MVC ARCHITECTURE

ESI-Algiers

n_bousbia@esi.dz

Plan

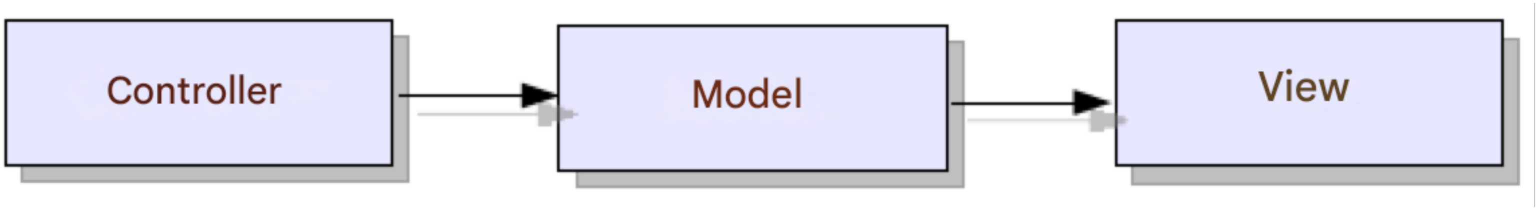
- Motivation
- MVC
  - Definition
  - Architecture (Model - View - Controller)
  - Advantages - Disadvantages
  - MVC and design patterns
- Variants-Derivatives of the MVC
  - MVC2
  - MVP
  - MVVM

Historical

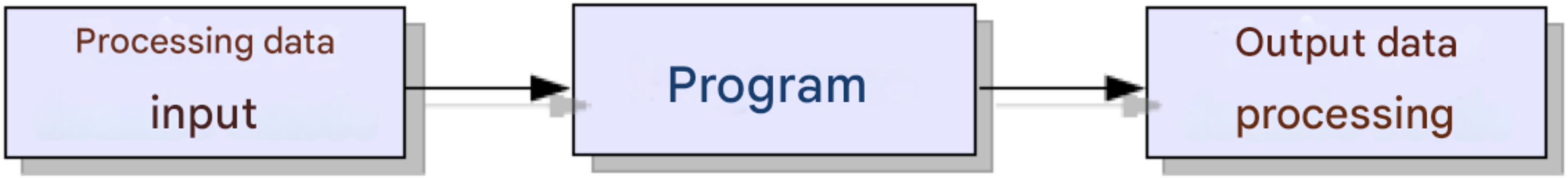
- The idea of separating the user interface from the rest of the code emerged in the late 1970s, when the main interface change was starting to become essential:
  - Transition from the textual interface to a "graphical" interface
  - How can existing software continue to work using this new display type?
- In 1979 Trygve Reenskaug (Smalltalk team), proposed a solution to the problem:
  - Highlights the problem of data interdependence as well as the need to consult the data
  - Originally called MVE (Model View Editor) then MVC (Model View Controller)

Motivation: Why MVC?

- Break down a GUI into three distinct parts



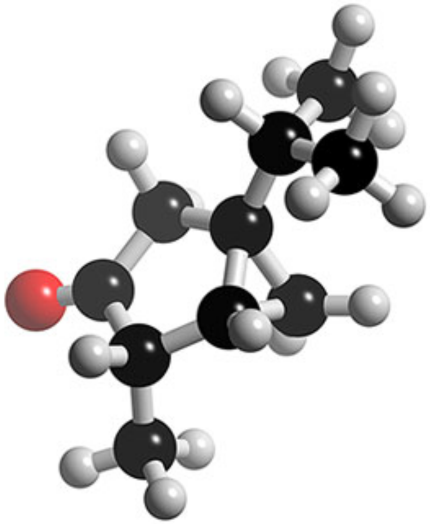
- Schema modeled on the conventional processing of a program



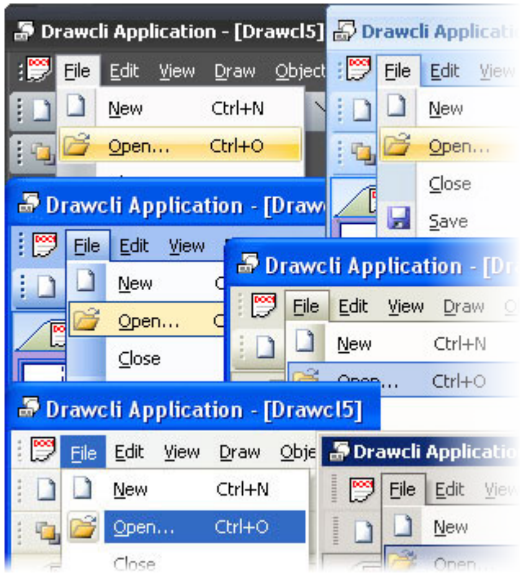
MVC

- The Model-View-Controller (MVC) is a design pattern widely used in the design of HMIs, in particular in web sites and applications.
- Often presented as an architectural pattern
- It shows great interest in teamwork and the distribution of tasks.
- It allows better source code structuration and separation into three distinct parts.

Model:



View:



Controller:

General principle

- Aim
  - Isolate the data from its presentation
  - Distinguish viewing from modification
  - Lowest possible coupling
- Principle
  - Separate Of Concern
  - The following 3 components of data are distinguished and separated
    - The model (its structure)
    - The view (its representation for display)
    - The controller (the means of changing the value)

The Model

- The **Model** is a set of components containing the data and the logical operations allowing the manipulation of this data.
  - Represents application behavior
  - Business data as well as communication to the data structure (RDB, NoSQL, XML, etc.).

