Diagramas de Use Case

Sumário

- Definição de requisitos.
- Diagramas de *Use Case* I — conceitos base
- Diagramas de *Use Case* II — conceitos avançados
- Resumo
- Exercícios

Definição de Requisitos

Definição dos requisitos do sistema, duas abordagens possíveis:

- Visão estrutural — interna
- Visão orientada aos *use case* — externa

Visão Estrutural (OO)

- Definir classes;
- Definir métodos das classes;
- Definir interface com o utilizador (comportamento do sistema face ao utilizador).

Problemas: O que interessa ao utilizador é o comportamento do sistema, no entanto a interface com o utilizador só é definida no final do processo.

- Perigo de o sistema não fornecer toda a funcionalidade pretendida;
- Perigo de o sistema fornecer funcionalidade não pretendida (= desperdício de trabalho).

Visão orientada aos use case

- Identificar *Actores* — quem vai interagir com o sistema?
- Identificar *Use Case* — o que se pretende do sistema?
- Identificar classes de suporte à realização dos use case.

Vantagens:

- Não há trabalho desnecessário;
- S.I. suporta as tarefas do cliente.

Use Case

- Uma unidade coerente de funcionalidade — um serviço
- define um comportamento do sistema sem revelar a estrutura interna — apenas mostra a comunicação entre sistema e actores
- o conjunto de todos os *use case* define a funcionalidade do sistema
- deve incluir o comportamento normal, bem como variações (erros, etc.)
  - vamos definir o comportamento com texto *estruturado*;
  - vamos também definir as pré-condições e pós-condições de cada *use case* (cf. *design by contract*).

Design by contract

- *Design by contract* (DBC) baseia-se na noção de um contrato entre um cliente e um fornecedor para a realização de um serviço.
- O conceito central do DBC é a asserção (uma asserção é uma expressão booleana que nunca deverá ser falsa).
- Tipicamente as asserções são automaticamente testadas durante a fase de *debug*.
- O DBC identifica três tipos de asserções:
  - pré-condições — condições que se devem verificar para a invocação de um dado *serviço* ser válida;
  - pós-condições — condições que se devem verificar após a execução de um *serviço*;
  - invariantes — asserções que se devem verificar durante o tempo de vida da *entidade* a que se aplicam.
- A partir da versão 1.4 o java passou a ter *asserts* que podem ser utilizados para definir pré- e pós condições — no entanto não suporta invariantes).

O *use case* para fazer um telefonema:

```
Use case: Fazer Telefonema
Pré-condição:
  Telefone ligado e em descanço
Comportamento Normal:
  1. Utilizador marca numero e pressiona OK
  3. Telefone transmite sinal de chamada
  4. Utilizador aguarda
  5. Telefone estabelece ligação
  6. Utilizador fala
  7. Utilizador pressiona tecla C
  8. Telefone desliga chamada
Comportamento alternativo:
  3. Telefone transmite sinal de ocupado
  4. Utilizador presssiona C
  5. Telefone cancela chamada
Comportamento alternativo:
  3. Telefone cancela chamada
Pós-condição:
  Telefone ligado e em descanço
```

Identificação de Use Cases

- Podemos identificar os *Use Case* do sistema a partir da identificação de cenários de utilização.
- Um cenário descreve um contexto concreto de interacção entre o utilizador e o sistema. Por Exemplo:

  > Durante o semestre o Prof. Faísca foi enviando os sumários com breves resumos da matéria leccionada, via e-mail, para o sistema Fly2. Após o fim das aulas, o Prof. Faísca utilizou a interface web do sistema para actualizar cada um dos sumários com descrições mais completas das matérias leccionadas. Finda essa actualização imprimiu os sumários e enviou-os à Secretaria.

- A partir dos cenários podemos identificar os *Use Cases* (serviços) necessários à correcta disponibilização da funcionalidade requerida pelo mesmo.

No caso anterior podemos identificar os seguintes *Use Cases*:

1. enviar sumários via e-mail
2. actualizar sumários via web
3. imprimir sumários (via web?/via e-mail?)
4. enviar sumários à secretaria — deverá este use case ser considerado?

   No cenários descrito o envio é feito em papel. Não se trata, portanto, de um serviço fornecido pelo sistema. No entanto, podemos discutir a possibilidade de o envio passar a ser feito electrónicamente — estariamos a alterar o modo de trabalho inicialmente previsto/actual!

