MVC architectural pattern and derivatives

Guillaume Communie – CS group – September 30th 2020

Summary

- Introduction
- MVC, MVP, MVVM
- Implementation of MVP
- Hands-on

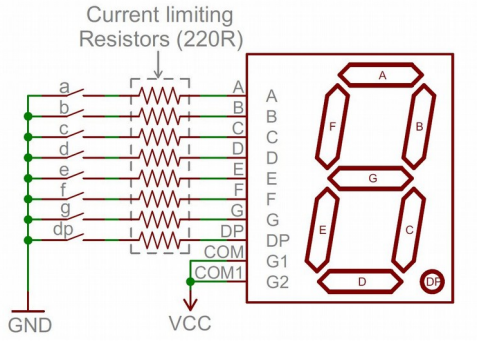
Introduction

- MVC/P: model – view – controller/presenter
- architectural design pattern
- development of user interfaces
- MVC was created in the 80’ with Smalltalk language
- Many derivatives

  → one goal, separation of concerns

Introduction – Separation of concerns

- example: a digital clock



Presentation

Model

Introduction – Separation of concerns

Presentation

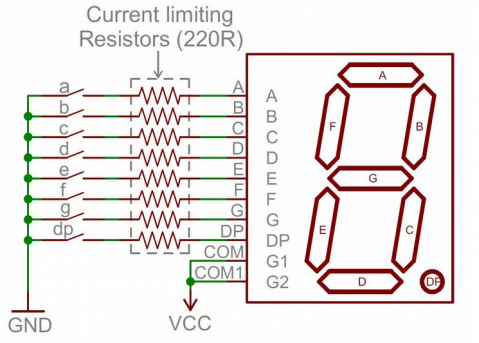
- example: ~~a digital clock~~ an analog clock

Model

Introduction - Principle

- Separation of concerns
  - Business related part (data and application logic)
  - Visual components (widgets, screens and presentation logic)
- Advantages
  - Multiple developers can work simultaneously
  - Facilitate unit testing
  - Change only the visual part, the data management system, ...

Introduction – Derivatives



Presentation

- Many variation depending on the way presentation and model interact

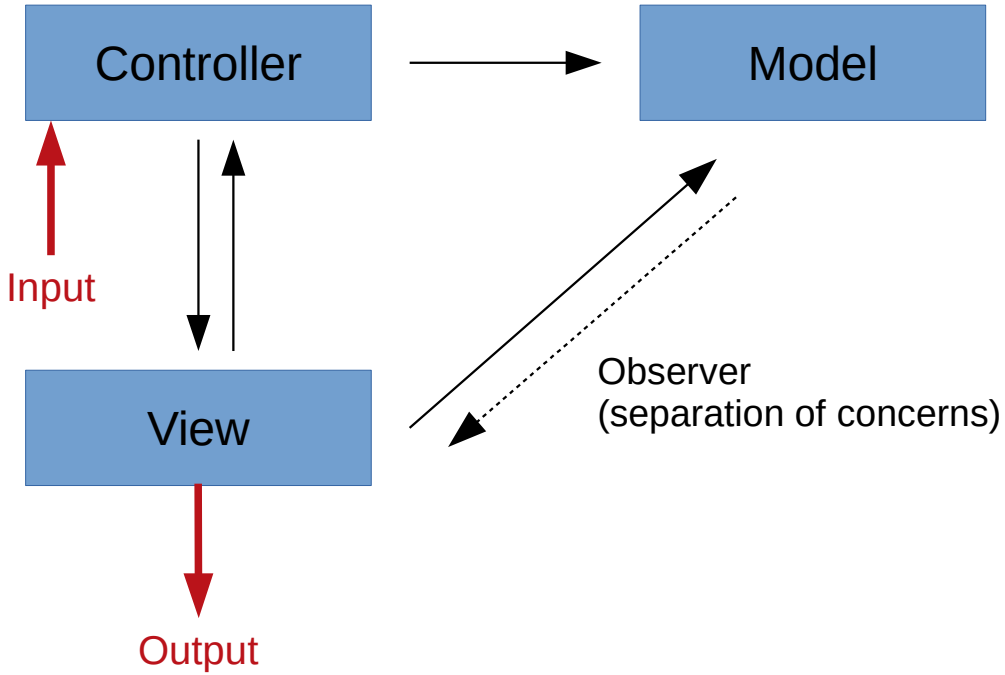
- Controller
- Presenter
- Viewmodel
- ...

