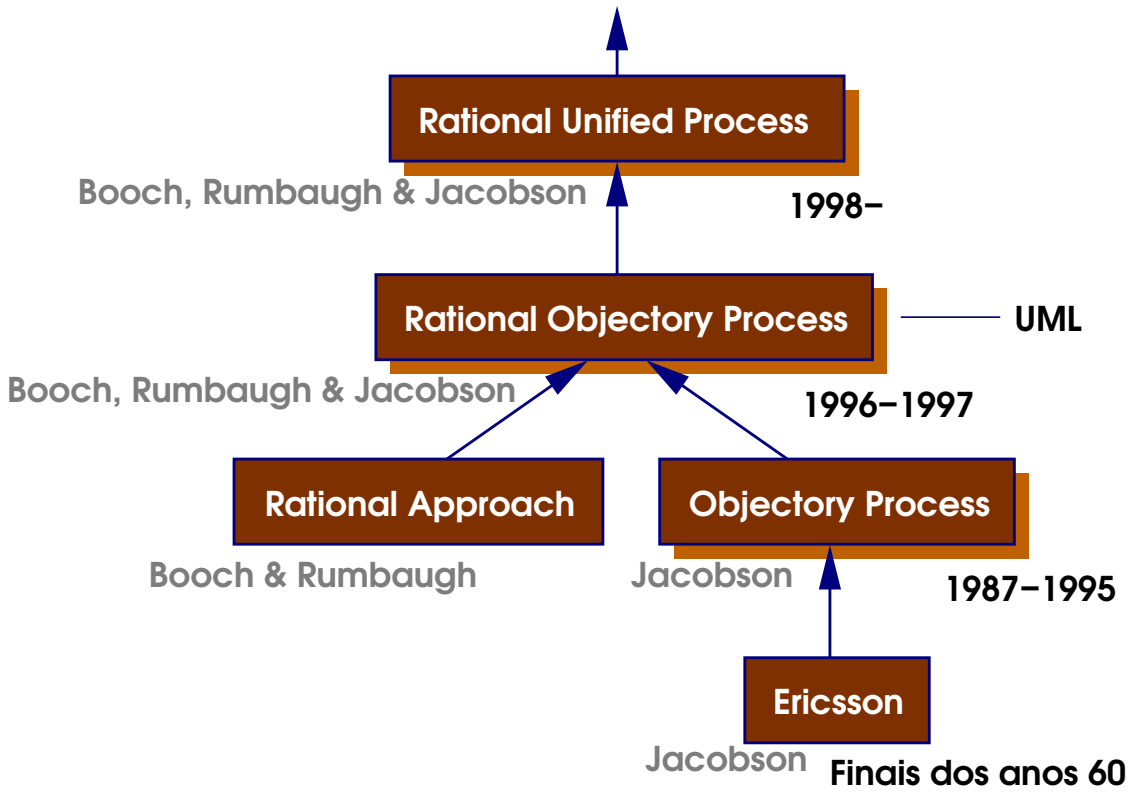
Unified Software Development Process

Sumário

- Breve história do Unified Process
- O Unified Process
- O ciclo de vida do Unified Process
- O RUP (Rational Unified Process)

Breve História do Unified Process



O Unified Process

O Unified Process é:

- Uma *framework* para o processo de desenvolvimento, genérica e adaptável.

O Unified Process fomenta:

- O desenvolvimento baseado em componentes.

O Unified Process utiliza:

- o UML como ferramenta de modelação durante todas as fases do processo de desenvolvimento.

Três ideias chave

- desenvolvimento guiado por *Use Cases* (Casos de Uso)
- desenvolvimento centrado na arquitectura;
- desenvolvimento iterativo e incremental.

Desenvolvimento guiado por Use Cases

- Um *use case* representa uma interacção entre o sistema e um humano ou outro sistema;
- O modelo de *use cases* é utilizado para:
  - guiar a concepção do sistema (captura de requisitos funcionais);
  - guiar a implementação do sistema (implementação de um sistema que satisfaça os requisitos);
  - guiar o processo de testes (testar que os requisitos são satisfeitos).

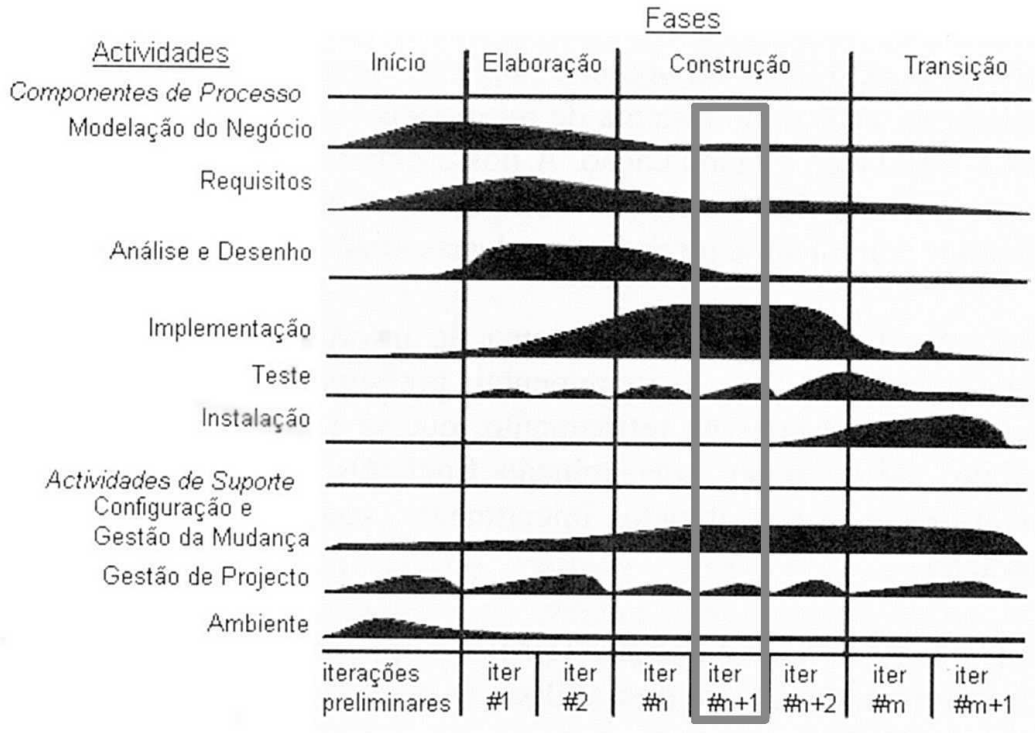
Desenvolvimento centrado na arquitectura

- O modelo de use cases descreve a função do sistema, o modelo da arquitectura descreve a forma;
- O modelo arquitectural permite tomar decisões sobre:
  - O estilo de arquitectura a adoptar;
  - Quais os componentes do sistema e quais as suas interfaces;
  - A composição de elementos estruturais e comportamentais.

