Ciências ULisboa
Faculdade de Ciências da Universidade de Lisboa

2019/2020

Construção de Sistemas de Software

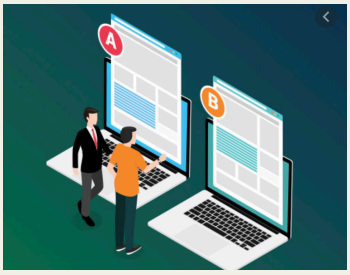
Padrões para a Camada de Apresentação
— MVC —

Padrões para as várias camadas

Apresentação	MVC Presentation Model Front controller Page controller Page template	JSP Servlet EL JavaFX
Negócio	Domain Model Transaction Script Table Module	
Dados	Row data gateway Table data gateway Active Record Data Mapper	JPA JDBC

Camada de Apresentação

- Responsável pela interface com o exterior através de oferta de serviços que tipicamente incluem a apresentação de informação
  - interação pode ser com humanos ou aplicações cliente



desktop app

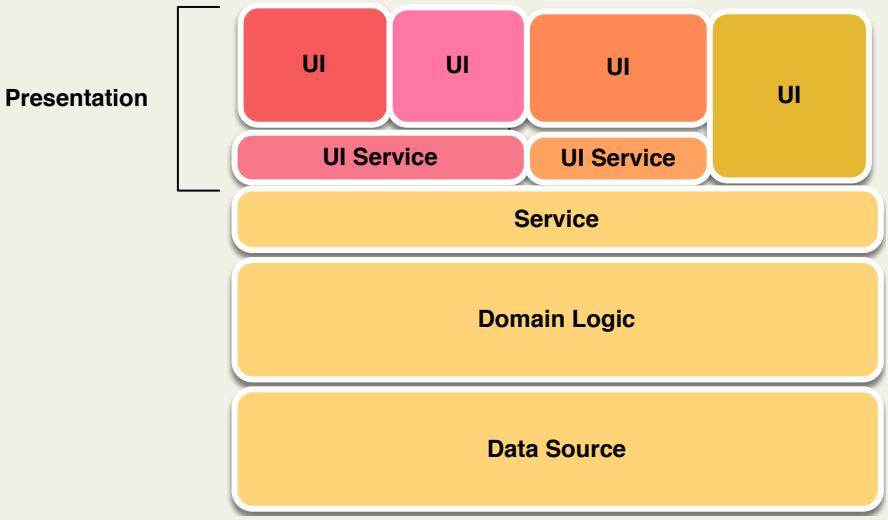
web app

mobile app

Camada de Apresentação

- Responsável pela interface com o exterior através de oferta de serviços que tipicamente incluem a apresentação de informação
  - interação pode ser com humanos ou aplicações cliente
- Pode disponibilizar diferentes tipos de interface
  - pedidos HTTP enviados por um *browser* numa ***Web Application***
  - eventos lançados por um GUI num ***Rich Client Application***
  - pedidos REST/SOAP para aplicações clientes que usam ***Web Services***

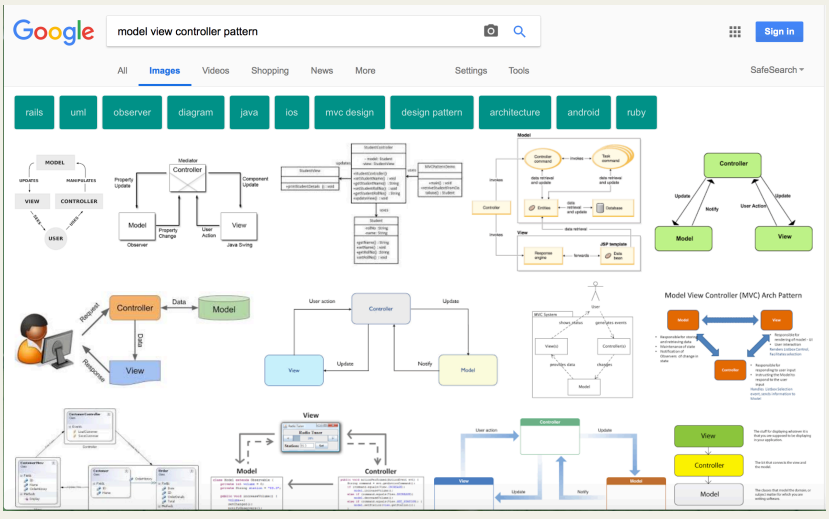
Camada de Apresentação



Ciências ULisboa | Informática

Model-view-controller (MVC)

- Padrão para organização do código dos interfaces com o utilizador introduzido no **SmallTalk-80** (*desktop apps*)
- Evolução ao nível dos ambientes alvo e das linguagens levou a que diferentes variantes do MVC fossem propostas
- Além disso, acumula muitas interpretações diferentes....