Model

Model – View – Controller (original)

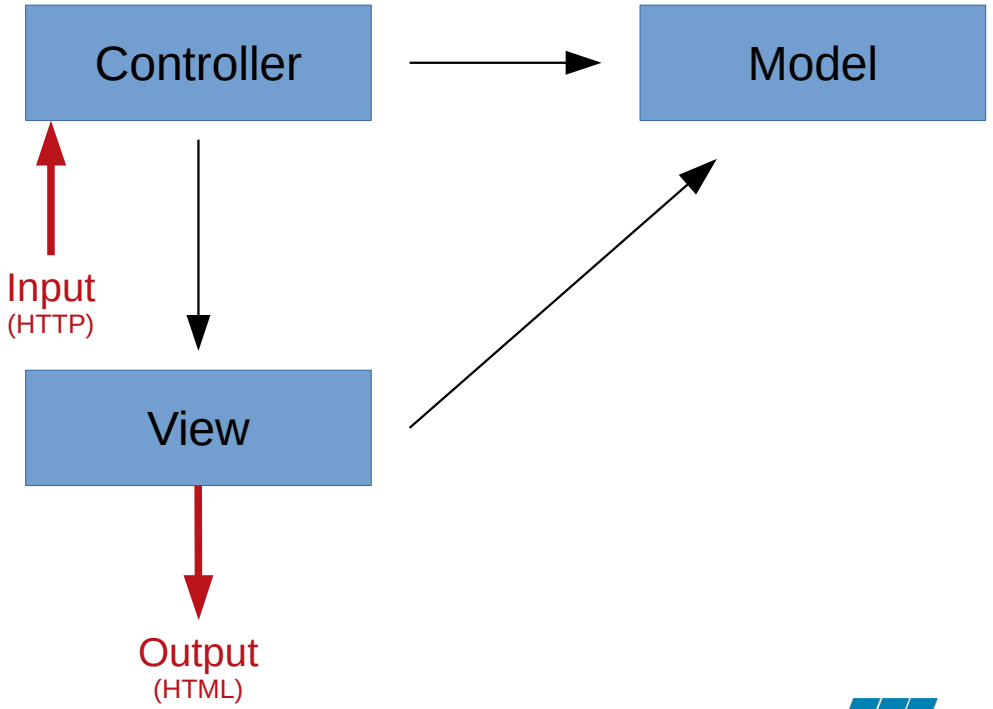
- Model:
  - data and business functionalities
- View:
  - visual representation of the model (output)
- Controller:
  - intercepts user inputs (mouse, keyboard)

- V and C interact with M (read, write)
- M can use the observer pattern to notify V



Model – View – Controller (web)