Desenvolvimento iterativo e incremental

- Permite dividir o desenvolvimento em "pedaços geríveis";
- Em cada iteração:
  - identificar e especificar *use cases* relevantes;
  - criar uma arquitectura que os suporte;
  - implementar a arquitectura utilizando componentes;
  - verificar que os *use cases* são satisfeitos.
- Selecção de uma iteração:
  - grupo de *use cases* que extendam a funcionalidade;
  - aspectos de maior risco.

O ciclo de vida do Unified Process



- O Unified Process divide o desenvolvimento de software em 4 fases:
  - Início (*Inception*);
  - Elaboração (*Elaboration*);
  - Construção (*Construction*);
  - Transição (*Transition*).
- Cada fase é constituida por um conjunto de iterações.
- Em cada iteração são realizadas um conjunto de actividades:
  - Análise de requisitos;
  - Análise e desenho;
  - Implementação;
  - Testes;
  - Instalação.
- Em função da fase em que se está a realizar a iteração, algumas actividades têm mais peso que outras.
- O objectivo é que, em cada iteração, seja produzida uma versão do produto final.

Fases do Unified Process

Início

- Identificar problema
- Decidir âmbito e natureza do projecto
- Fazer estudo de viabilidade

Resultado da fase: decisão de avançar com o projecto.

Elaboração (Análise/Concepção Lógica)

- O que vamos construir (quais os requisitos?)
- Como vamos fazê-lo? (qual a arquitectura?)
- Que tecnologias vamos utilizar?

Resultado da fase: uma arquitectura geral (conceptual) do sistema.

Construção (Concepção Física/Implementação)

- Processo iterativo e incremental
- Em cada iteração: análise/especificação/codificação/teste/integração de parte do SI

Resultado da fase: um sistema de informação!

Transição

- Acertos finais na instalação do SI
- Optimização, formação.

Resultado da fase: um SI instalado e 100% funcional.

O RUP (Rational Unified Process)

- O RUP é uma *concretização* do Unified Process.
- O RUP fornece:
  - ferramentas de gestão do processo de desenvolvimento segundo a *framework* definida pelo Unified Process (ver página 59);
  - ferramentas para a modelação e desenvolvimento baseadas no UML;
  - uma base de conhecimento (*knowledge base*).
- Na verdade as coisas passaram-se um pouco ao contrário...

Apresentação do UML

Sumário

- UML
- Breve história do UML
- Diagramas do UML
- Meta modelo do UML
- Mecanismos de extensão
- Decorações

UML — Unified Modelling Language

UML: *Unified Modelling Language* (Booch, Jacobson & Rumbaugh)

- UML foi pensado para o desenvolvimento de sistemas orientados aos objectos
  → permite explorar o paradigma OO
- UML possibilita o trabalho a diferentes níveis de abstracção
  → facilita a comunicação
- UML não é uma linguagem, mas um conjunto de linguagens
  → inclui modelos para as diferentes fases do desenvolvimento.
- UML não é um processo de desenvolvimento de software, pode ser usado com diferentes processos.

Breve história do UML

- anos 60
  - Simula 67 é a primeira linguagem orientada aos objectos;
- anos 70
  - Aparece o Smalltalk;
- anos 80
  - as linguagens orientadas aos objectos (OO) tornam-se *utilizáveis*
    Smalltalk estabiliza; surgem o Objective C, C++, Eiffel, CLOS, etc.
  - finais dos anos 80 — surgem as primeiras metodologias de modelação OO
- anos 90
  - existem dezenas de metodologias de modelação OO
    Shlaer/Mellor, Coad/Yourdon, Booch, OMT (Rumbaugh), OOSE (Jacobson), etc. etc.
  - meados dos anos 90 — começam as tentativas de unificação dos métodos
  - 1994 — Rumbaugh junta-se a Booch na Rational Software Corporation

  - 1995 — Booch e Rumbaugh apresentam a versão 0.8 do *Unified Method* (viria depois a mudar de nome para *Unified Modelling Language*); Jacobson junta-se a Booch e Rumbaugh
  - 1996 — o OMG (*Object Management Group*) pede propostas para um standard de modelação OO
  - Setembro, 1997 — a Rational, em conjunto com outras empresas (HP, IBM, Oracle, SAP, Unisys, ...), submete o UML 1.0 ao OMG como proposta de standard (existiram outras propostas)
  - Novembro, 1997 — o UML é aprovado como standard OO pelo OMG; o OMG assume a responsabilidade do desenvolvimento do UML
- 00(?!)
  - 2004 — está a ser finalizado o UML 2.0

`www.uml.org`

Diagramas do UML

O UML define os seguintes oito diagramas base:

- ❏ Diagramas de Use Case
- ❏ Diagramas de Classe
- Diagramas comportamentais
  - Diagramas de Interacção
    - ❏ Diagramas de Sequência
    - ❏ Diagramas de Colaboração
  - ❏ Diagramas *statechart* (de estados)
  - ❏ Diagramas de Actividade
- Diagramas de implementação
  - ❏ Diagramas de Componentes
  - ❏ Diagramas de Instalação (*Deployment*)