   > Durante o semestre o Prof. Faísca (1.) **foi enviando os sumários** com breves resumos da matéria leccionada, **via e-mail**, para o sistema Fly2. Após o fim das aulas, o Prof. Faísca (2.) **utilizou a interface web** do sistema **para actualizar** cada um dos **sumários** com descrições mais completas das matérias leccionadas. Finda essa actualização (3.) **imprimiu os sumários** e (4.) enviou-os à Secretaria.

Diagramas de Use Case I — conceitos base

- Modelam o contexto geral do **sistema**. Quais os **actores** que com ele se relacionam e que **use case** deve suportar.
- A **concepção** do sistema é guiada pelo modelo de *use cases*:
  - Utilizam-se *use cases* para capturar os requisitos funcionais do sistema de uma forma sistemática;
  - O modelo de *use cases* captura toda a funcionalidade requerida pelos utilizadores;
- A **implementação** do sistema é guiada pelo model de *use cases*:
  - cada *use case* é implementado sucessivamente:
  - quando todos os *use cases* estiverem implementados obtém-se o sistema final;
  - fica facilitada a manutenção do sistema sempre que os requisitos sejam alterados;
- O modelo de *use cases* é utilizado para o planeamento de **testes**:
  - Após a definição do modelo de *use cases*: planear *black-box testing*.
  - Após a implementação dos *use cases*: planear *white-box testing*.

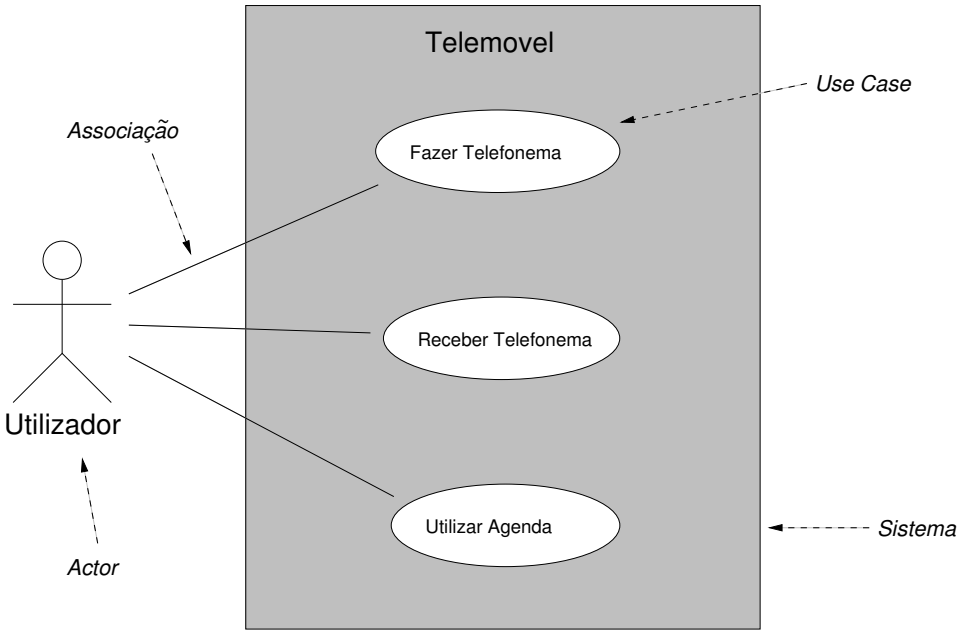
Black-box testing

- Utilizado para verificar se o sistema implementa toda a funcionalidade pretendida.
- Permite detectar erros de "omissão" (funcionalidade não implementada).

White-box testing

- Utilizado para verificar se o sistema implementa a funcionalidade de forma correcta.
- Permite detectar erros na implementação da funcionalidade pretendida.

Exemplo de diagrama de Use Case



Sistema

- define as fronteiras do sistema

Use Case (novamente)

- Uma unidade coerente de funcionalidade — um serviço
- define um comportamento do sistema sem revelar a estrutura interna — apenas mostra a comunicação entre sistema e actores
- o conjunto de todos os *use case* define a funcionalidade do sistema
- deve incluir o comportamento normal, bem como variações (erros, etc.)
  - vamos definir o comportamento com texto *estruturado*;
  - vamos também definir as pré-condições e pós-condições de cada *use case* (cf. *design by contract*).

