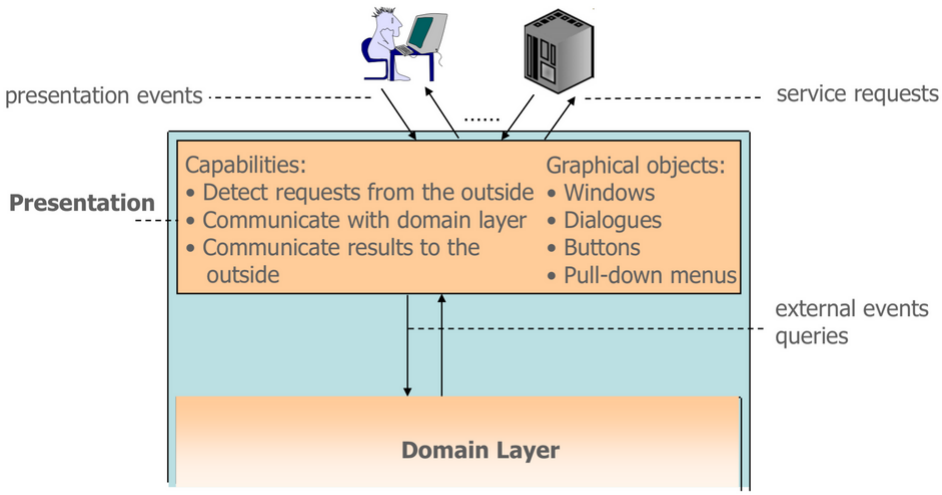
# AS - Teoria 3

## 3.3. Presentation Layer Design

## Índex

# Introduction




- Starting point for the design of the Presentation Layer:
- Responsibilities assigned to the Presentation Layer.
- Technologic characteristics of the input peripherals (keyboard, mouse, ...) and the output peripherals (screen, printer, ...).
    - Functionalities of the GUI libraries.
- The design of the Presentation Layer includes two clearly differentiated tasks:
    - **External Design**: definition of the interaction of the user with the software system.
        - Its purpose consists of designing the (tangible) elements that the user sees, feels and touches when he interacts with the system.
        - Produces the design of a (graphical) user interface (GUI)
    - **Internal Design**: definition of the interaction between the GUI and the Domain Layer.

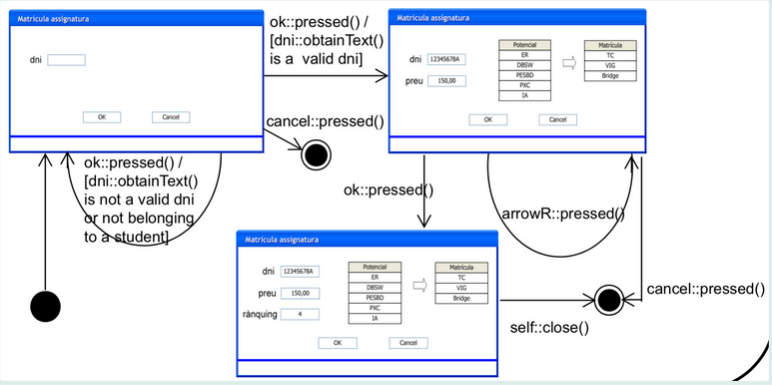
# External Design of the Presentation Layer

- Consists of the definition of:
    - Mechanisms that allow the user to make requests to the system → **Interaction mechanisms**
    - Ways to show the user the results of his requests → **Mechanisms for the presentation of information**

- Examples:
  - Interaction mechanisms: command language, function keys, pointing objects/menus with mouse or touch-sensitive screen, oral commands...
  - Mechanisms for the presentation of information: graphic formats, images, textual, video; presentation in the screen or in printed list; ...
- The team of designers has to include experts in diverse areas:
  - Knowledge of the system's domain → participation of the final user
  - Knowledge of object orientation → programmers, ...
  - Knowledge of sociology, psychology and physiology → psychologist, ...
  - Knowledge of ways to present the information → graphic designers, ...
- Design process based on prototyping.
- Three golden rules behind the **principles of the user interface** design:
  - The user is in control
    - Interaction modes, flexibility, cancelations, rectifications, customization, transparency, ...
  - Minimize memorization:
    - Reduce short term memory, default options, shortcuts, metaphors, hierarchies, ...
  - Maintain a consistent interface
    - Inform about the context, consistency between windows, consistency between product families, respect conventions, ...
- We have to bear in mind a lot of factors, for example:
- Human factors:
  - Green background, red letters: very nice, but 7% of male population is daltonic!
- Social or cultural factors:
  - 7/6/2005: June the 7th or July the 6th?