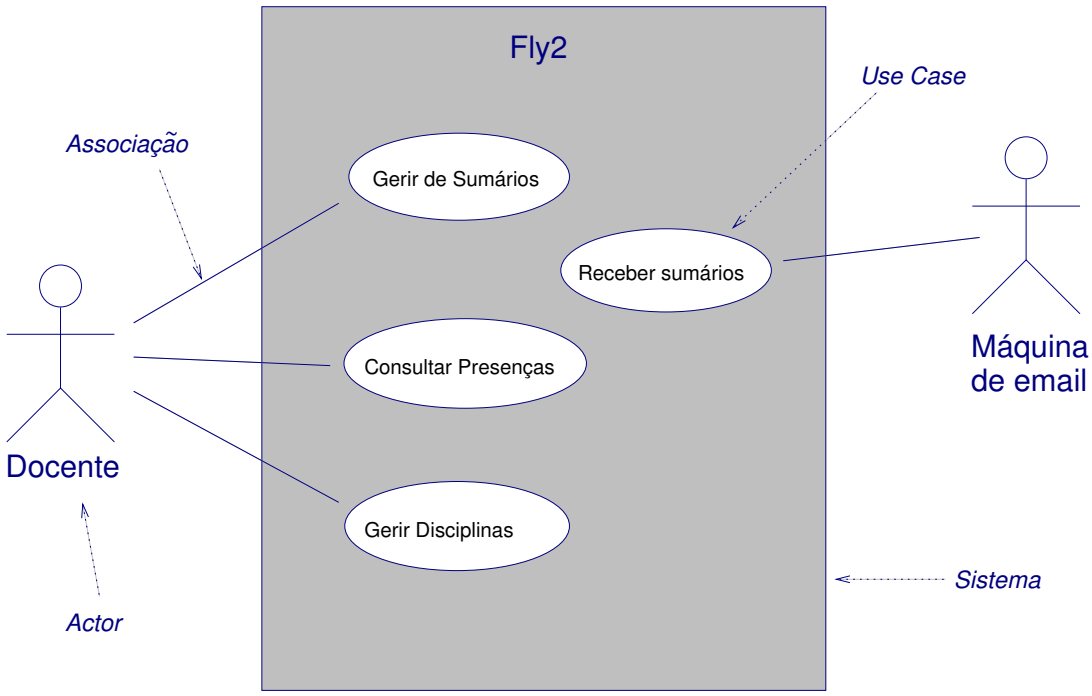
❏ Diagramas UML

- A escolha dos modelos a utilizar tem uma grande impacto na forma como um dado problema é abordado e, consequentemente, na solução que se irá atingir.
- A *Abstracção* (prestar atenção aos detalhes importantes e ignorar ou irrelevantes) é um factor fundamental.
- Assim:
  - Todo o sistema complexo deve ser abordado através de um pequeno conjunto de vistas/modelos tão independentes quanto possível;
  - Cada modelo pode ser expresso a diferentes níveis de detalhe;
  - Os melhores modelos são aqueles que têm relação directa com a realidade.
- Os oito tipos de diagramas identificados anteriormente procuram cobrir todas as necessidades de modelação que ocorram durante o desenvolvimento de software.

Um exemplo

Desenvolver uma aplicação para gestão de sumários e presenças nas aulas para universidades.

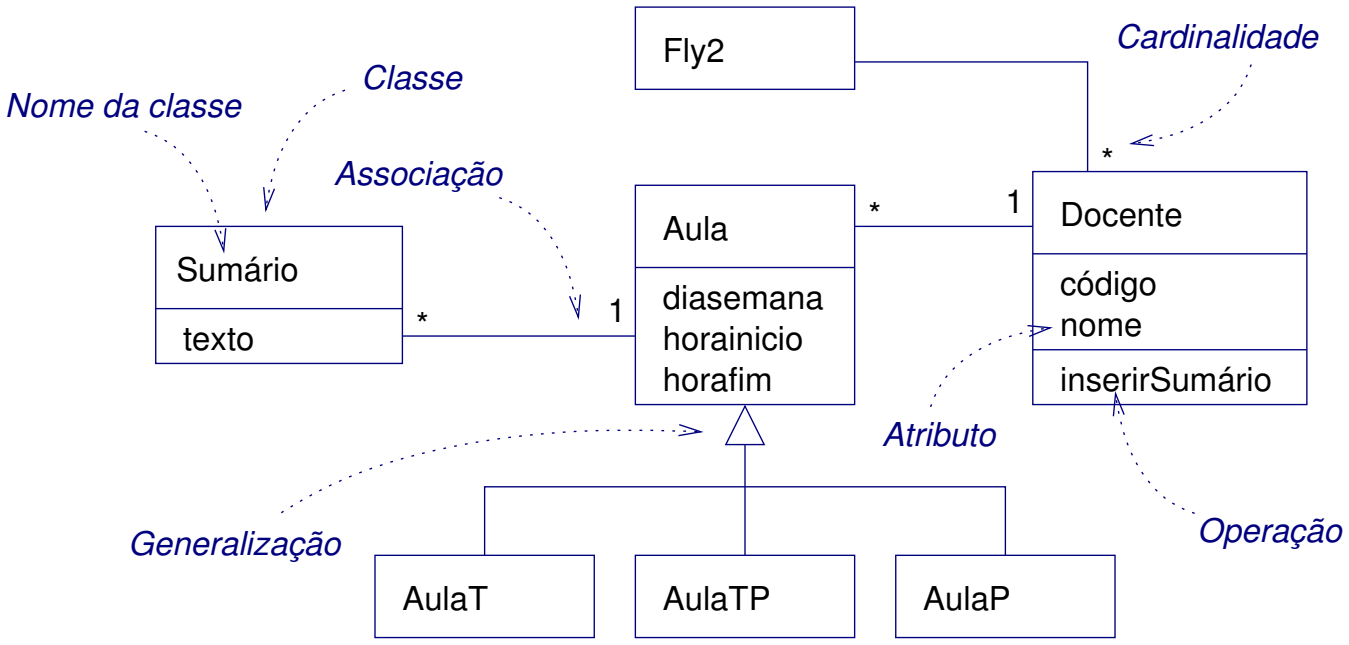
Diagramas de Use Case



- Identificam os utilizadores do sistema / capturam os requisitos funcionais.