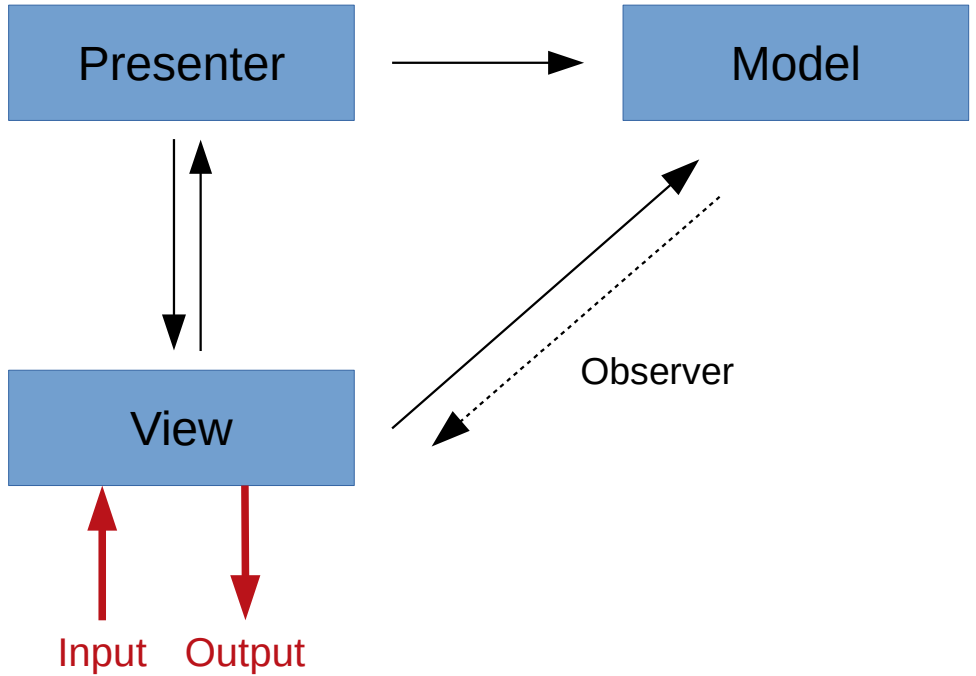
- View:
  - HTML (JSP, ASP, ...)
  - visual representation of the model (output)
- Controller:
  - intercepts user inputs (web requests)
  - select the view and provide it with the appropriate state



Model – View – Presenter

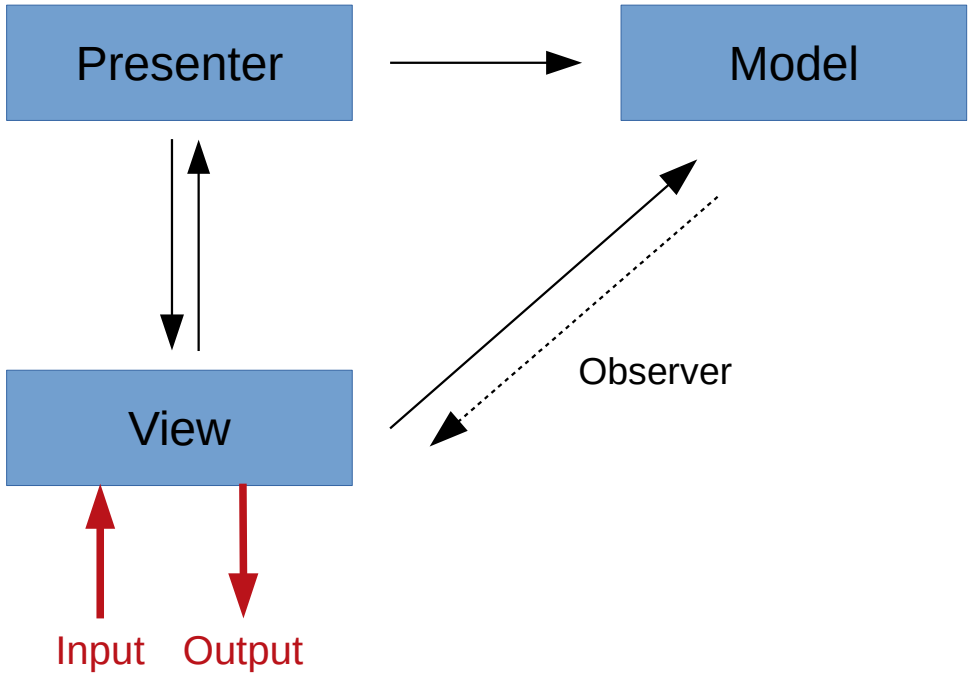
- Modern widgets can deal with inputs.
- Controller → Presenter
- View:
  - visual representation of M
  - intercept user inputs
  - Presentation logic
- Presenter:
  - Updates M
  - Presentation logic

