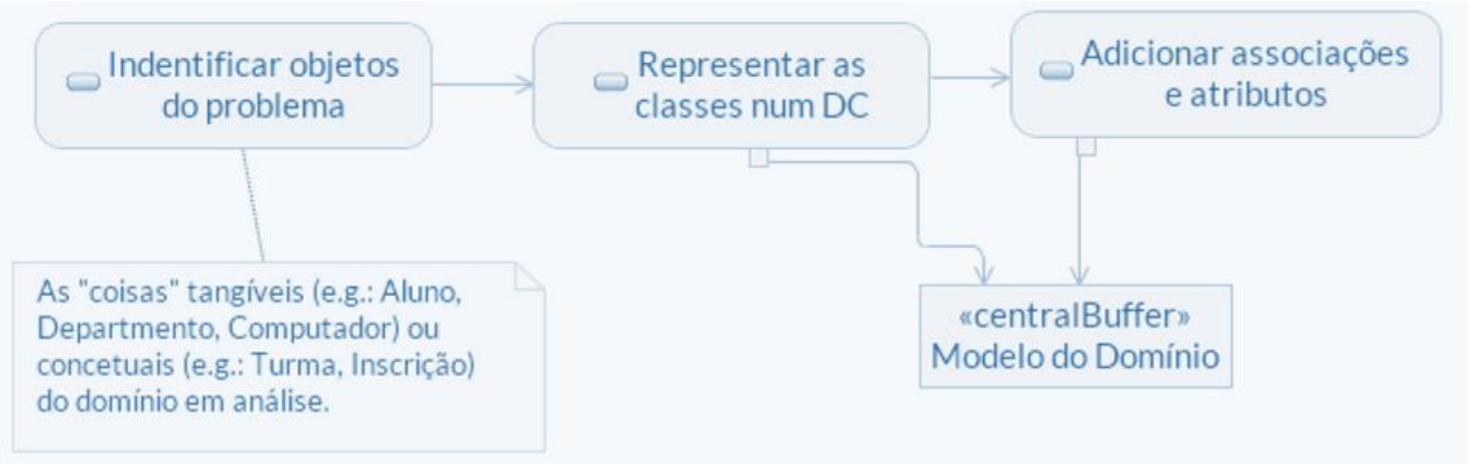
Classes de análise (Assunto de MAS)

- Resultado da análise dos requisitos (analista)
- Neutro em relação à implementação
- Não fornece diretamente o modelo de dados nem classes de programação

Classes em Java (Assunto de POO (e, em parte, de MAS))

- Resultado do desenho/implementação (programador)
- Escritas numa linguagem concreta (OO)

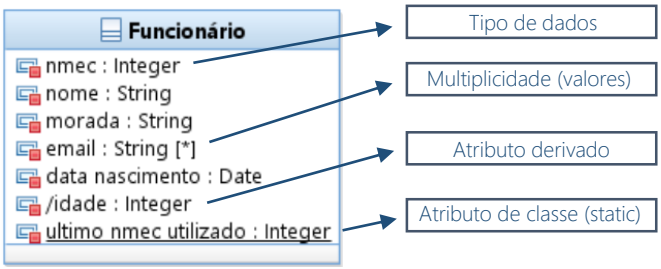
Fluxo para criar o Modelo de Domínio



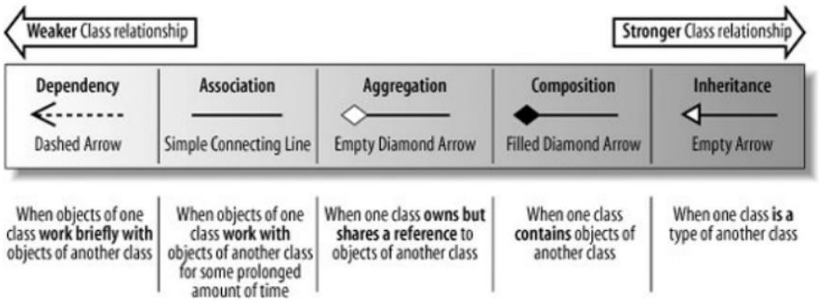
É possível encontrar as classes: Procurando numa lista de situações comuns, explorando documentos/relatórios existentes na área do problema e analisando os nomes.