Ver página 59

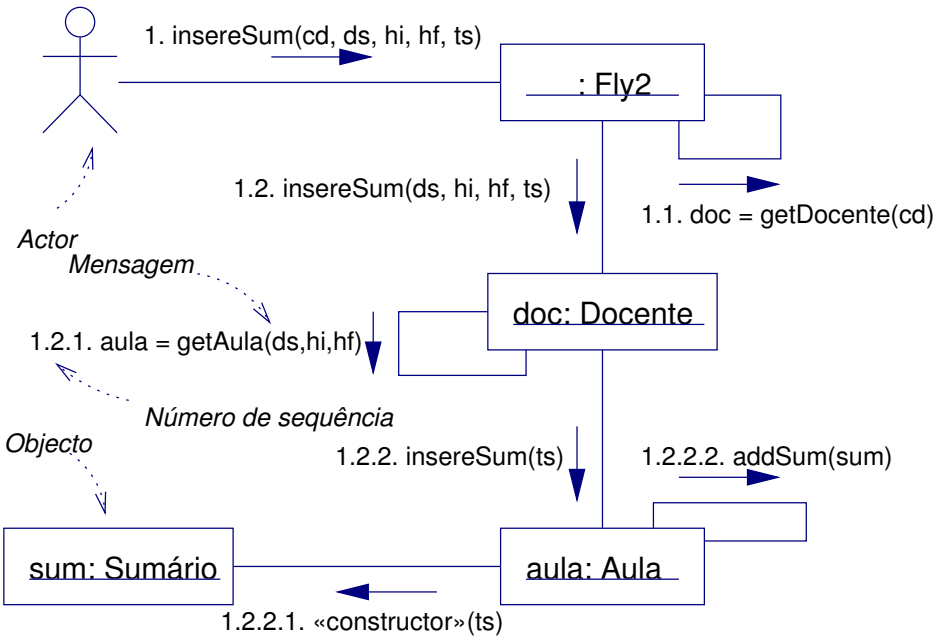
Diagramas de Classe



- Modelam a arquitectura(?) do sistema.

Ver página 59

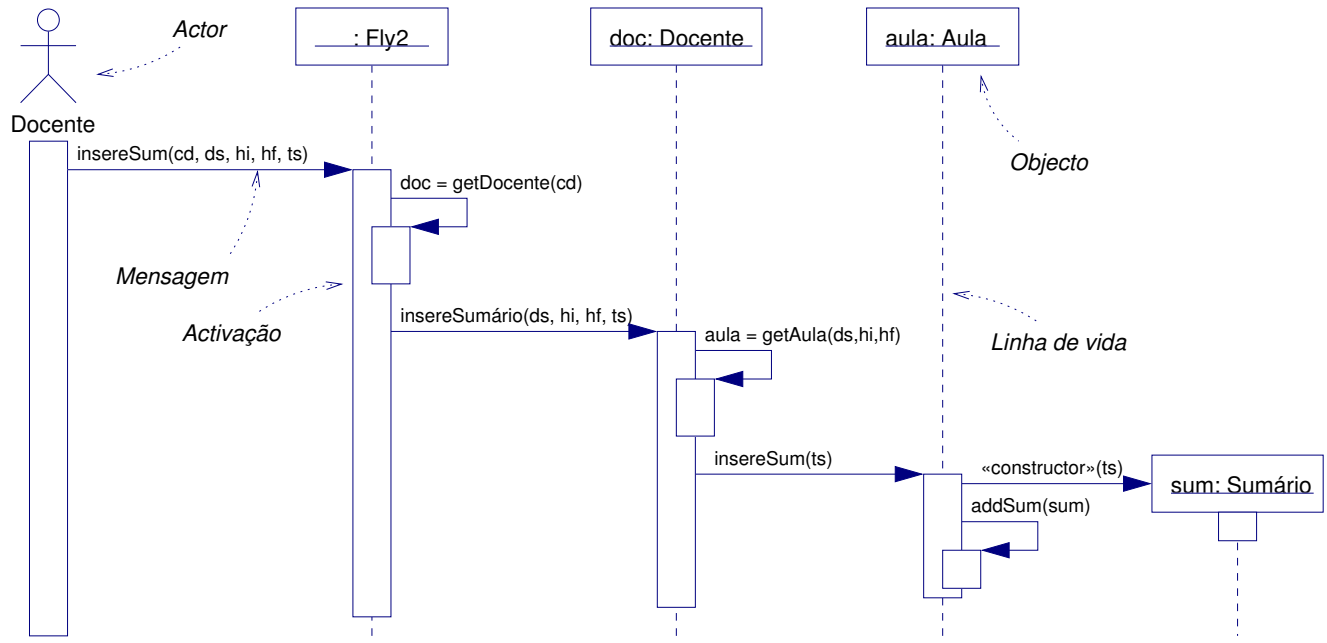
Diagramas de Colaboração



- Modelam o comportamento do sistema (ênfase no aspecto estrutural).