- Role:
  - Encapsulate application data
  - Carry out processing on this data
  - Be independent of views and controllers

- Consequences
  - The model interface offers
    - Updating data
    - Consultation of data
  - Model results lacking any presentation
  - Maintain a list of listeners (views and/or controllers declare themselves as listeners)
  - Notify listeners when data is modified
  - Implementation of the Observer design pattern: Model, is Observable (Subject)

The View

- The view is a visual representation of the application data (UI)
- Role
  - Display model data
  - Stay up to date when the model is changed
  - Retrieve all user events.
  - Transmit user events to the Controller.
- Consequences
  - Multiple views possible for the same data
  - Must register as a listener at model level (depends on the implementation)
  - The view does not perform any processing

The Controller

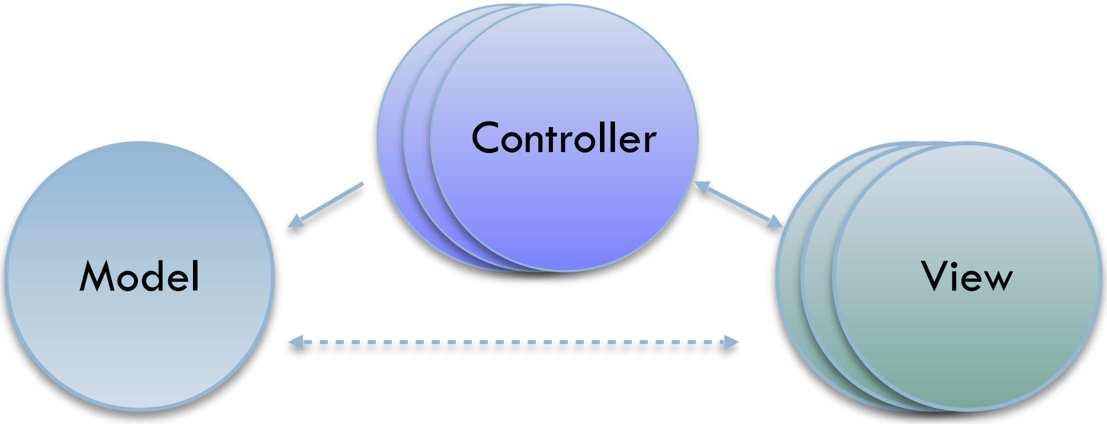
- The Controller receives the events sent by the View, and during a modification request asks the model to synchronize the new value

- Role
  - Allow the user to modify the data encapsulated in the model
  - Receives user events
    - Via the view
    - It analyzes the request and applies control processing
  - Triggers the actions to be performed
    - It transmits messages to the model
    - If necessary, it informs the view to update itself (Handling synchronization events)

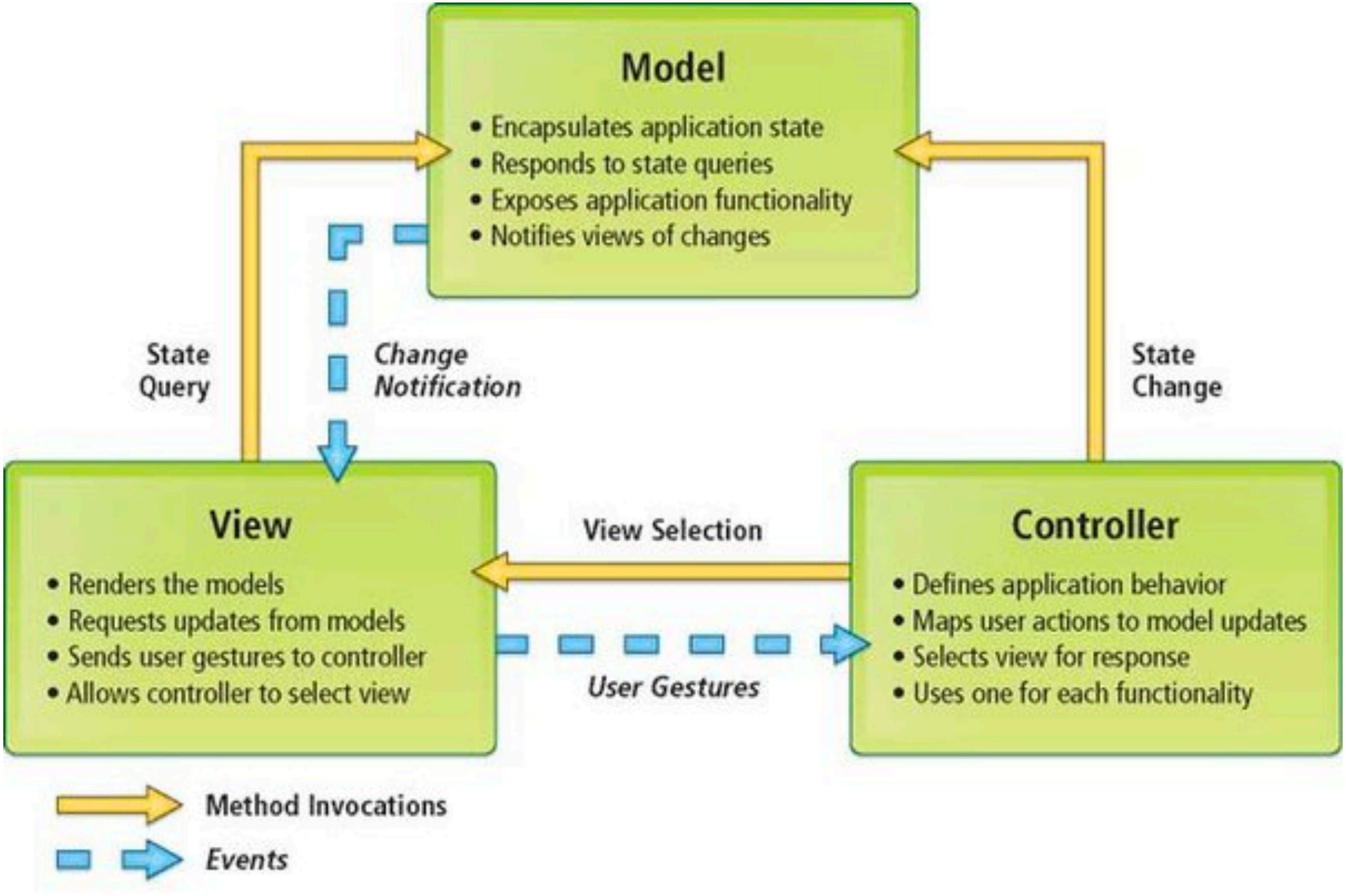
- Consequences
  - Should optionally register as a model listener to be updated if the model is changed (depends on the implementation)
  - Must call model accessors based on user actions