Actor

- uma abstracção para uma entidade fora do sistema
- um actor modela um propósito (*alguém* que tem um interesse específico no sistema) — pode não mapear 1 para 1 com entidades no mundo real
- um actor não é necessariamente um humano — pode ser um computador, outro sistema, etc.
- cada actor define um conjunto de papeis que utilizadores do sistema podem assumir
- o conjunto de todos os actores definem todas as formas de interacção com o sistema

Associação

- representa comunicação entre o actor e o sistema — através de *use cases*

Novamente a Gestão de Sumários

Sistema de gestão de sumários e presenças.

Etapas a cumprir (com o auxílio de cenários de utilização do sistema):

1. Identificar actores
2. Identificar *use cases*
3. Identificar associações

Identificar actores

- Quem vai utilizar o sistema?
- Neste caso: Docente, Secretaria e Aluno

Identificar use cases

- Objectivos dos utilizadores/actores?
- Resposta a estimulos externos.

Identificar associações

- Que actores utilizam que *use cases*?
- Nem sempre é imediatamente evidente se a comunicação entre o sistema em análise e sistemas externos deve ser representada, quatro abordagens podem ser identificadas:
  - mostrar todas as associações;
  - mostrar apenas as associações relativas a interacção iniciada por sistemas externos;
  - ➪ mostrar apenas as associações relativas a interacções em que é o sistema externo o interessado no *use case*;
  - não mostrar associações com sistemas externos.

Todas as associações

- Todos os sistemas externos que interagem com o sistema em análise são apresentadas como actores e todas as interacções são representadas nos diagramas.
- Demasiado abrangente, em muitos casos existem interacções com outros sistemas apenas por razões de implementação e não por se tratarem de requisitos do sistema.

Apenas as associações relativas a interacção iniciada por sistemas externos

- São representados como actores os sistemas externos que iniciem diálogo com o sistema em análise.

- Mesmo assim pode ser uma solução demasiado abragente.

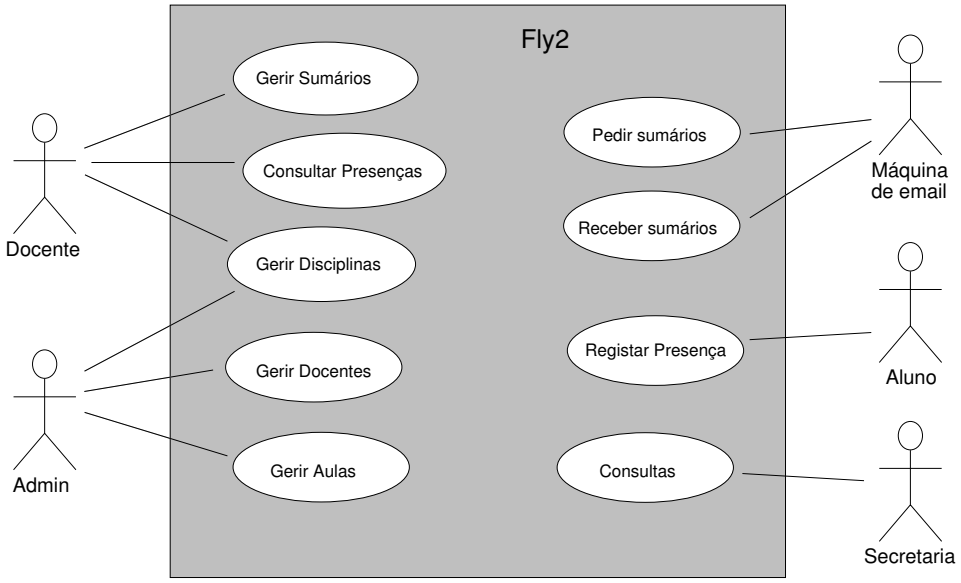
Apenas as associações em que é o sistema externo o interessado

- São apresentados como actores os sistemas externos que necessitam de funcionalidade fornecida pelo sistema em análise (por vezes esses sistemas podem actuar como canais de comunicação).
- Usalmente esta é uma solução equilibrada.