- V and P share the presentation logic



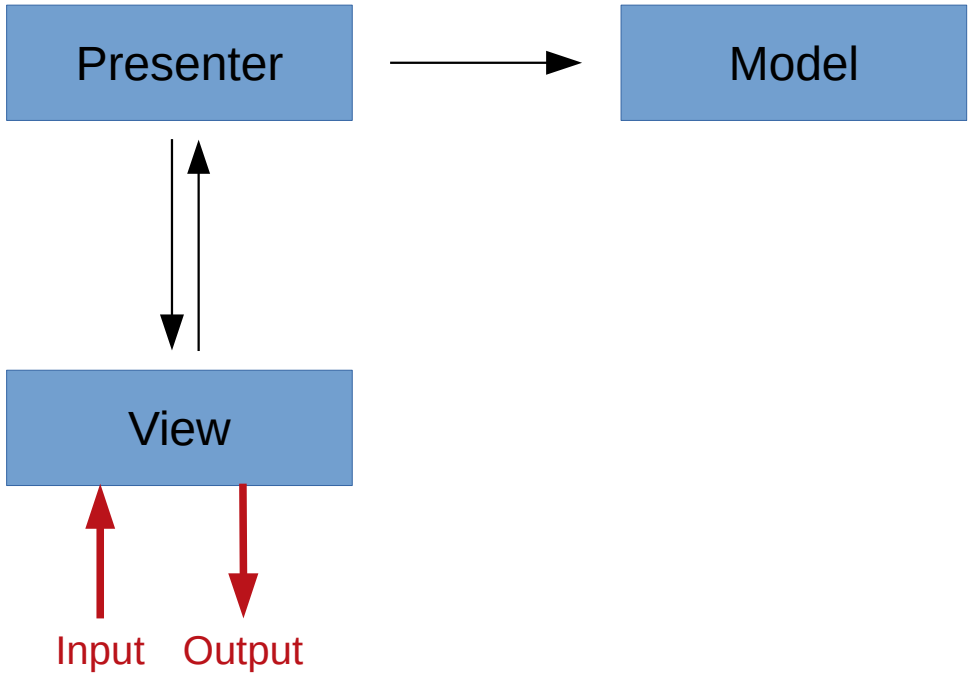
MVP - supervising controller

- View:
  - simple presentation logic
  - update from M state
- Presenter:
  - complex presentation logic (that need unit testing)