Features

- Model level classes should completely encapsulate information and behavior related to the logic of the application
- View classes are responsible for input and output, but do not contain application-related data or functionality
- A model can have multiple views and controllers
- A controller can be associated with multiple views, but a view is only associated with one controller



MVC architecture



Model

- Encapsulates application state
- Responds to state queries
- Exposes application functionality
- Notifies views of changes

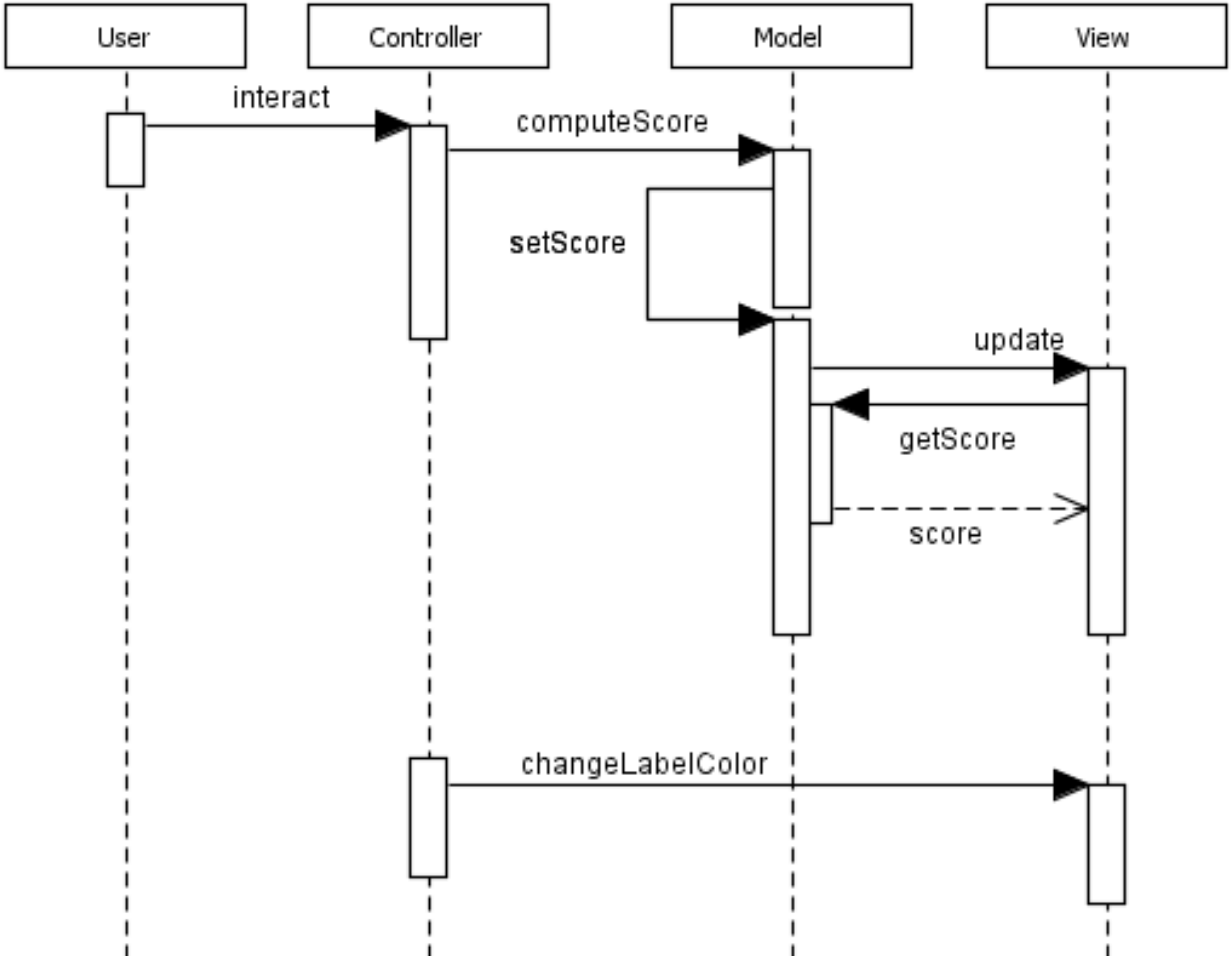
View

- Renders the models
- Requests updates from models
- Sends user gestures to controller
- Allows controller to select view

Controller

- Defines application behavior
- Maps user actions to model updates
- Selects view for response
- Uses one for each functionality

MVC: Collaboration Example



Advantages of MVC (1/2)

- Avoid storing data in multiple locations.
- Proposes a coherent definition of the model, focused on domain processes, and not focused on human-machine interfaces,
- Easier maintenance by decoupling view(s)-model
- Ability for the model to inform the view of changes incrementally (for example, adding a row to a table).

Advantages of MVC (2/2)

- Modularity
  - Clear separation between the program data and the graphic part displaying the data.
  - Allows separation of development tasks (one person can take care of the views, another the model, etc.).
  - Allows model execution independent of the interface

- Extensibility
  - Allows the addition of new views without changing the model.
  - Ensures greater portability by making it easier to migrate the model to a new type of interface management environment
  - Minimizes the impact of changes in interface-related specifications on the application logic level