Não mostrar associações com sistemas externos

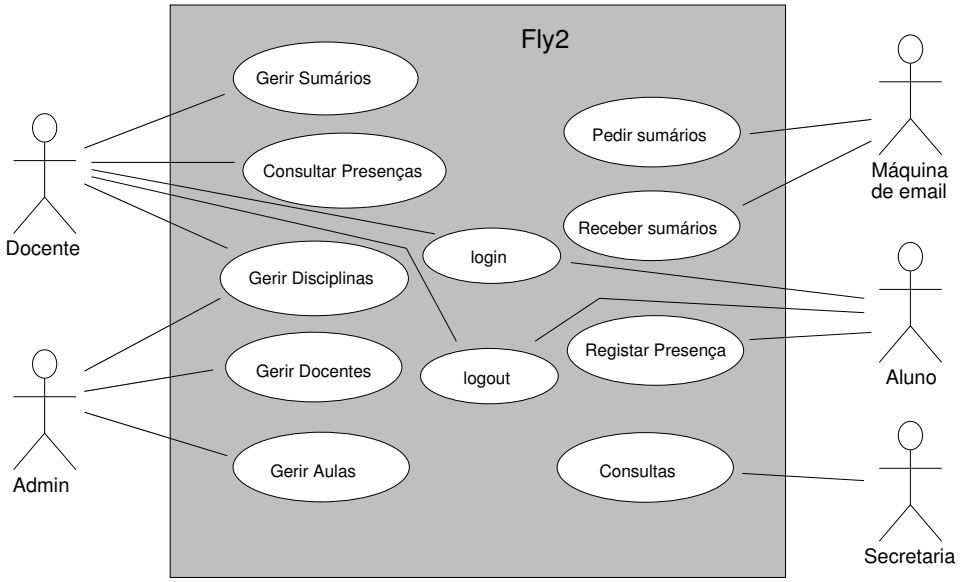
- Apenas os utilizadores são actores, neste caso quando existem sistemas externos apresentam-se os seus actores em diálogo directo com o sistema a ser modelado.
- De uma outra forma esta solução também é demasiado abrangente e pode levar a confusão sobre quem está realmente a utilizar o sistema.

Visão geral – versão 1



- falta mecanismo de autenticação

Visão geral – versão 2

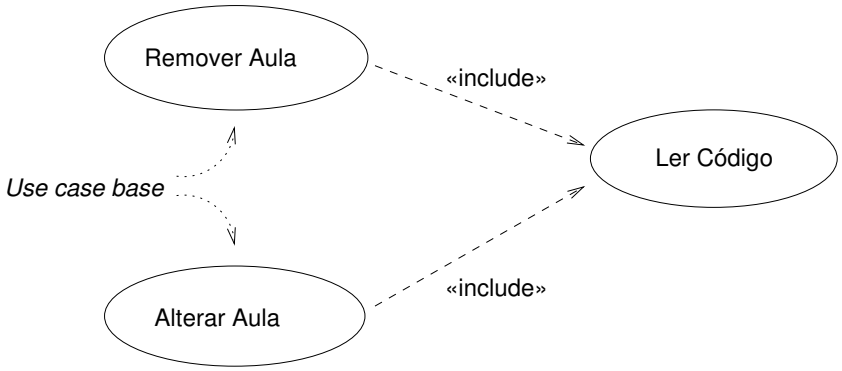


- são adicionadas pré-condições nos *use case* Gerir Sumários, Gerir Presenças, Gerir Disciplinas e Registar Presença a exigir que tenha sido feito login.