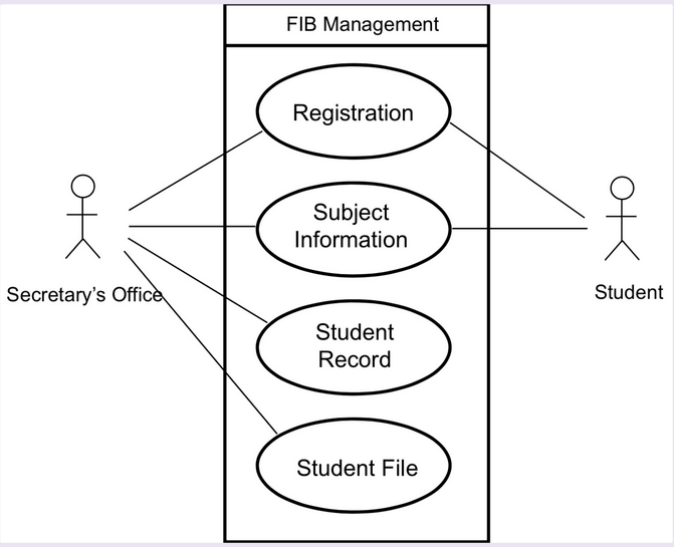
# Navigational Maps

- Working with events complicates the quick comprehension of the progression of the user interface
  - Which are the relevant presentation events?
  - Under which conditions do they provoke changes?
- Navigational maps: they represent the navigational paths between screens
  - Use case scope and system scope
  - Several formalisms and detail levels are possible
- The state diagrams are a good option to visualize these paths:
  - They’re a formalism we already know
  - We can represent the relevant states the interface goes through
  - We can identify the relevant presentation events
  - We can establish the conditions that influence the result of each presentation event
- Use Case Navigational Maps: High level description