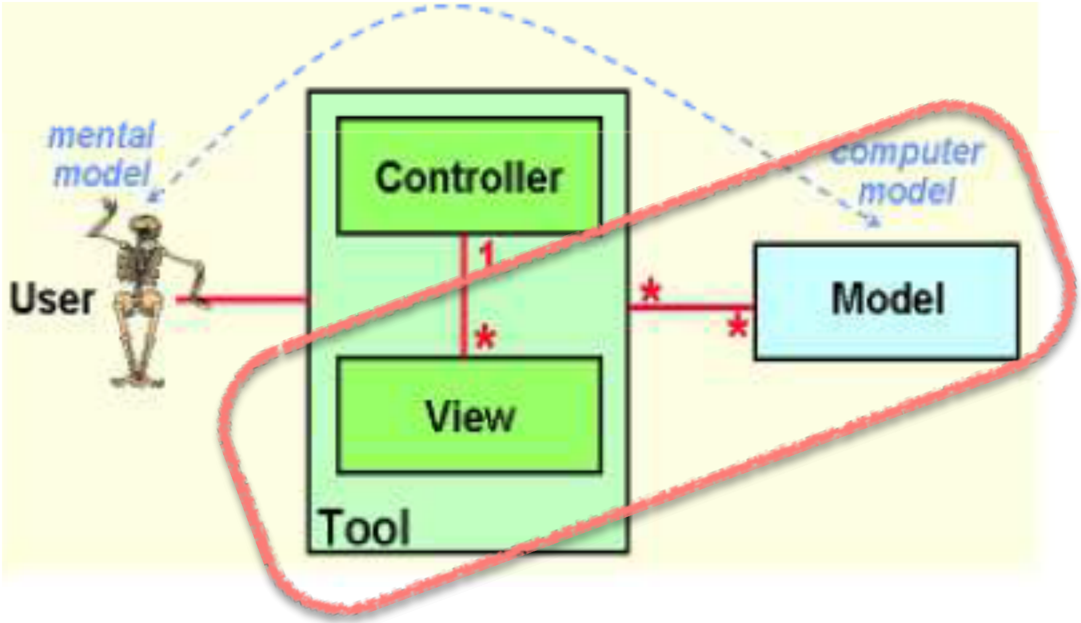
Disadvantages

- The major disadvantage of the MVC model is that it is not suitable for small-scale projects.
  - To respect the logic of separation of components, it is necessary to create a file/directory for each of them, thus unnecessarily increasing the size of the application.
- Cost of component communications
- Potentially blurred boundary between components

MVC and GoF Patterns

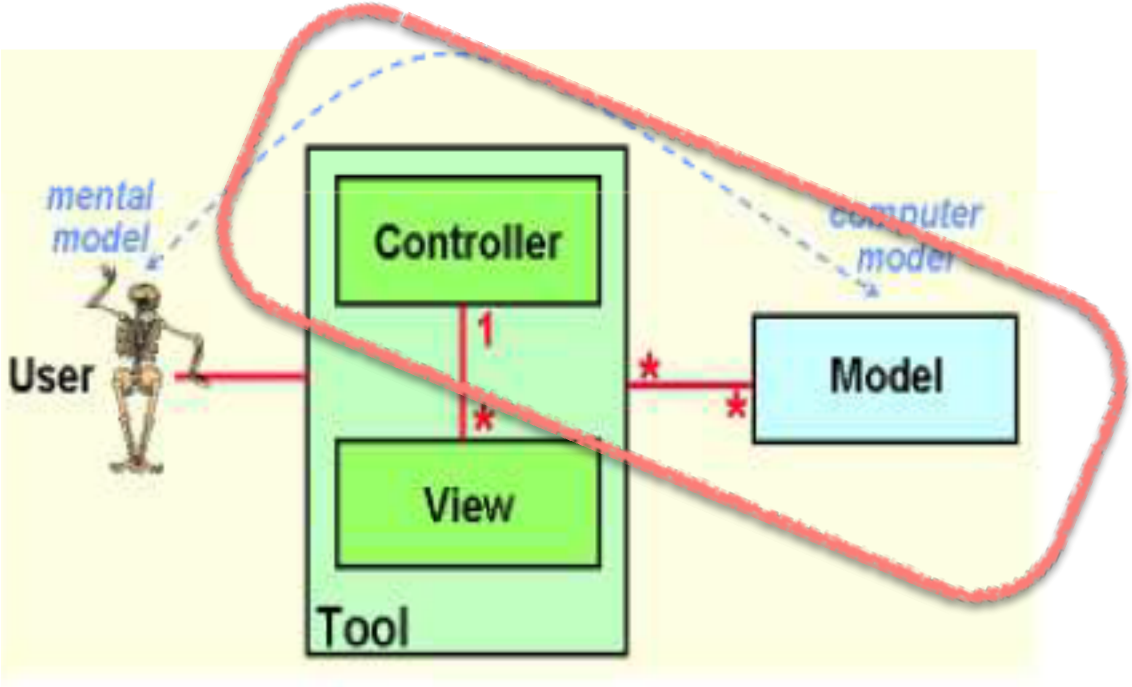
- Communication between the Model and View of the MVC model is an example of many situations requiring indirect communication between entities in a system.
- This type of communication requires the introduction of specialized Patterns: Observer Pattern, Mediator Pattern.
- Other patterns are also used for graphical interfaces: Strategy, composite, decorator, etc.

MVC -Observer (View-Model)



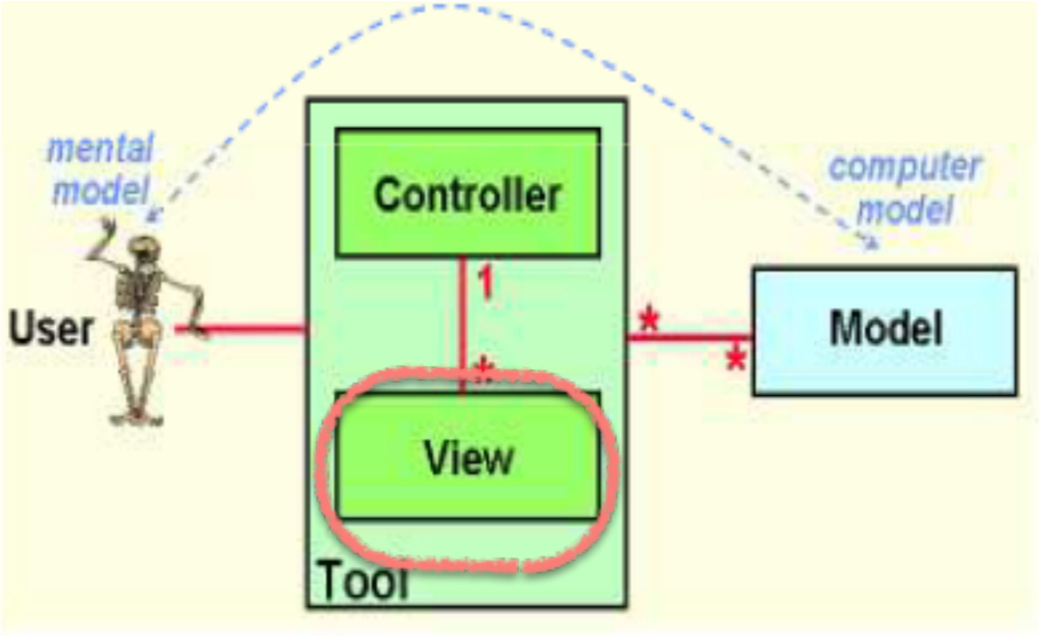
- The application must be able to manage several views simultaneously.
- As soon as the model is modified, the views are redrawn appropriately.