Ver página 128

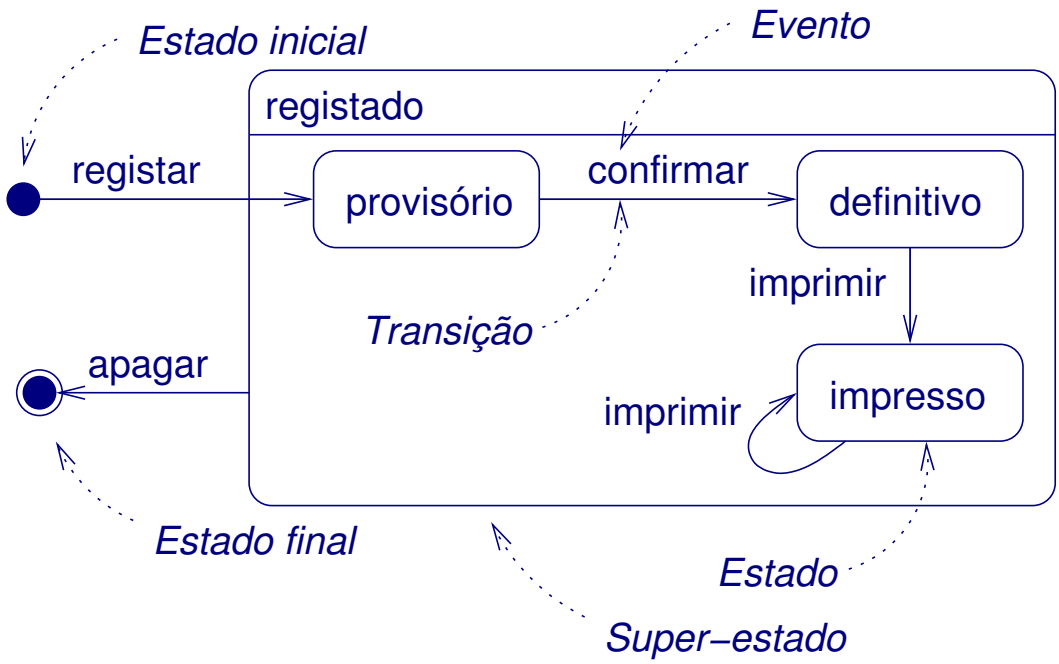
Diagramas de Sequência



- Modelam o comportamento do sistema (ênfase no aspecto temporal).

Ver página 71

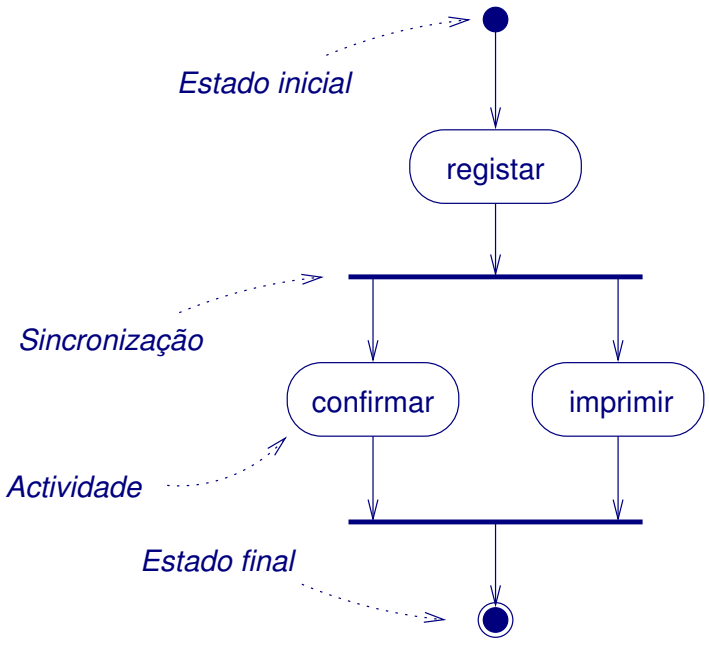
Diagramas de Statechart



- Modelam ciclo de vida de objectos no sistema.

Ver página 59

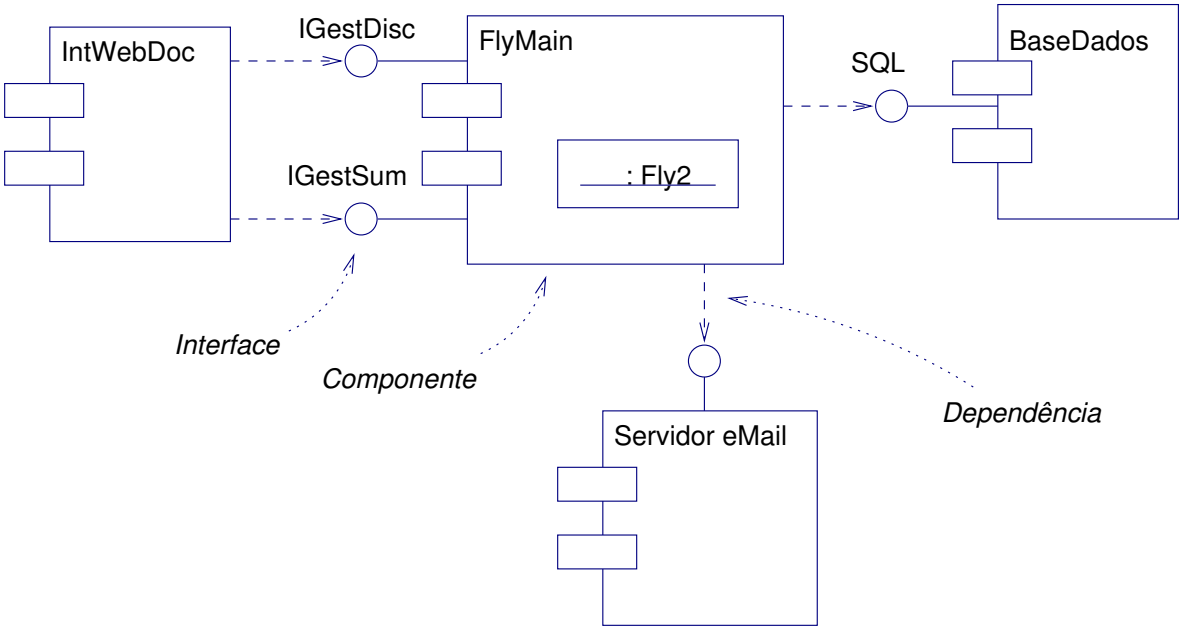
Diagramas de Actividade



- Modelam comportamento (ênfase actividades realizadas).