4. Modelo do domínio com classes da UML

Especificação dos atributos



Cinco tipos diferentes de relacionamento entre classes com diferentes níveis de ligação



Associação entre conceitos

Descoberta

- Requisitos na forma [entidade][verbo]{entidade]
- Formulários/relatórios da área do problema ligam informação de várias entidades
- As anotações também são um elemento de modelação

Práticas

- A generalidade das associações do Modelo de Domínio são binárias
- As associações devem ter um nome para clarificar a interpretação
- Por convenção, lê-se de cima para baixo, da esquerda para a direita

Conceitos:

- Atributos de classe são partilhados por todas as instâncias.
- Associação reflexivas relacionam instâncias da mesma classe.
- Classes-associação captam a informação que descreve o relacionamento.
- Indicações para o uso de uma classe-associação no modelo de domínio
  1. Um atributo está relacionado com (a ocorrência de) uma associação.
  2. As instâncias da classe-associação têm um tempo de vida dependente da associação.
  3. Há uma relação de N:M entre dois conceitos e informação que carateriza a própria associação.
- Uma classe abstrata não é instanciada diretamente.
- Classes abstratas facilitam implementações parciais (partes comuns na superclasse).
- No conceito de programação as interfaces são contratos sem implementação e sem estado..
- Modelo de domínio não é o modelo de uma base de dados.

5. Requisitos com Use Cases

Os atores interatuam com o sistema para realizar objetivos

Caso de utilização (CaU)

O caso de utilização capta quem (ator) faz o quê (interação) com o sistema, com que fim (objetivo)

É uma sequência de ações que um sistema executa e que produzem um resultado com valor para algum ator em particular. Implica, na análise de requisitos, o foco no utilizador do sistema e nos episódios de uso, foco na compreensão daquilo que os atores consideram um resultado de valor. Conjunto de cenários relacionados com o mesmo objetivo. Um episódio de utilização mais as variantes.

As entidades-chave do modelo de CaU

Ator: Qualquer entidade (o papel de alguém, outro sistema) externa ao sistema sob especificação, que interage com este.

Cenário: Uma situação/história particular de utilização do sistema, isto é, um caminho possível na execução de um caso de utilização.

Fluxo do CaU

O guião para o diálogo ator/sistema diz-se um **fluxo-tipo**. Vários fluxos alternativos: significam variações "normais", casos particulares/incomuns, situações de erro e respetivo tratamento.

Narrativa

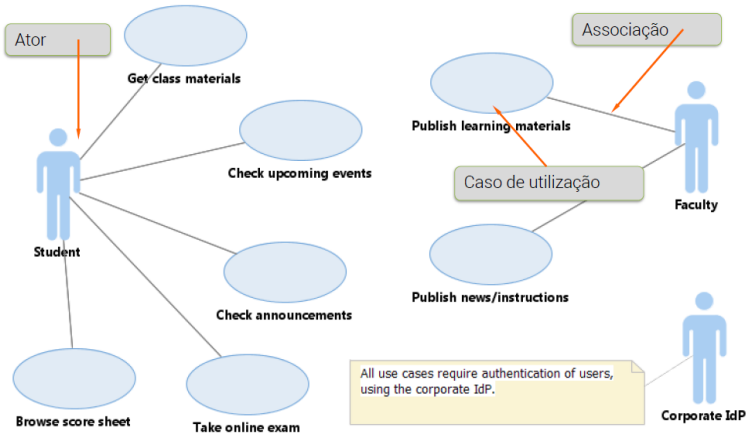
1. Descrever como é que o CaU começa e acaba.
2. Descrever o fluxo de eventos que provocam as ações/reações Descrever apenas o cenário "dentro" deste CaU. Não incluir fluxo de outros, nem comportamento externo ao sistema.
3. Clarificar a troca de informação entre Atores e Sistema.
4. Evitar designação vagas, ambíguas.
5. Verificação de qualidade.

Como encontrar os CaU?

1. Identificar a fronteira do sistema.
2. Identificar os atores que, de alguma forma, interagem com o sistema.
3. Para cada ator, identificar os objetivos/motivações para usar o sistema.
4. Definir os CaU que satisfazem os objetivos dos atores, dar nomes que refletem a motivação do ator.