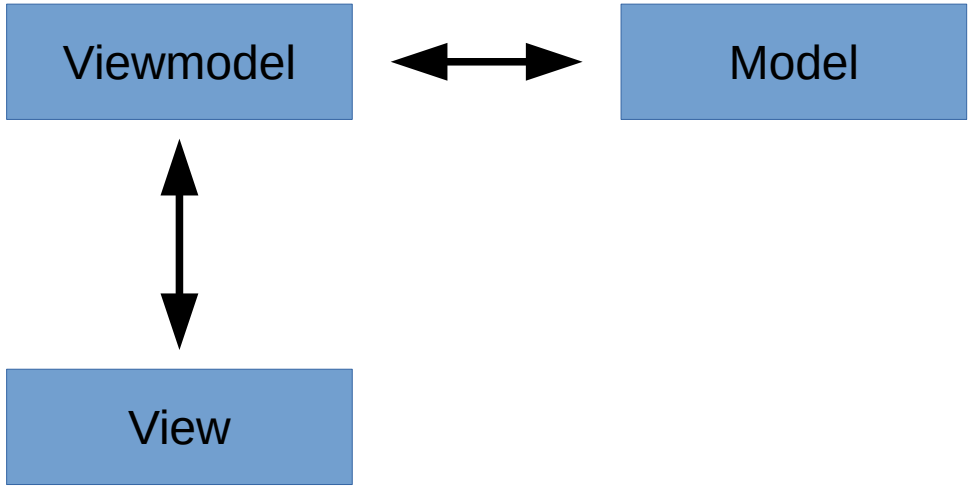
MVP - passive view

- View:
  - Informs P for every changes
  - Updated by P only
- Presenter:
  - Read and write access to M
  - all presentation logic
- No dependencies between V and M
- Unit testing

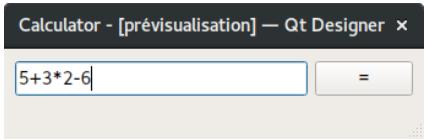


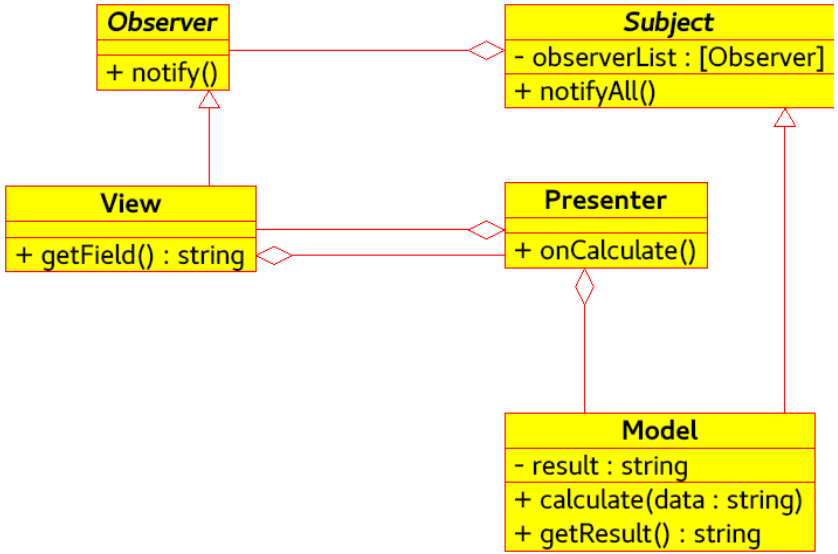
Model – View – Viewmodel

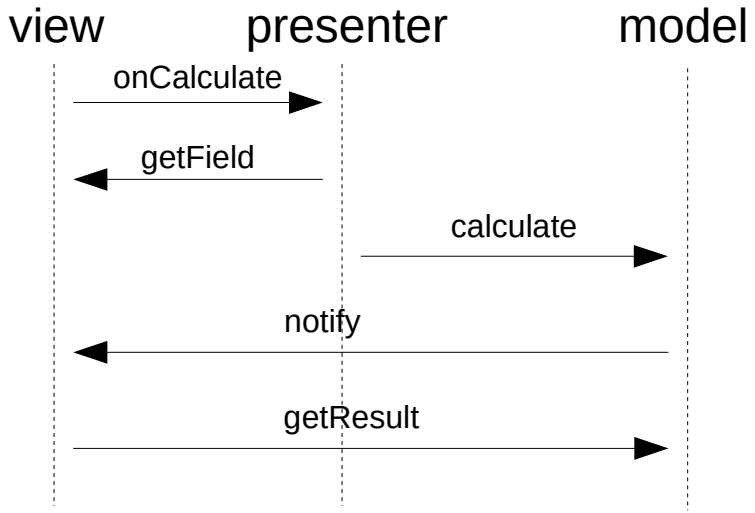
- From Microsoft. Sometimes called Model – View - Binder
- View contains only the graphical part (html, xaml, …). Responsibility of a designer team
- Viewmodel contains the view logic and converts the model data to be understood by the view