Diagramas de Use Case II — conceitos avançados

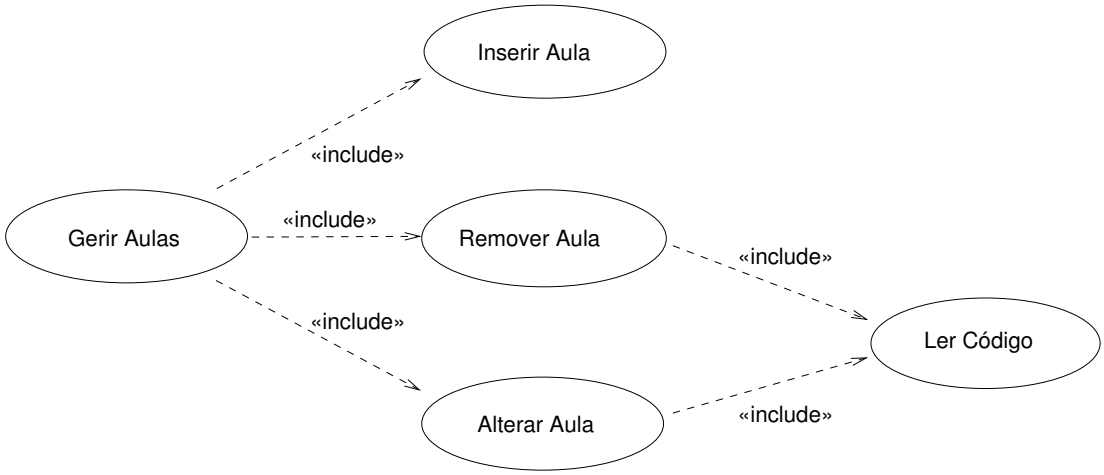
<<include>>

- Um estereótipo de dependência.
- Utilizado para indicar a reutilização de comportamento.



- Actores utilizam o use *case base*
- Quando o use *case base* é executado, também o *use case* incluido o é

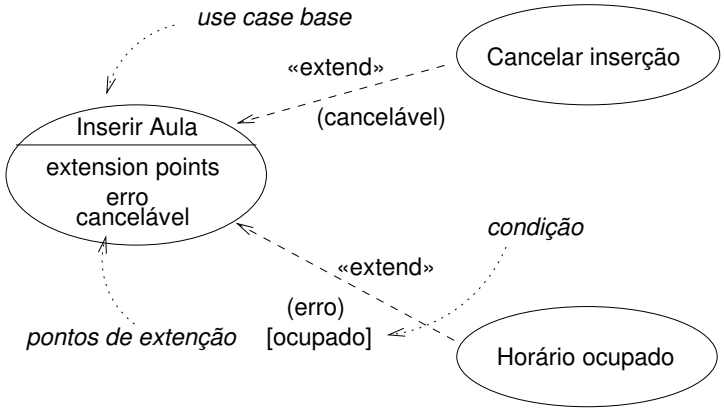
- Também pode ser utilizado para estruturar *use cases*:



- Não exagerar!
- Em alternativa, utilizar sub-diagramas.

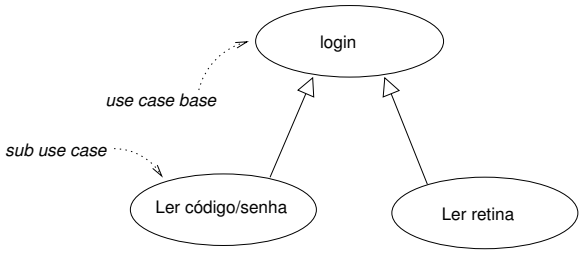
<<extends>>

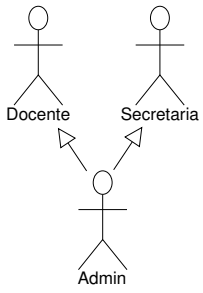
- Outro estereótipo de dependência.
- Permite adicionar comportamento a um *use case* base.



- Estratégia: escrever caso base; identificar variações; utilizar extensões para elas.
- Caso base deve ser um *use case bem formado* sem as extensões!
- Extensão pode não ser um *use case* bem formado por si só.

Generalização/Especialização





- Sub-elementos são casos particulares de super-elementos.
- Um sub-elemento pode ser utilizado onde quer que o super-elemento possa.
- Útil para *user profiling* (definição de níveis de acesso).
- Nos exemplos apresentados:
  - Existem duas formas de fazer login.
  - O actor Admin pode realizar todos os *use cases* de Docente e Secretaria.

Resumo

- Os diagramas de *Use Case* permitem definir os requisitos funcionais de um sistema:
  - que serviços deve fornecer;
  - a quem os deve fornecer.
- Notação diagramática facilita o diálogo (com os clientes e dentro da quipa de desenvolvimento).
- Utilizando diagramas de use case, clientes e equipa de desenvolvimento podem chegar a um acordo sobre qual o sistema a desenvolver.
- O processo de testes do sistema fica facilitado.
- A resolução de alterações nos requisitos funcionais fica facilitada.

No entanto:

- Os diagramas de use case não suportam a captura de requisitos não funcionais.

Quando utilizar diagramas de Use Case?

- Sempre que se estiverem a analisar requisitos!