Nota: Os casos de utilização podem ser agrupados em pacotes

Elementos do diagrama de casos de utilização



Reutilização de comportamento com *include*

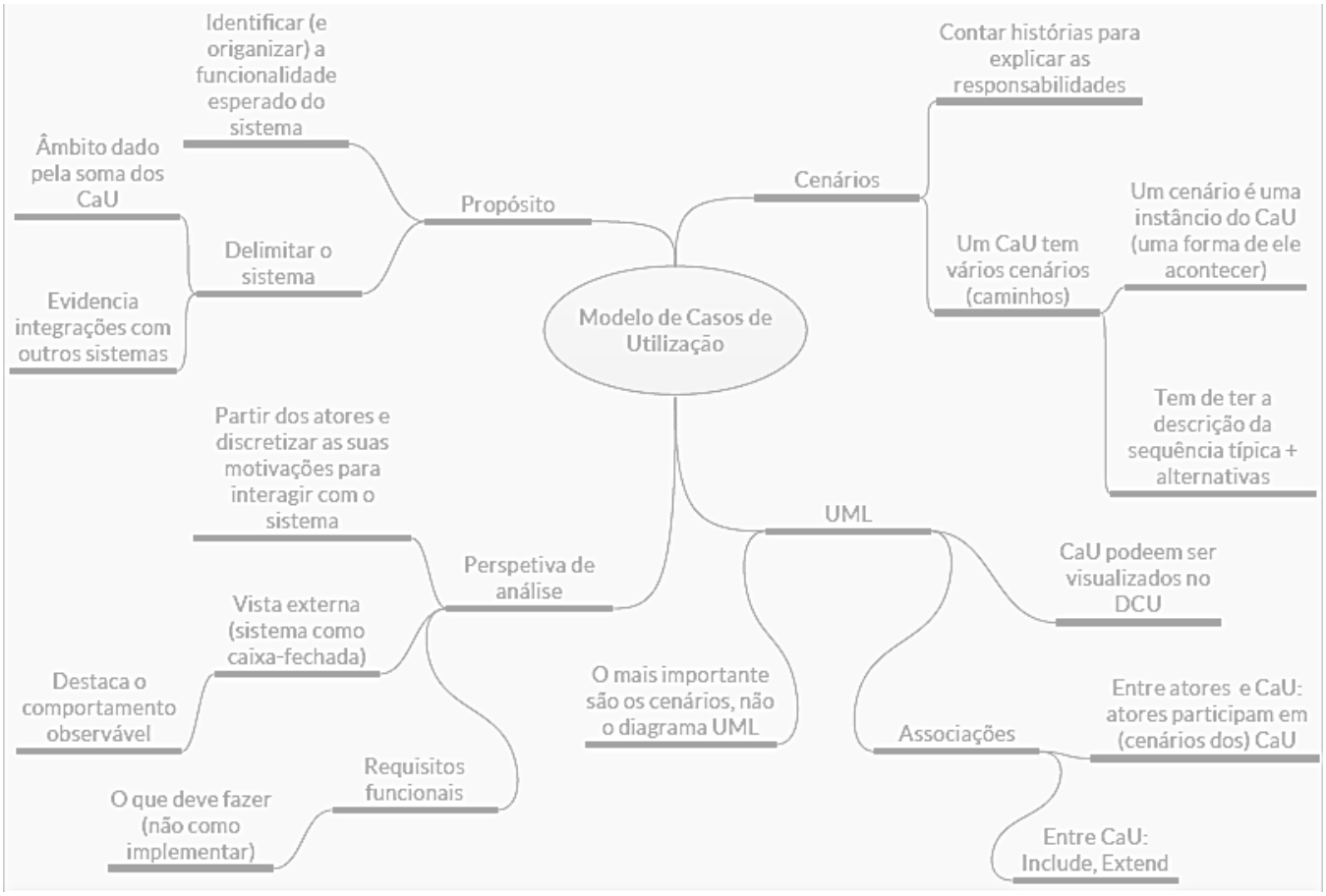
A **inclui** B: o comportamento de A incorpora o comportamento de B, facilita a factorização e a reutilização.

Ativação de comportamento opcional com *extend*

A **estende** B: O comportamento de B pode incorporar o comportamento em A, dependendo da verificação de uma "condição de extensão".

Para que serve o modelo de casos de utilização?

- Uma vista de um sistema que destaca o comportamento observável, tal como é percecionado pelos utilizadores.
- O modelo de casos de utilização divide a funcionalidade do sistema em episódios relevantes para os utilizadores/atores. Capta os problemas/motivações que levam à utilização do sistema.
- O modelo capta os requisitos do sistema ("o quê"), não a implementação da solução ("o como").
- A UML fornece uma visualização, mas o mais importante é a descrição dos cenários.



6. Casos de utilização

Especificação de requisitos através de cenários de utilização

Atributos de Qualidade

Requisitos que têm de ser selecionados:

- Requisitos essenciais vs facultativos
- Capacidade vs Orçamento vs Tempo
- Enquadramento regulamentar

Os atributos de qualidade são necessários para definir o produto. Um sistema tem funcionalidade e atributos de qualidade:

Requisitos Funcionais

- Captam o comportamento pretendido do sistema. São expressos como serviços, funções ou tarefas que o sistema deve realizar.
- Pode ser captado nos CaU.
- Pode ser descrito com diagramas de comportamento: atividades, sequência.

Requisitos Não Funcionais

- Restrições globais num sistema de software (rebustez).
- Também se designam atributos de qualidade.
- Por regra, não afetam apenas um módulo/CaU.

Os requisitos são muitas vezes listados "**o sistema deve...**"

#	Requisitos
RF. 1	O sistema deve permitir a um profissional criar um novo pedido de adesão, em auto-serviço, na web.
RF. 2	O sistema deve enviar credenciais temporárias para os pedidos de adesão e enviá-las, por e-mail, aos solicitantes.
RF. 3	O sistema deve permitir a pesquisa de cheques-dentista (emitidos) por número de utente do SNS.
RNF. 1	As pesquisas de cheques-dentista têm de retornar resultados em <5 segundos ou um evento de tempo expirado.