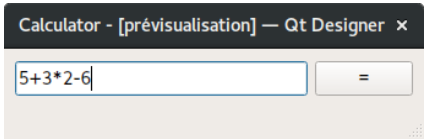
Implementation of MVP - SC

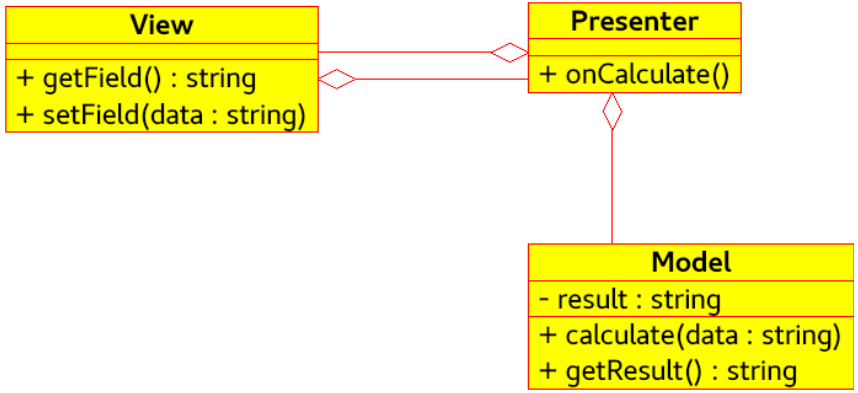


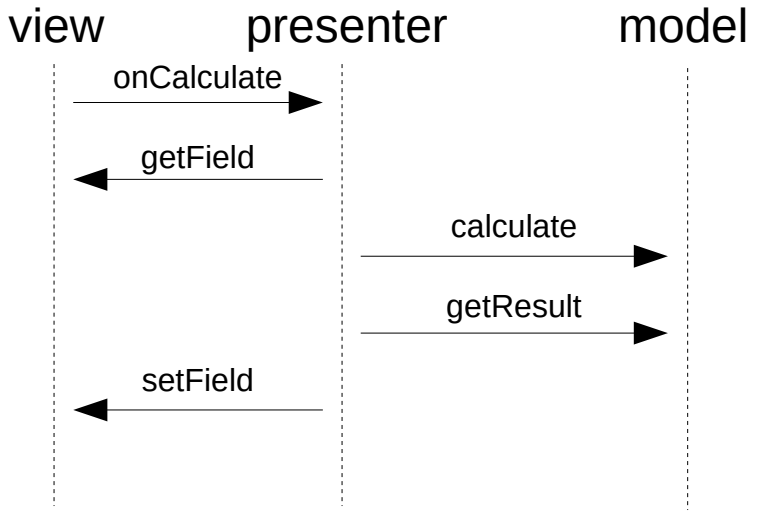




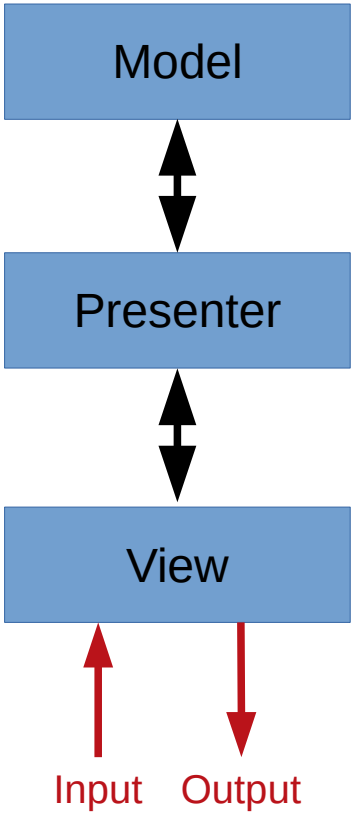