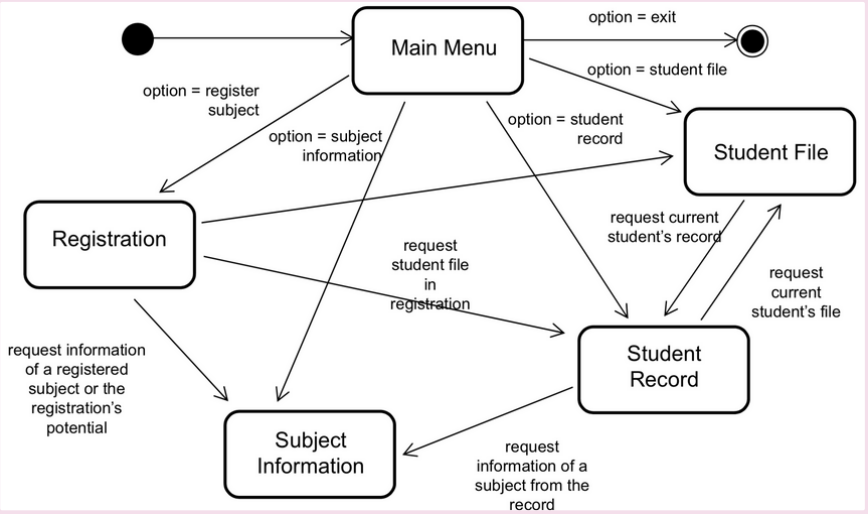


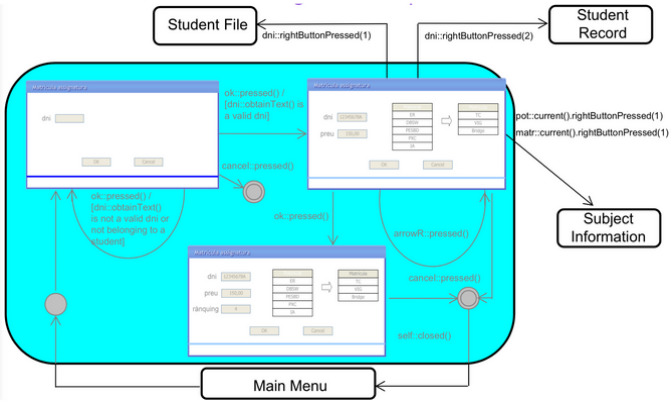
- System Navigational Map
  - We need a general perspective of the system
    - Generally, for usability purposes, we define many valid transitions between diverse screens from different use cases
      - new elements in the interface (i.e., facilitating the use of the right button of the mouse)
    - Use cases show the relation between the system's functionalities, but not the details of the transitions from one screen to another
  - The state diagrams of the individual use cases have to be consistent with the system's state diagram

- Use Case Diagram of the FIB's system



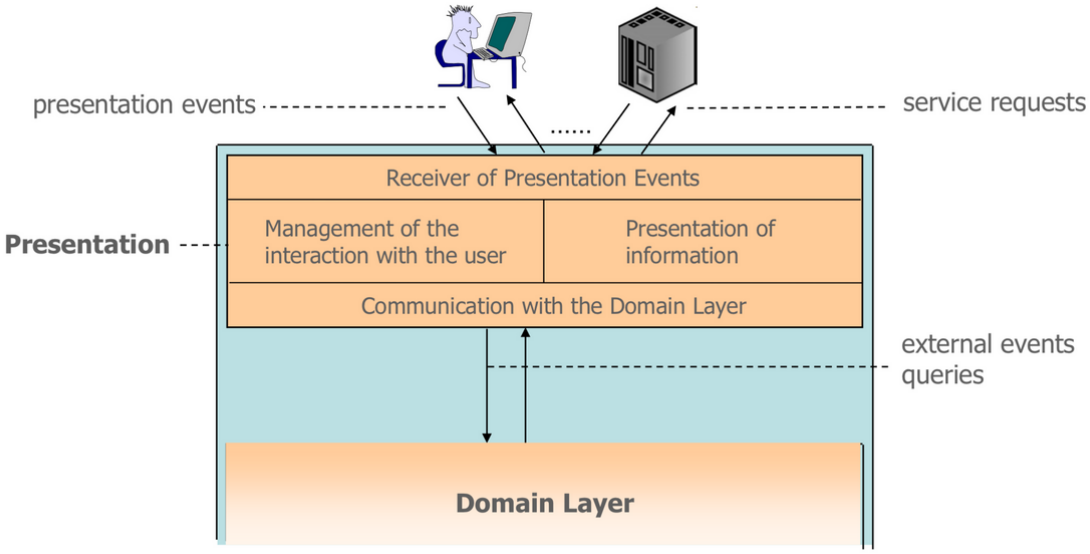

- System Navigational Maps: High level description






# Internal Design of the Presentation Layer

Logic structure of the Presentation Layer:




- Receiver of the Presentation Events
  - User interfaces based on events.
  - How does the Presentation Layer receive these events?
  - Management of the communication between the software system and the operating system.
- Management of the Interaction with the User
  - Controls the communication of presentation events of the receiver.
  - Processes these events and identifies external events.
- Communication with the Domain Layer
  - Sends the external events that have to be processed.
  - Receives the answers to these events.