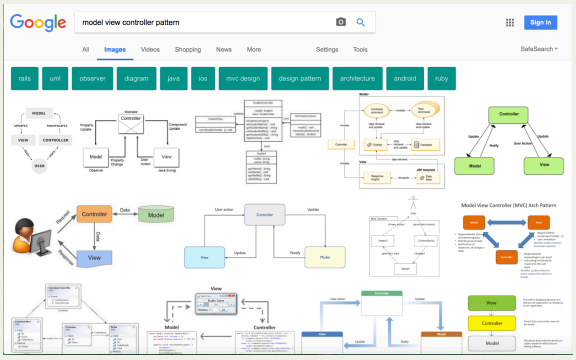
Model-view-controller

Contribui para esta multiplicidade de “interpretações”

- a **natureza** dos diferente UIs
  - interface de uma aplicação web em php

    vs
  - interface de uma aplicação desktop em java
- o **nível de abstração** a que é aplicado
  - um *UI control* como uma caixa de texto

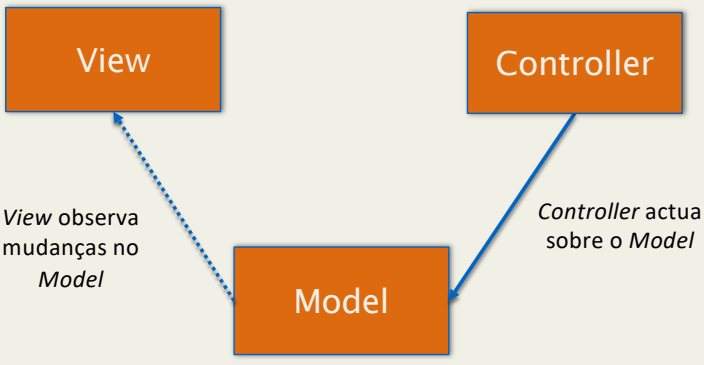
    vs
  - um formulário de uma aplicação



Model-view-controller

Lida com **output**: que dados são mostrados, com que estrutura e aparência?

Lida com **input**: determina reação aos estímulos, fazendo validações, as mudanças necessárias no *model* e decidir o que acontece a seguir (seleção da *view* e dados a apresentar)



Mantém o **estado** da aplicação e dá acesso a esses dados

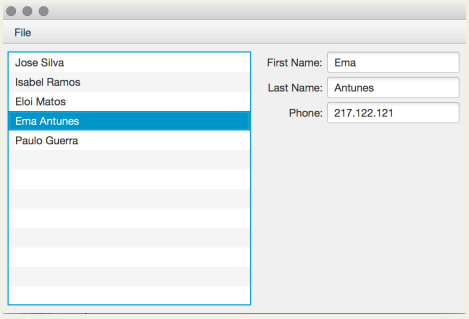
Pontos fortes

- **Separação de Responsabilidades**
  - *View*: output
  - *Controller*: input
  - *Model*: dados e funcionalidade

- ***Decoupling View - Model***
  - Os dois podem evoluir de forma separada (na medida do possível)
  - Pode-se oferecer mais do que uma *View* para o mesmo *Model* (*model* é reutilizado)
  - *Views* podem ser reutilizadas desde que se garanta que *Model* implementa o mesmo interface

Pontos fracos

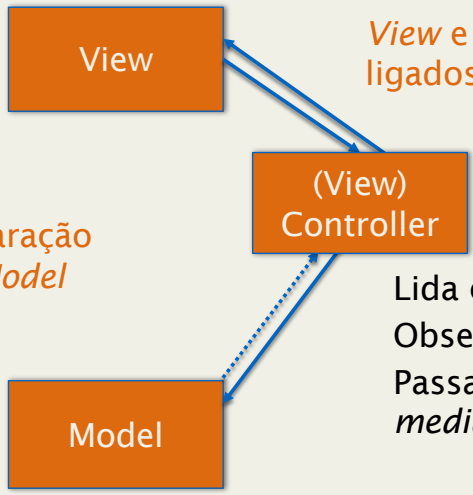
- Em certos contextos, nomeadamente GUIs, é **difícil manter** *controller* e *view* **separados**
- Não fornece uma solução para lidar com o **estado transiente da interação**, partilhada entre *controller* e *view*
  - eg., o elemento que está selecionado



- Em muitos sistemas atuais, o estado e a funcionalidade da aplicação estão **remotos**, há muitos dados envolvidos e a comunicação está longe de poder ser considerada "instantânea"

Variante MVC

Lida também com **input** de baixo nível passa a ser responsável pelo estado transiente



View e *Controller* fortemente ligados (eg, num *widget*)

Mantida a separação entre *View* e *Model*

Lida com input de **alto nível**

Observa mudanças no *model*

Passa a ter mais o papel de *mediator*

Presentation Model

- Representar **o estado e comportamento da interação** de uma forma que **é independente dos componentes GUI** (*widgets*) usados no interface
- Este elemento, chamado ***PresentationModel***, é quem se coordena com a camada de negócio
- A ***View*** ou guarda todo o estado no ***PresentationModel*** ou sincroniza-se frequentemente com ele
- Permite resolver o problema da responsabilidade de manter e guardar o estado transiente da interação
- Facilita o teste