Implementation of MVP - PV





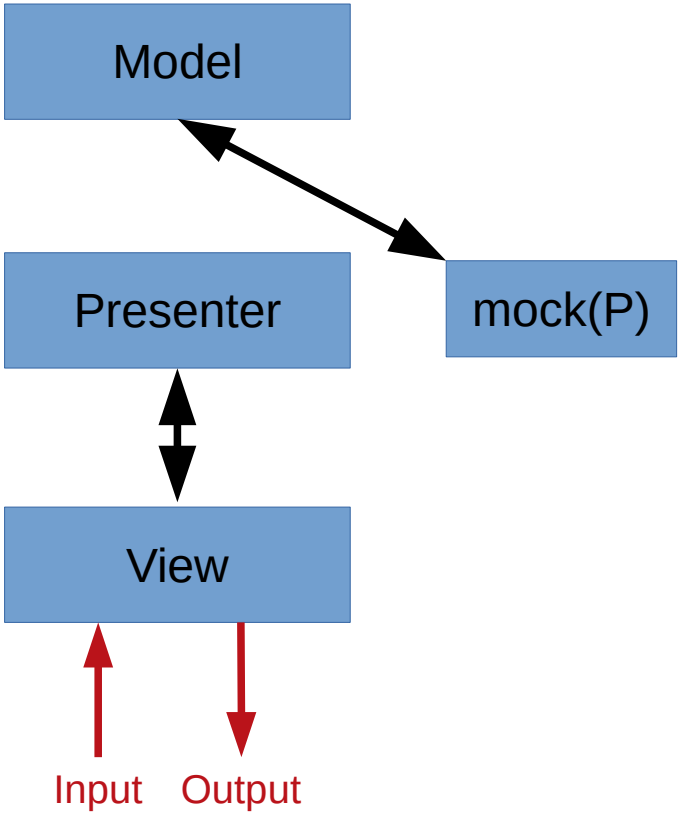


MVP – Unit testing



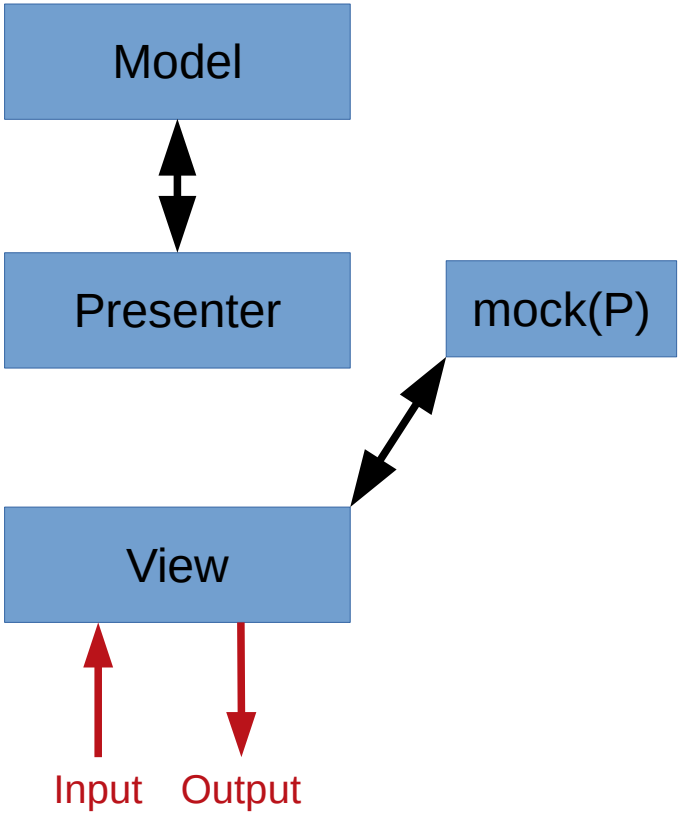
- M, V and P communicate via a common interface