A apresentação de requisitos segue um modelo: Norma **ISO/IEC/IEEE 29148:2011**: Systems and software engineering -- Life cycle processes -- Requirements engineering.

CaU contam histórias que mostram os requisitos funcionais em contexto.

CaU Essencias vs Concretos: Estilo essencial para manter as referências à interface com o utilizador de parte e focar na intenção do ator.

Situações de Modelação com CaU

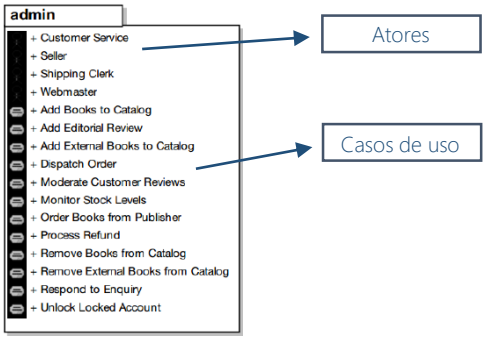
Ator

Primário

- Solicita o sistema para resolver problemas ou realizar objetivos.
- Iniciam os CaU.

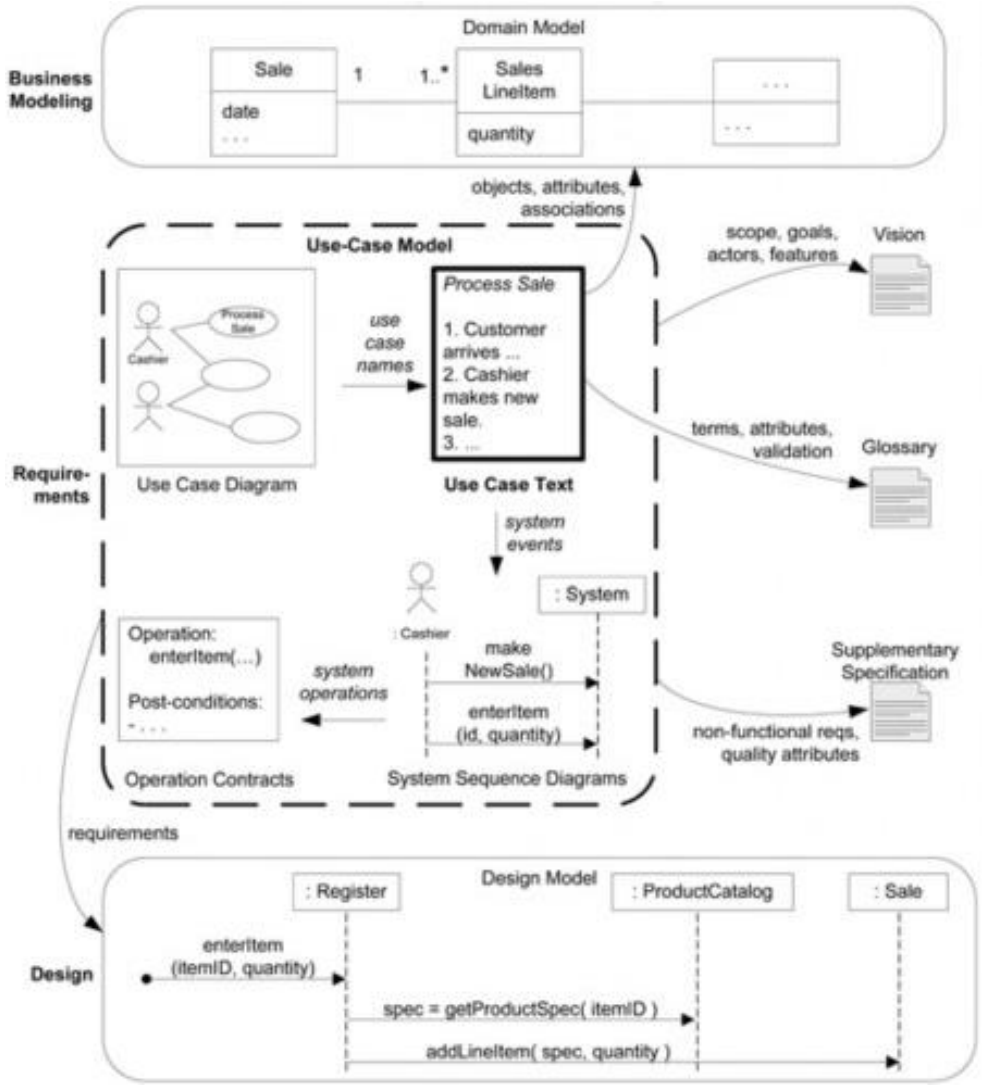
Secundário

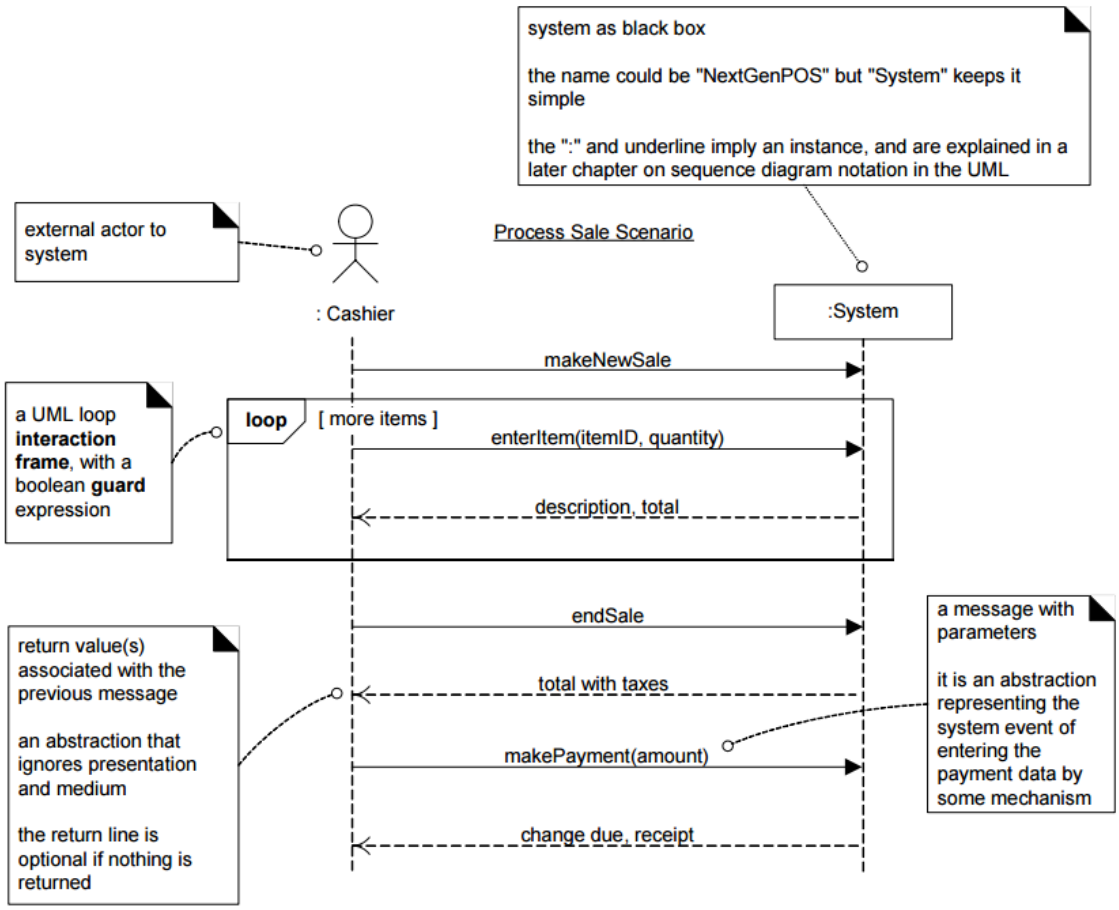
- Fornece serviços ou informação para algum cenário do CaU.
- Pode ser sistemas externos ou papéis de pessoas, que não são utilizadores.