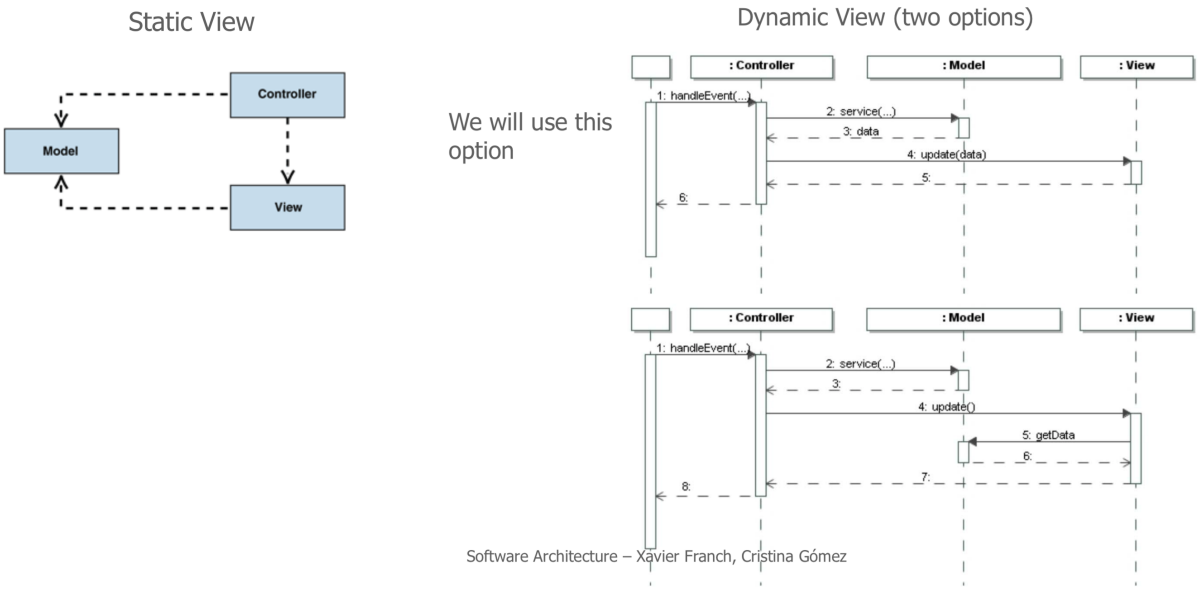
- Presentation of the Information
  - Presents the data (own or received from the Domain Layer) in the formats determined by the external design.
- Receiver of the Presentation Events
  - The elements of the UI are modeled as objects

- Management of the Interaction with the User
  - Model-View-**Controller** Pattern.
- Communication with the Domain Layer
  - **Model**-View-**Controller** Pattern.
- Presentation of the Information
  - Model-**View**-ControllerPattern.

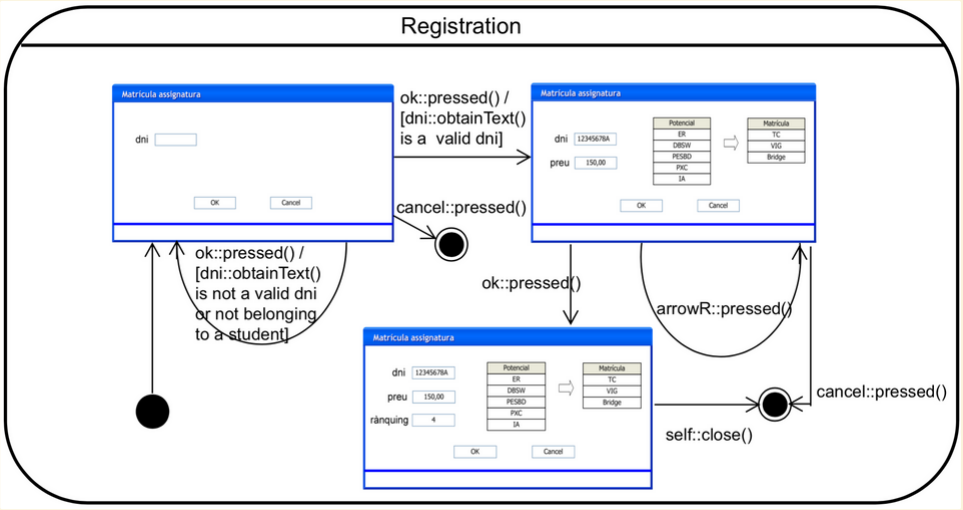
## Model-View-Controller Pattern

- Context:
  - Interactive software systems with flexible user interfaces
- Problem
  - How to modularize user interface functionality of a software system so that it can easily modified?
- Forces to balance
  - User interface logic tends to change more frequently than business logic.
  - It is necessary to display the same data in different ways. If the user changes data in one view, the system must update data in the other views (only for active model variant).
  - User interface code tends to be more device-dependent than business logic.