Ver página 74

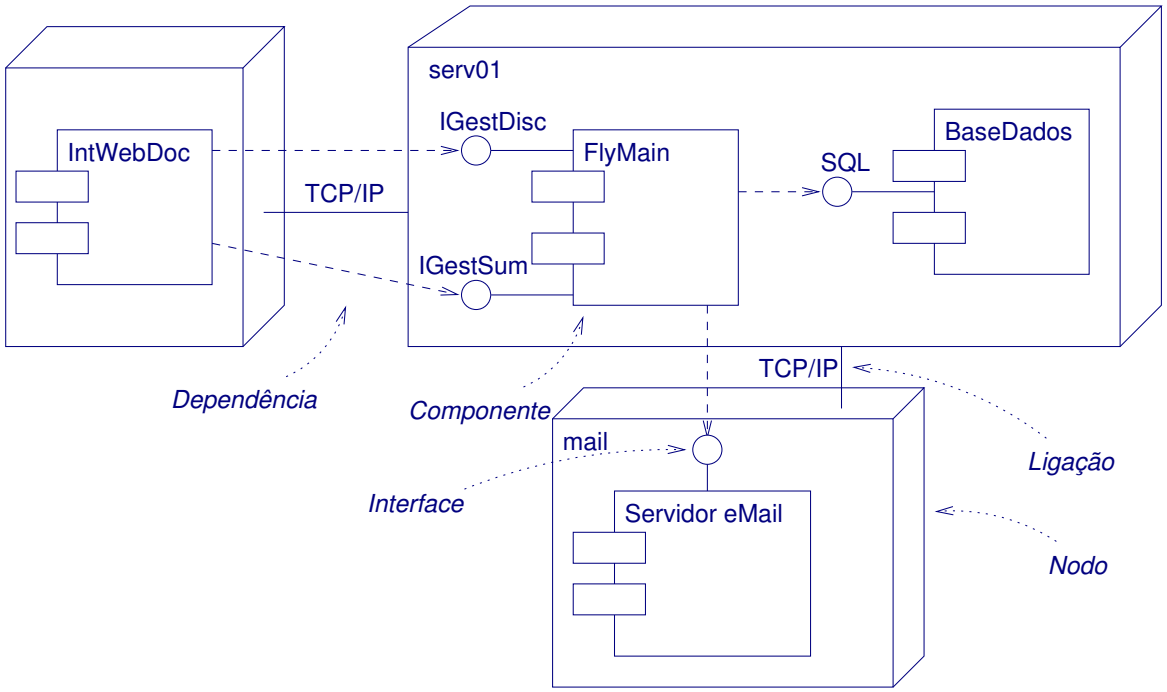
Diagramas de Componentes



- Definem como o sistema é construido a partir de componentes.

Ver página 59

Diagramas de Instalação



- Definem como o sistema deverá ser instalado.

Ver página 70

Meta-modelo do UML

Se o UML permite escrever modelos, poderemos modelar o UML?

- A definição do UML engloba quatro níveis de abstracção:
  - meta-meta modelo

    Define o elemento mais básico em que o UML se baseia: o conceito de *Coisa*.
  - meta modelo do UML

    Define o tipo de *Coisas* que podem ser utilizadas num modelo UML (por exemplo, o conceito de Classe é uma instância de *Coisa*).
  - modelos UML

    Os modelos UML que podemos escrever (por exemplo, a classe Sumário é uma instância do conceito de Classe).
  - modelos do utilizador

    Os modelos utilizados para mostrar instâncias concretas de modelos UML (cf. diagramas de objectos).

Mecanismos de extensão

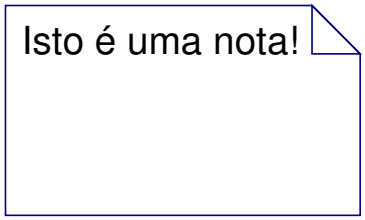
O meta-modelo define mecanismos que permitem aumentar a expressividade da linguagem.

- Restrições
  - Permitem definir restrições nos elementos de um modelo que vão para além das previstas de base.
  - Exemplo: {ordered} numa associação (para dizer, por exemplo, que a lista de sumários de uma disciplina está ordenada)
- Propriedades (*tagged values*)
  - Pares nome/valor que definem características adicionais de um elemento.
  - Exemplo: {Linguagem = Java} para definir que uma dada classe deverá ser implementada em Java.
- Estereótipos
  - Permitem efectuar a especialização semântica de elementos existentes.
  - Exemplo: <<interface>> (podem ter representação própria, neste caso: —○)

Decorações

Existem ainda dois mecanismos sem semântica especial associada que podem ser utilizados para decorar (os elementos de) um diagrama.

- Notas
  - Utilizadas para adicionar comentários ao diagrama.
  - Podem estar associadas a um elemento específico (através de um traço interrompido).



- Compartimentos extra
  - Permitem adicionar informação directamente a um elemento de um diagrama.

Desenvolvimento de Sistemas de Informação - LESI/LMCC
José Creissac Campos

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*.

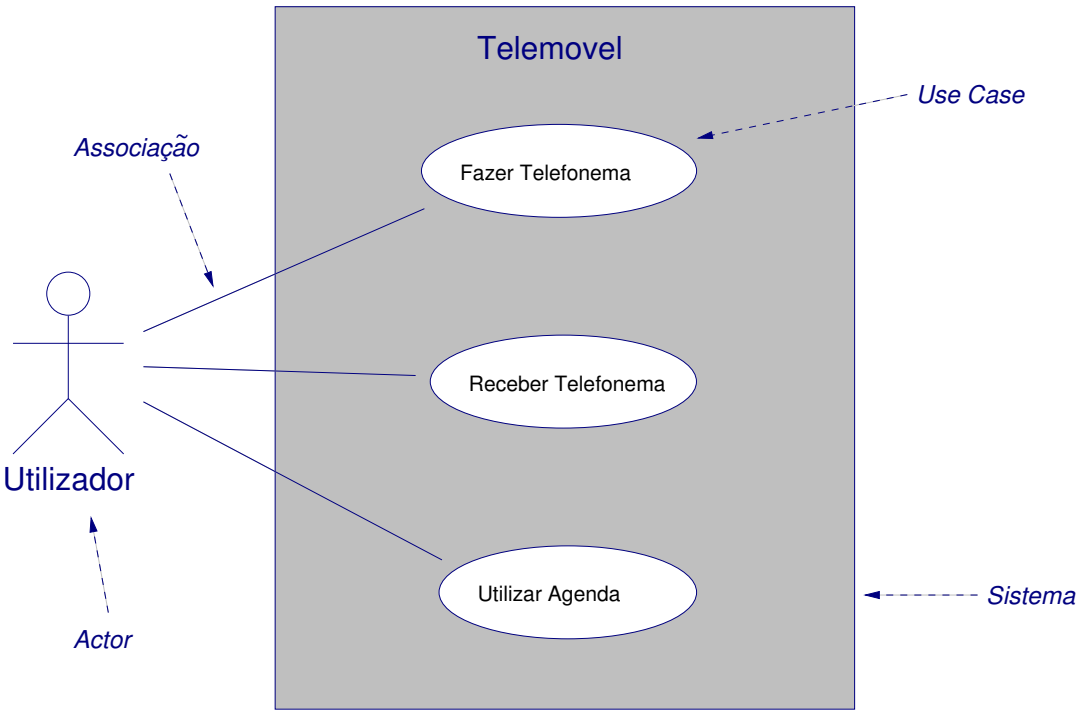
Black-box testing

- 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

- Só são representados como actores os sistemas externos que iniciem diálogo com o sistema em análise.
- Mesmo assim muito abragente.