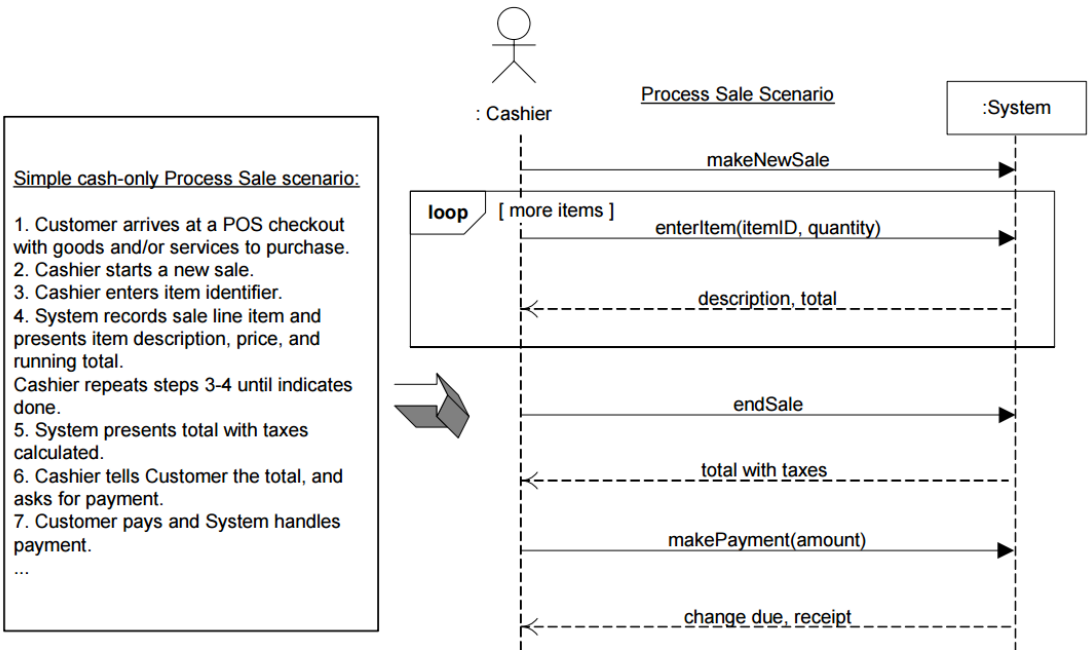
Nota: Os casos de utilização podem ser agrupados em pacotes.

7. Desenho do código e responsabilidades





Dos casos de utilização para o sistema



Para representar o actor primário (ou os actores) e o sistema usamos uma *lifeline*.
Os diagramas de sequência de sistema (DSS) **mostram operações de Sistema** para cada CaU. O sistema é modelado por uma classe que o representa globalmente. Opcionalmente, pode-se incluir o texto da descrição do CaU.

Uma **operação de sistema** (OpS) corresponde a um ponto de entrada no sistema, encapsula um conjunto de interações subjacentes entre objetos, necessárias para realizar essa operação.

A etapa seguinte na construção do **modelo dinâmico:** mostra, para cada OpS, a rede de objetos que vai ser ativada. Estes objetos são instâncias de classes que devemos identificar.
Intitulamos de atividade o que designamos por desenho, quando escolhemos as classes que participam na solução. Os diagramas de interação ajudam a distribuir responsabilidades, ou seja, a encontrar métodos.

Ponto essencial: distribuir responsabilidades pelos objetos, seguindo os princípios do desenho por objectos (*Object-Oriented Design*).

Responsabilidades de um objeto

Fazer
- Sobre o estado, como calcular alguma coisa, criar objetos.
- Iniciar uma ação noutros objetos.
- Coordenar/controlar as ações de outros objetos

Saber
- Conhecer o estado interno (escondido)
- Conhecer os objetos relacionados

UML para programar
Forward engineering: Do modelo para a implementação (código).
Reverse engineering: Da implementação (código) para o modelo.
Round-trip engineering: Transição transparente nos dois sentidos.

8. Modelação de Interações

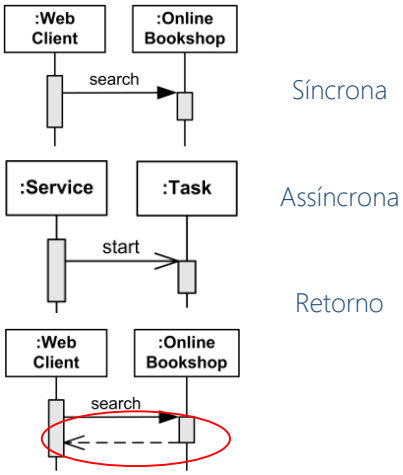
Modelação de comportamento (dinamismo) com diagramas de sequência

Os Diagramas de Interação são utilizados para visulaizar a interação via **mensagens entre objetos**. Estes diagramas dão origem a diagramas de sequência e de comunicação.

Existem 4 tipos de diagramas de interação disponíveis:

- Diagrama de sequência que tem o formato **alinhado**.
- Diagrama de comunicação que tem o formato de um **grafo**.
- Diagrama temporal (**timming**).
- Diagrama de visão geral de interação (**interaction overview**).