MVC-Observer (Controller-Model)



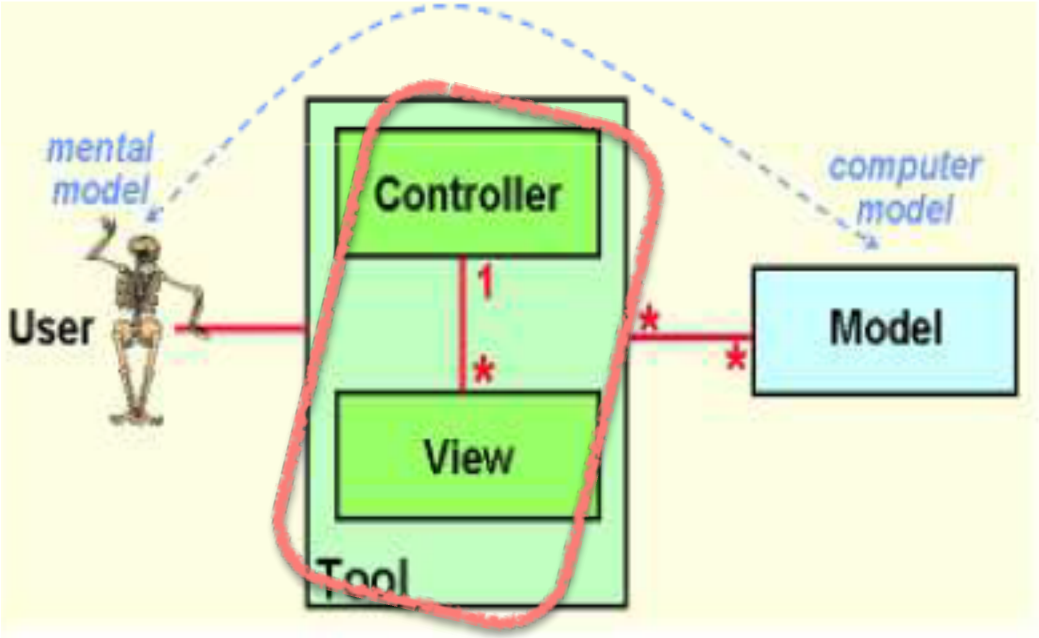
- It can happen that the state of the model has an impact on the controllers. For example, depending on the model, certain options may not be available (menus, check boxes, etc.).
- Controllers must therefore be notified of all changes of the model.

MVC-Composite (View-View)



- We consider that each component can be the subject of a view and that the view in question can contain subviews representing the subcomponents

MVC-Strategy (Controller-View)