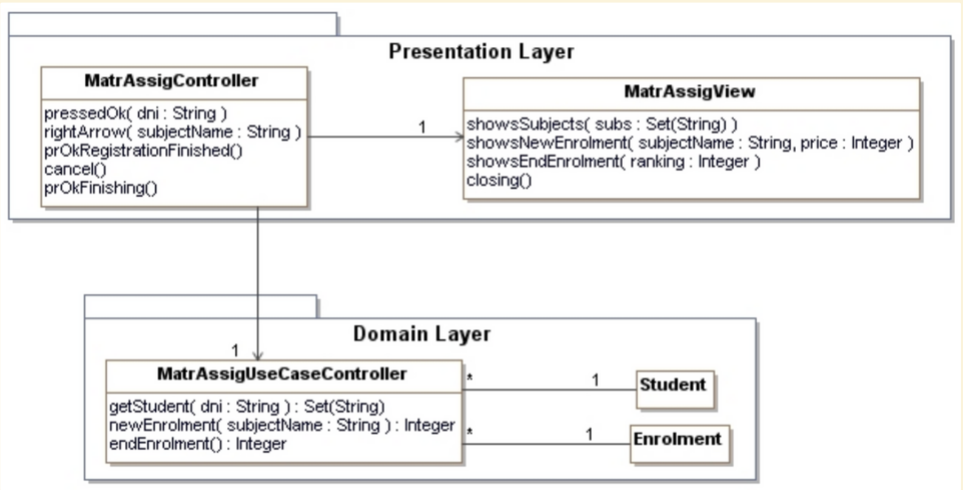
- Solution
  - Separate the modeling of the domain, the presentation and the actions based on user input into three separate classes:
    - Model: manages the behavior and data of the application domain, responds the requests for information about its state (usually from the view) and responds to instructions to change state (usually form the controller).
    - View: manages the display of information
    - Controller: interprets the mouse and keyboard inputs from the user, informing the model and/or the view to change as appropriate
  - The Presentation Layer contains the Views and the Controllers.
  - The Model represents the Domain and the Data Layer.