Semântica da invocação



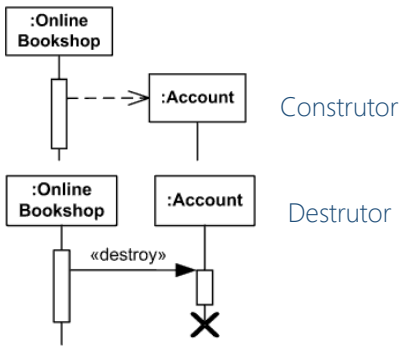
Síncrona

Assíncrona

Retorno

```
...
result = B.search();
...
```

Modelar a criação/destruição do participante

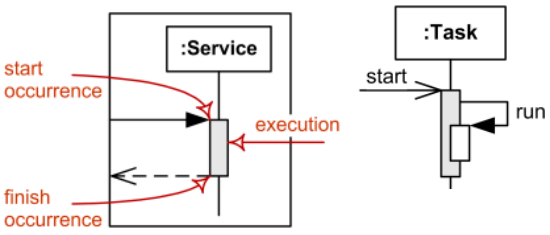


Construtor

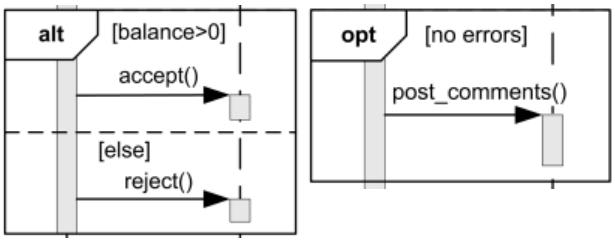
Destrutor

```
...
Account a = new Account()
...
...
a=null;  // in java
[a release] // objective C
free a; // C
```

Ativação



Fragmentos para mostrar alternativas

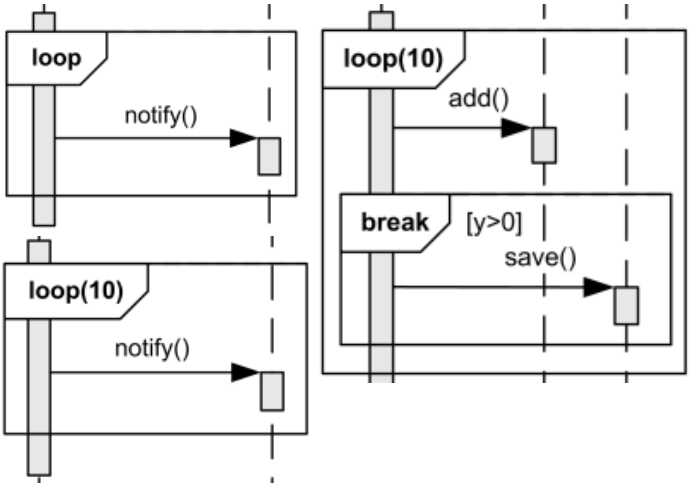



```
If (balance>0)
   otherObj.Accept()

Else
   otherObj.Reject()

If ( no errors )
   otherObj.Post_comments()
```

Fragmentos para mostrar loops




```
i=0;
While( i<10 )
{
 otherObj.Add()
 If ( y>0 ) break;
 i++
}
```

Diagramas de sequência vs Diagramas de Comunicação

	Benefícios	Limitações
Diagrama de Sequência	Mostra claramente a sequência/ordem temporal das mensagens e possibilita o alargamento da anotação.	Cresce para a direita à medida que se acrescebtam objetos.
Diagrama de Comunicação	Mais fácil de desenhar (os objetos podem ser adicionados em qualquer parte).	Menos expressivo, menos opções de notação e pouco suportado nas dferramentas UML.

Diagramas de sequência de sistema (DSS)

Para um cenário de um CaU mostram os eventos que os atores externos geram a sua ordem temporal, as necessidades de integração entre sistemas.

Num contexto de estilo para a contrução o sistema é tratado como uma caixa-fechada. Pois não mostra os componentes internos. As mensagens que chegam são as operações extermas ao sistema e são serviços solicitados/ invocados pelos atores.

9. Modelo de processo (Open) Unified Process

É necessário um processo de trabalho: **quem** faz o **quê**; **quando** é que alguma coisa deve ser construída; e **como** atingir certos objetivos.

Os requisitos novos/modificados passam por um processo de Engenharia de Software para obterem um sistema novo. O **processo** de engenharia de software é o conjunto das atividades e resultados associados que produzem uma peça de software.

Atividades fundamentais de um processo de Engenharia de Software:

A Especificação de requisito é caracterizada pela definição do problema/produto, pelo estado atual, pelo âmbito, pela visão da solução pretendida e pelo levantamento de requisitos/capacidades.

A **Especificação** do software (ou seja o desenho) é a definição do software a construir e as restrições de operação. Caracteriza-se pelos modelos de solução, apresenta uma estrutura de dados, apresenta também uma arquitetura/organização da solução e interfaces entre sub-sistemas/integrações com sistema externos.

O **Desenvolvimento** do software integra a implementação: escrever o código, desenvolver a base dados e procurar as oportunidades de reutilização.

A **Validação** e garantia de qualidade do software pretende garantir que o software satisfaça as expectativas do cliente. É necessário verificar a correção (realizar testes de módulo/unidade, testes de integração de módulos e testes do sistema num ambiente alvo) e validar a adequação (aceitação por parte do utilizador).

A **Instalação/operação** necessita de uma instalação de software e hardware, uma formação e uma entrada em produção.