Apenas as associações em que é o sistema externo o interessado

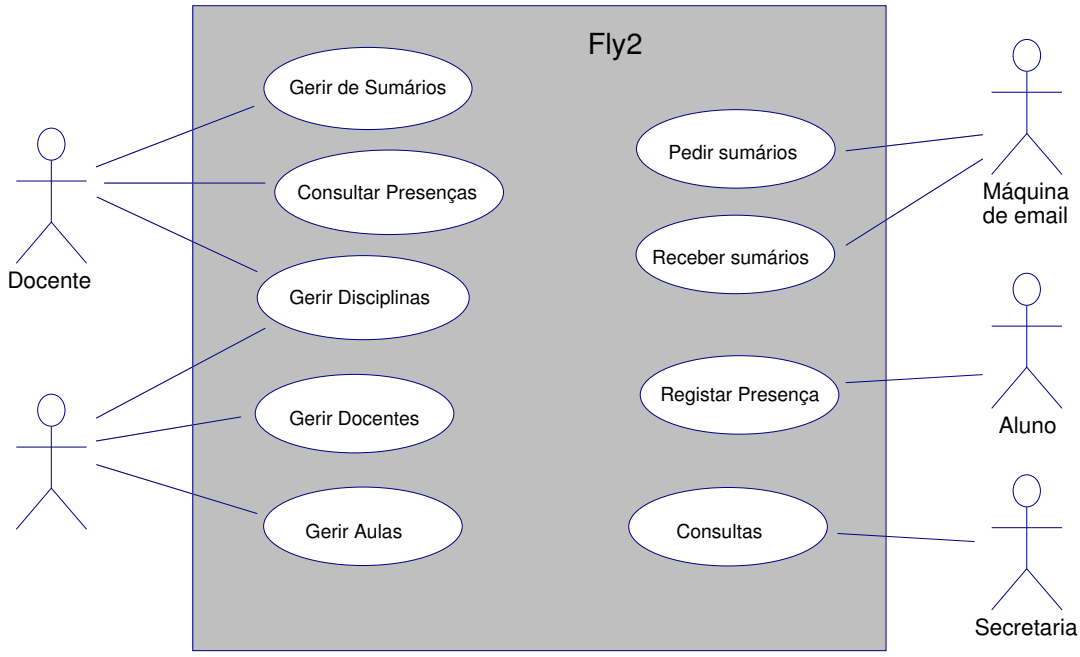
- Neste caso só são apresentados como actores os sistemas externos que necessitam de funcionalidade fornecida pelo sistema em análise.
- Usalmente esta é uma solução equilibrada.

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.

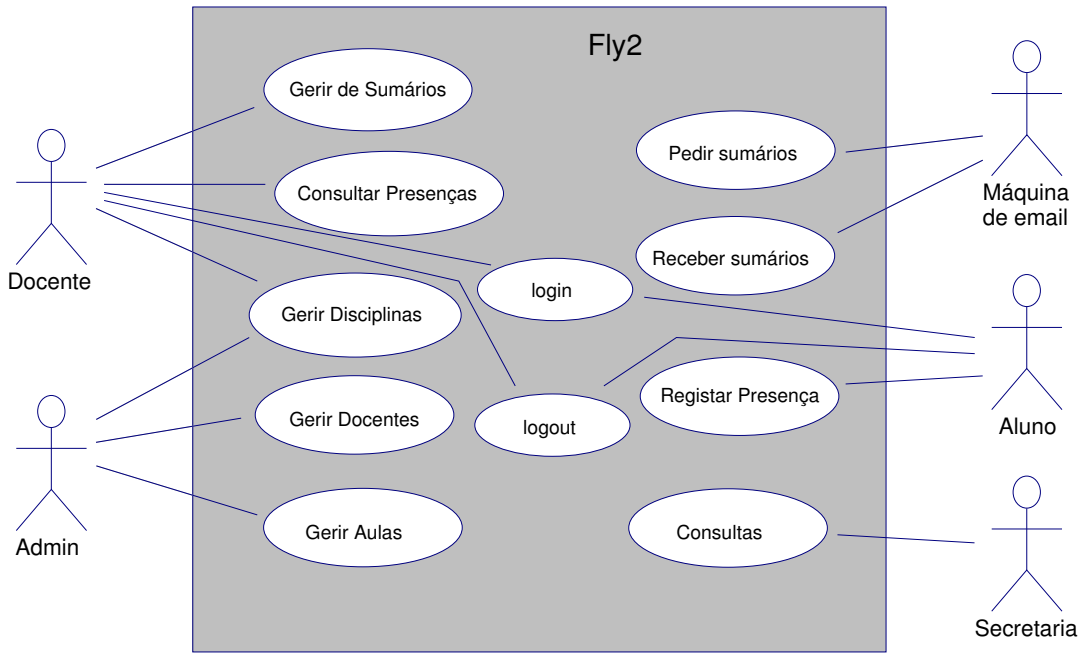
Desenvolvimento de Sistemas de Informação - LESI/LMCC
José Creissac Campos