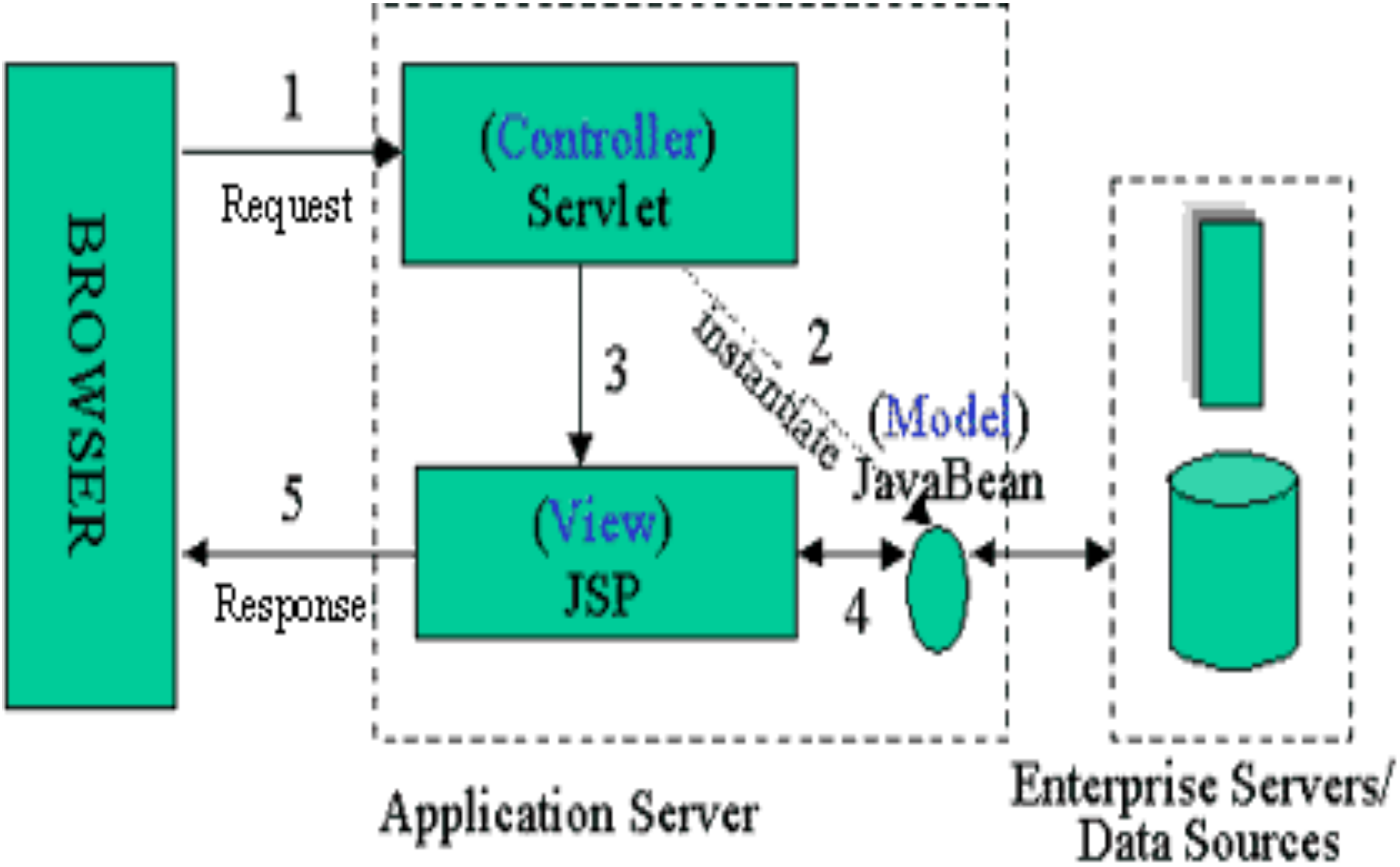


- MVC also allows to change how a view responds to user input without changing its visual presentation.
- The same view can be used with different controllers to have different behavior in different parts of the system.
- The view is an object that is configured with a policy. The controller provides the policy.

Best practices to apply

- The most common mistake is giving too much importance to the controller
  - Note: it receives a request, applies a control processing, delegates operations and returns a response.
- A few tips
  - All validation operations for a user's data must be outsourced to a dedicated validation service.
  - Help and utility functions also need to be outsourced.
  - In web application, don't put HTML directly in the controller (even in the response). Return a view if applicable.
  - The processing of business data must of course be carried out in the model.
  - Create as many controllers as there are responsibilities (Authentication, User, etc.)
  - Create as many models as entities of your database (Language, Article, Category, etc.)

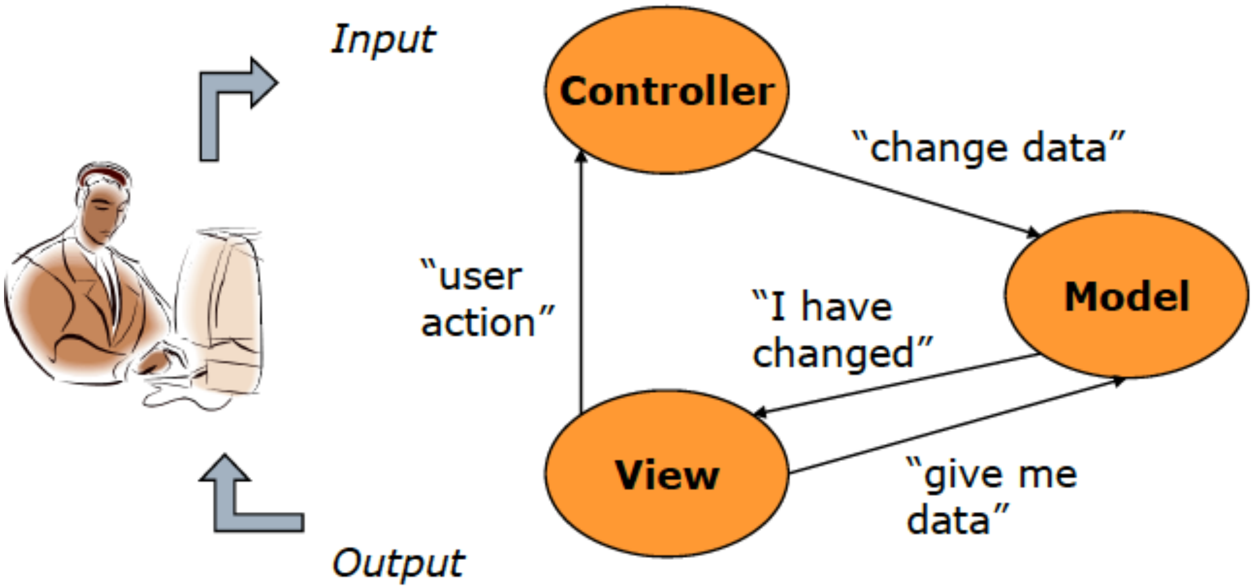
MVC Example


BROWSER
1
Request
(Controller)
Servlet
3
Instantiate
2
(Model)
JavaBean
5
(View)
JSP
Response
4
Application Server
Enterprise Servers/
Data Sources