A **Evolução/manutenção** do software inicializa uma adaptação e modificação do software para responder à alteração de requisitos do cliente e do ambiente. A **manutenção corretiva** corrige os erros no código, a **manutenção adaptativa** adapta o software às novas condições de operação e a **manutenção evolutiva** melhora/expande o produto.

Os modelos de Processo de Engenharia de Software são descrições abstratas de um processo de Engenharia de software segundo uma interpretação Indicam atividades e produtos a realizar e os papéis na equipa.

Waterfall

Aspetos chave como o resultado de cada fase, um conjunto de artefactos que deve ser aprovado. A fase seguinte começa quando a anterior termina. Em caso de erros, é necessário repetir passos anteriores.
Vantagens: abordagem simples, passos simples, o facto de ser disciplinado. O fim de cada etapa que progride linearmente.

Abordagem "ágil"

Objetivo: Desenvolver software rapidamente para responder às alterações rápidas dos requisitos.

Para atingir a agilidade é necessário: práticas que equilibrem a disciplina e o feedback e aplicar princípios de desenho que favoreça a construção de software flexível e evolutivo.

Fazes do projeto

Inception: Saber o **que** contruir
Apresenta uma iteração curta, por norma. É necessário produzir um documento de visão, um caso de negócio e desenvolver e focar-se apenas no que realmente interessa. Desenvolver os requisitos do projeto a alto nível: iniciar os casos de uso (e, é opcional, os modelos de domínio).

Elaboration: Saber **como** contruir
Apresenta várias ou longas iterações. Equilibra, faz referência às principais questões técnicas e riscos de negócios. Fase de testes. Produz e valida a arquitetura executável, é importante definir pontos-chave e implementar as interfaces cuidadosamente.

Construction: **Contrução** do produto
É necessário definir uma estratégia e o código da implementação do produto. Ao realizar vários testes, o objetivo é verificar a evolução da arquitetura.

Transition: A transição, a fase de **estabilizar** e **implementar** o produto
Constante atualização das versões do projeto

10. Máquinas de estado

Notação básica dos Diagramas de Estados

Estados → Caixas : condição em que se encontra o objeto.
Transições → Setas : evolução de um estado para o outro.
Triggers → Etiquetas : acontecimentos relevantes que causam transições de estado
Condições de acesso.
Marcador início/fim.

Quando usar?
Objetos com comportamento dependente do estado: controlador de um dispositivo físico, protocolos de comunicação, coordenadores do fluxo da interface gráfica.
No Modelo de Domínio: caracterizar os estados de uma classe complexa.

11. Aplicação da UML ao longo do processo

Estrutura e comportamento de sistemas
- Análise, desenho e implementação de sistemas baseados em software.
- Elementos do modelo representam entidades do mundo do software.
- Especialmente adequada para o desenvolvimento orientado por objetos.

Processos organizacionais novos ou já existentes
- Especificar ou documentar o processo de negócio
- Não implica ou assume uma implementação em software

Conceitos
Anotações são comentários que pode ser usado para anotar qualquer elemento
Pacotes são mecanismos para dividir um modelo em partes, serve como mecanismo genérico para fazer agrupamentos.
Estereótipos são especializações de uma semântica de um elemento de modelação.
Valores etiquetados (tagged values) estendem elementos do modelo com uma linguagem "computável"
Restrições permitem adicionar regras ao modelo ou condicionar a sua interpretação. A condição ou restrição relacionada com um ou mais elementos.

12. Vistas de Arquitetura

A arquitetura do sistema aborda diferentes perpetivas de análise

Assuntos da arquitetura do sistema

Organização estrutural do sistema em grandes blocos: componentes desenvolvidos e/ou integrados. Uma arquitetura é definida para satisfazer os requisitos não funcionais e as restrições de operação que são determinantes. Compromissos e decisões.

Arquitetura lógica: organização geral dos blocos de software (**packages**), independente da tecnologia de implementação.

Arquitetura de instalação: visão dos equipamentos e configuração concreta de produção (conectividade, distribuição).

Arquitetura de componentes: peças construídas com uma tecnologia concreta. É normal dividir sistemas complexos em subsistemas mais geríveis. O componente é uma peça substituível, reusável de um sistema maior, cujos detalhes de implementação são abstraídos. A funcionalidade de um componente é descrita por um conjunto de interfaces fornecidos. Para além de implementar, o componente pode requerer funcionalidades de outros.

Diagramade instalação da UML

Nós (*node*): Um equipamento de hardware
Ambiente de execução: Um ambiente externo à solução que proporciona o contexto necessário à sua execução.
Artefactos (*artifact):* Ficheiros concretos que são executados ou utilizados pela solução.

Arquitecturas por camadas

Divisão modular da solução de software em camadas/níveis. As camadas são sobrepostas, cada camada tem uma especialização, não se pode saltar camadas.