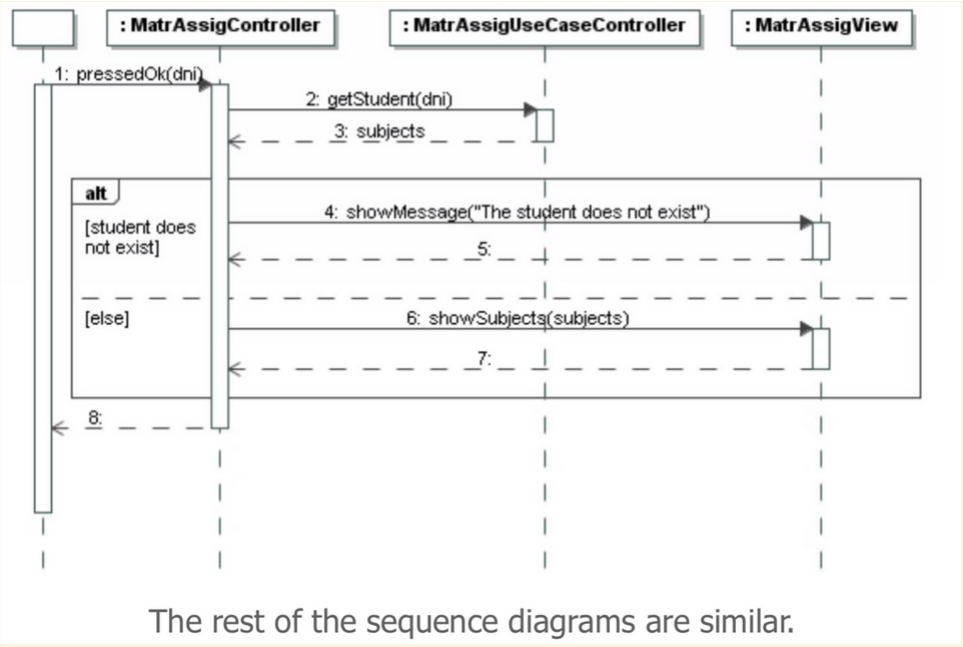


- Example




| **Presentation Layer** Controller Operations | **Domain Layer** Model Operations | **Presentation Layer** View Operations |
|---|---|---|
| pressedOk(dni) | getStudent(dni):Set(string) | showSubjects(subs:Set(String)) |
| rightArrow(subjectName) | newEnrolment(subjectName): Integer | showNewEnrolment(subjectName, price) |
| prOkRegistrationFinished() | endEnrolment():Integer | showEndEnrolment(ranking) |
| cancel() | - | closing() |
| prOkFinishing() | - | closing() |






The rest of the sequence diagrams are similar.

- **Advantages**:
  - It may exists several views related to a model.
  - At the execution time, it may be several open views.
  - Views and controllers are easily reusable.
  - High portability of the Presentation layer.
  - Low coupling between Domain and Presentation layers.
- **Drawbacks**:
  - High coupling between views and controllers.
- **Alternatives**:
  - Model-View-Controller Pattern (Active Model variant)
  - Presentation-Abstraction-Control Pattern.
  - Document-View Pattern.

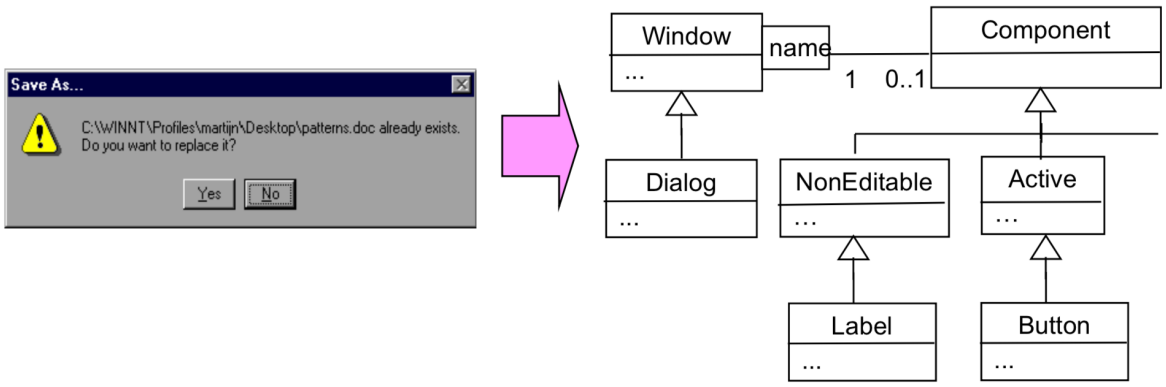
# Presentation Layer Patterns

- They allow tackling the design of the Presentation Layer with the classic context problem-solution approach in mind.
  - External design: impact on the user interface
  - Internal design: effect on the Presentation Layer objects
- From the point of view of the external design, the patterns are related with the principles of interface design
- From the point of view of the internal design, the patterns follow the MVC pattern and the principles of the OO architectural pattern
- Several types of patterns:
  - Generic patterns
  - Specific patterns for web applications
  - Specific patterns for mobile technology applications
  - ...

- Some categories.
  - **Mode**: control / visualization of the current working mode
    - Examples: cursor mode, automatic change of mode
  - **Layout**: organization of the information in the window
    - Examples: grid presentation, navigable spaces
  - **Selection**: insertion of the information in the system
    - Examples: contextual menu, continuous filtering, non-ambiguous format
  - **Guide**: actions that help the user
    - Examples: protection, warnings, progress, undo
  - **Navigation**: transition between windows
    - Examples: assisstant, persistent options, two-level information, lists

# Protection Pattern

- Context:
  - There are several actions that have important (and irreversibe) effects on the system
  - There are several actions that, once completed, are very costly to undo
- Problem
  - The user can accidentally select an option that has irreversible effects or that is very costly to undo
- Solution
  - Add an extra level of protection in the function
    - the mistake has to be done twice instead of once
  - The user has to confirm explicitly the chosen option
    - default option: not executing the function
  - Consider the possibility of allowing to configure (deactivate) this behavior



# Presentation Layer Patterns:
## Protection Pattern