Variants of MVC

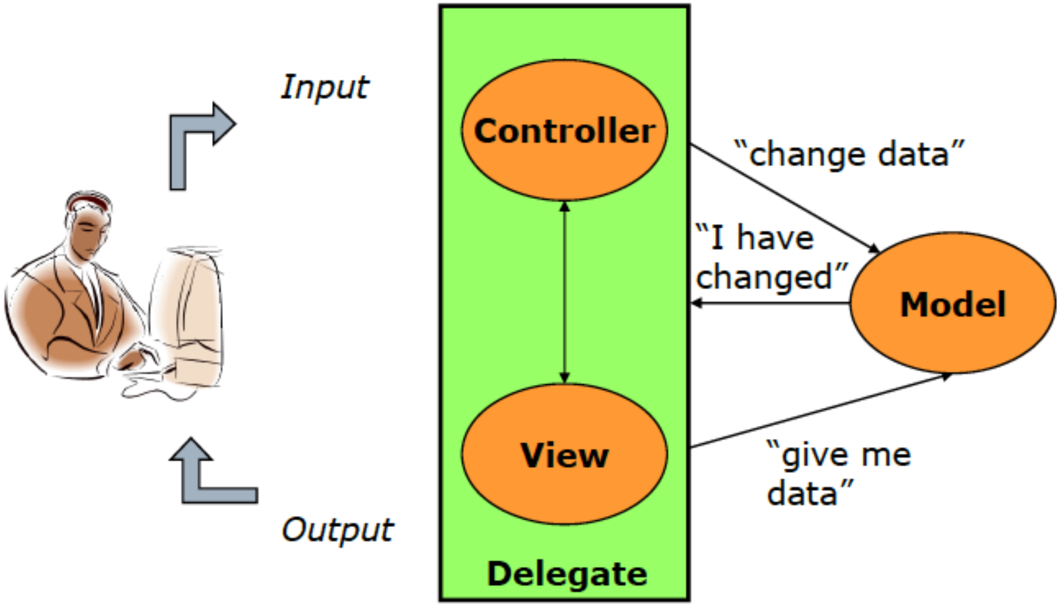
1. Supervising Controller

- The original MVC pattern uses this implementation variant.
- The controller communicates with the model and the view.
- Model and view can communicate directly (the view is an observer of the model)
- The model notifies its associated views and controllers when a state change occurs.
- The view requests data from the model to update the representation to the user



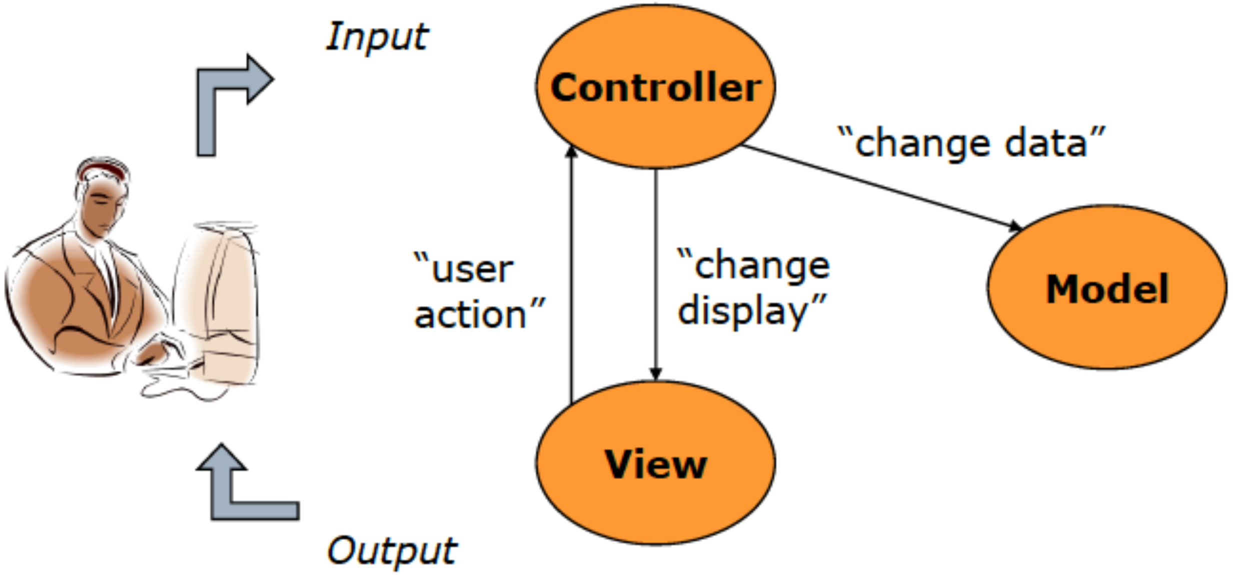
2. Delegate Model

- The pattern is the same as original MVC
- Delegate is responsible for input and output: it is a combination of the view and the controller
- Also called: UI-Model, Document-View



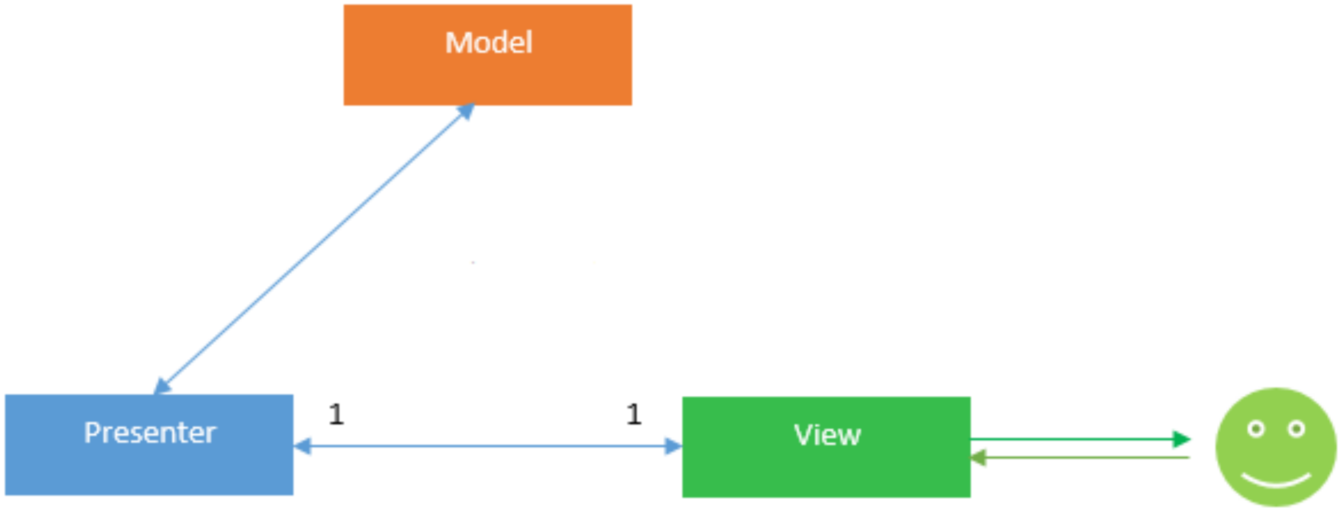
3. Passive View

- The controller is the only component that can communicate with the model and the view
- There is no direct communication between Model and View
- Also called: Model-View-Adapter