Visão geral – versão 1



- falta mecanismo de autenticação

Visão geral – versão 2

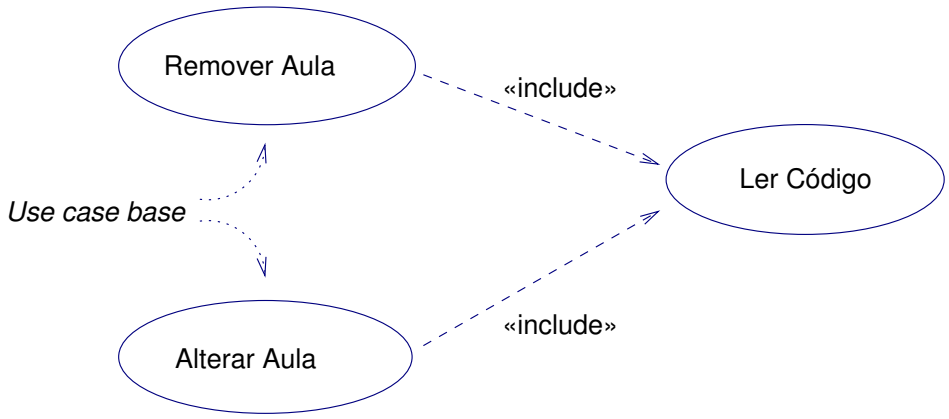


- são adicionadas pré-condições nos *use case* Gerir Sumários, Gerir Presenças, Gerir Disciplinas e Registar Presença a exigir que tenha sido feito login.

Diagramas de Use Case II — conceitos avançados

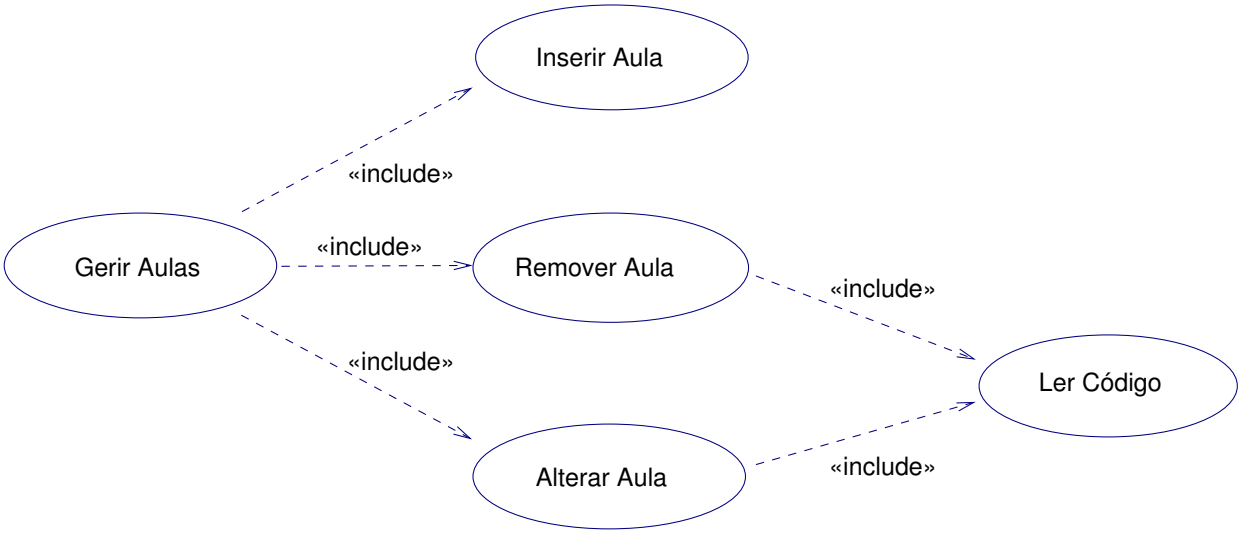
<<include>>

- 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

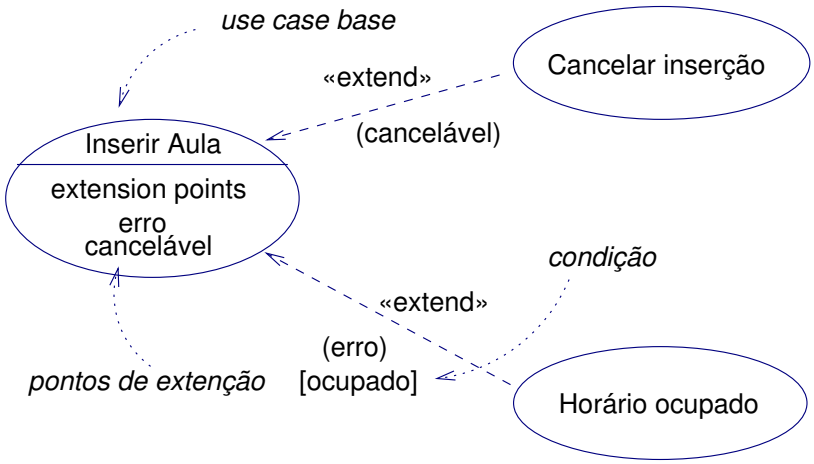
- Também pode ser utilizado para estruturar *use cases*:



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

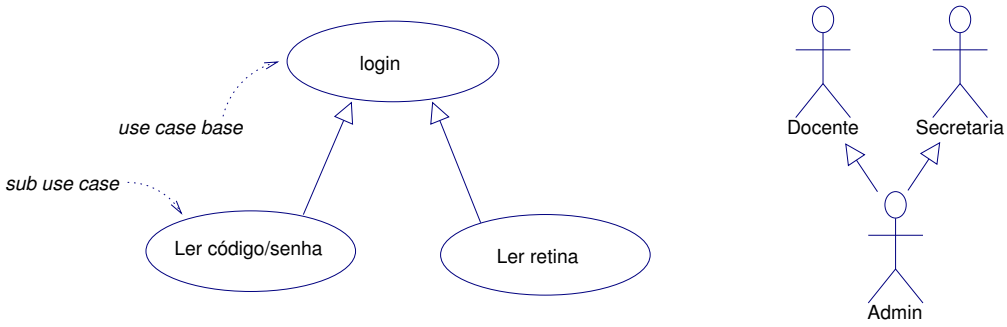
<<extends>>

- 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.
- 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!