MVP – Unit testing



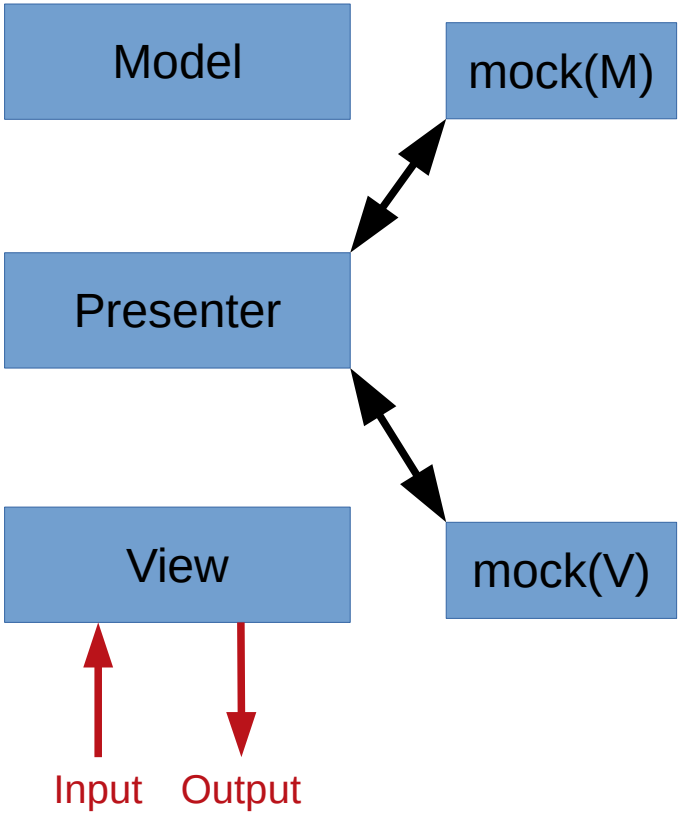
- M, V and P communicate via a common interface
- M can be tested by mocking P

MVP – Unit testing



- M, V and P communicate via a common interface
- M can be tested by mocking P
- V can be testes by mocking P

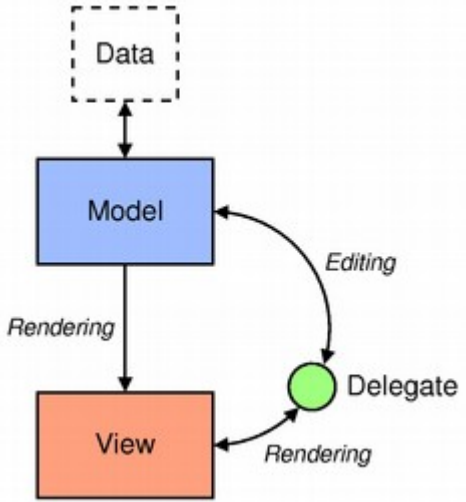
MVP – Unit testing



- M, V and P communicate via a common interface
- M can be tested by mocking P
- V can be testes by mocking P
- P can be tested by mocking V and M

Qt Model – View pattern

- Qt uses a derivative of the MVC pattern
- Model – View - Delegate
- Model: contains the data or the way to collect them (file dialog)
  → QAbstractItemModel
- View: presentation logic
  → QAbstractItemView
- Delegate: some methods to help in the rendering / editing
  → QAbstractItemDelegate



Conclusion

- Separation of concerns facilitate the development, testing and evolution of the application
- “Real” MVC not really used anymore. Except for web applications
- Two main MVP patterns: passive view and supervising controller
- Some drawbacks:
  - Can be difficult to read
  - Several times ~ the same method (getValue, setValue, onValueChanged, …)
  - Be careful with updateViewFromModel type methods

Thank you for you attention

Questions?

Hands-on?

INSTITUT LAUE LANGEVIN

NEUTRONS FOR SOCIETY

Hands-on

https://github.com/gui-co/mvc.git