4. MVP: Model-View-Presenter

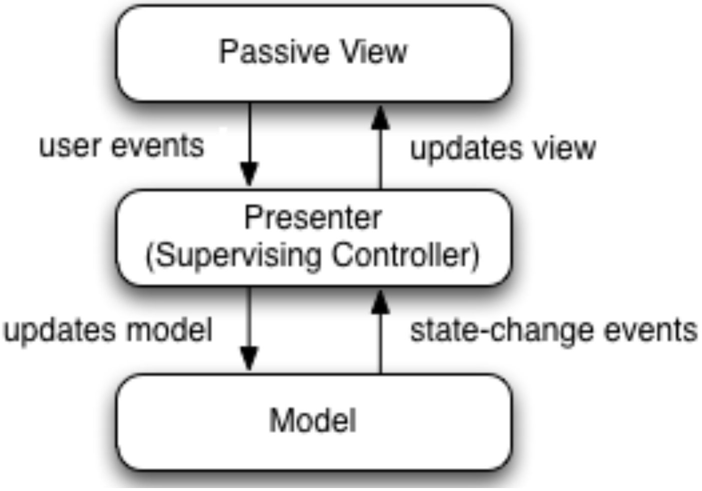
- The most used pattern is the MVP (Model-View-Presenter)
  - even if users often wrongly believe they have followed an MVC pattern.
  - Model and View have the same role in MVP and MVC patterns, but Presenter differs from Controller



MVP Pattern

4. MVP: Model-View-Presenter

- The Presenter acts as an intermediary in both directions between the View and the Model, because it will be used to “present” the data on one side or the other (Passive View).
- The user interacts with the view
- **There is a one-to-one relationship between the view and the Presenter (a view/presenter, a presenter/view).**
- The view has a reference to the Presenter but not to the model
- This pattern is widely used in web application



5. MVVM: Model ViewViewModel

- In 2004, Microsoft added to its .NET library the MVVM model (Model View ViewModel)
  - Their objective is the unification of Views using a ViewModel.
  - A design pattern based on MVC and MVP
  - has been widely used on the web by JavaScript libraries.
- The idea is to ensure that the modeled data is common to any application and this genericity of the interpretation of the data is done in the ViewModel.
  - The goal of MVVM is to break that coupling between the view and the logic that the controller element manages in the MVC pattern.
  - To accomplish this, MVVM inserts the ViewModel element as an abstracted representation of the logic contained in both the view and the model. Thus, the front end and back end of the application can fetch the information they require from the ViewModel without needing any awareness of each other.

MVVM: Model-View-ViewModel

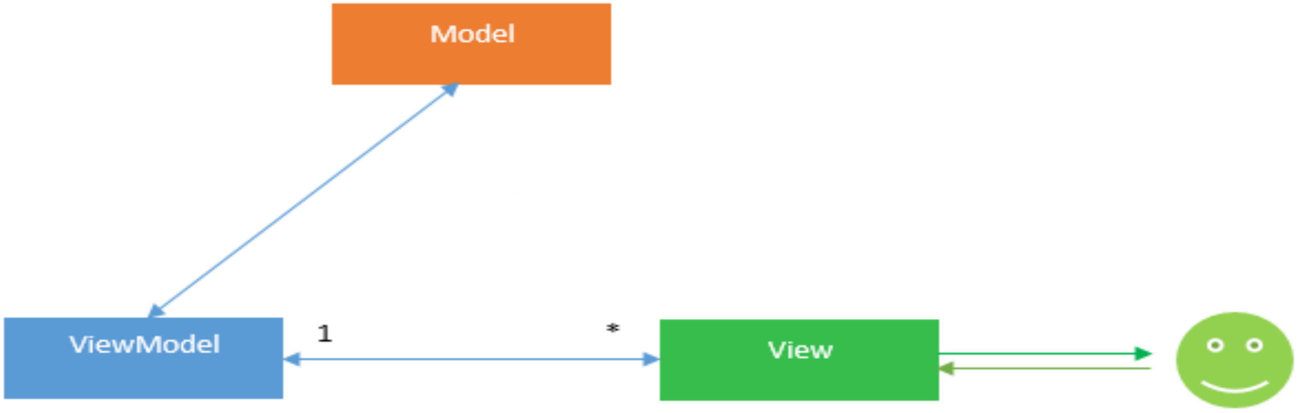
- Interest
  - When we develop a web application or other user interface application, we tend to write a lot of UI logic behind the UI itself: several lines of code to manage interactions specific to the context that we cannot reuse and which is also difficult to test
    - Microsoft has several proprietary languages that use the .NET framework (VB, C#, asp, apsx, etc.). So if each View inherits from the same ViewModel, the code to produce will be less.

MVVM: Model-View-ViewModel

- Communication
  - One of the most powerful concepts that comes with MVVM is that of **bidding** . This is a way to automatically update the view every time the model changes and conversely update the model every time the view changes.
    - Microsoft introduces a new component: the **Binder** which serves as an intermediary between the View and the ViewModel.
    - The view will NEVER have to process data because the ViewModel will be responsible for it. The View will just display them.

MVVM: Model-View-ViewModel

- Components
  - Model and View: have the same role in MVC and MVVM patterns
  - ViewModel: This is where the UI-specific logic is represented. It allows the connection between the model and the view. It can be considered as a special controller that would act as a data converter.



MVVM Pattern

MVC and its variants

- The main difference between MVC and its derivatives is the dependence that each component of the model has on the others and how closely they are interrelated .
- In MVC, the View is at the very top of the architecture with the Controller below it:
  - Views have direct access to models.
  - Exposing the complete model to view may incur a cost in terms of security and performance depending on the complexity of the application.

References

- http://heim.ifi.uio.no/~trygver/2007/MVC_Originals.pdf
- Separation between Human Machine Interface and Functional Core. What technologies? Myth or reality? Romain Huneau, Jérémy Nesmes, Adam Palma, Zhou Ren
- MVC architecture: Model – View – Controller. Software Engineering License 3 - University of Le Havre. Bruno Mermet
- LOG2410 - Software design, François Guibault, 2006 CHAPTER 5 Architectural model. Ecole Polytechnique Montréal.
- Patternmaking. Design Patterns. Bruno Bachelet: http://www.nawuak.net