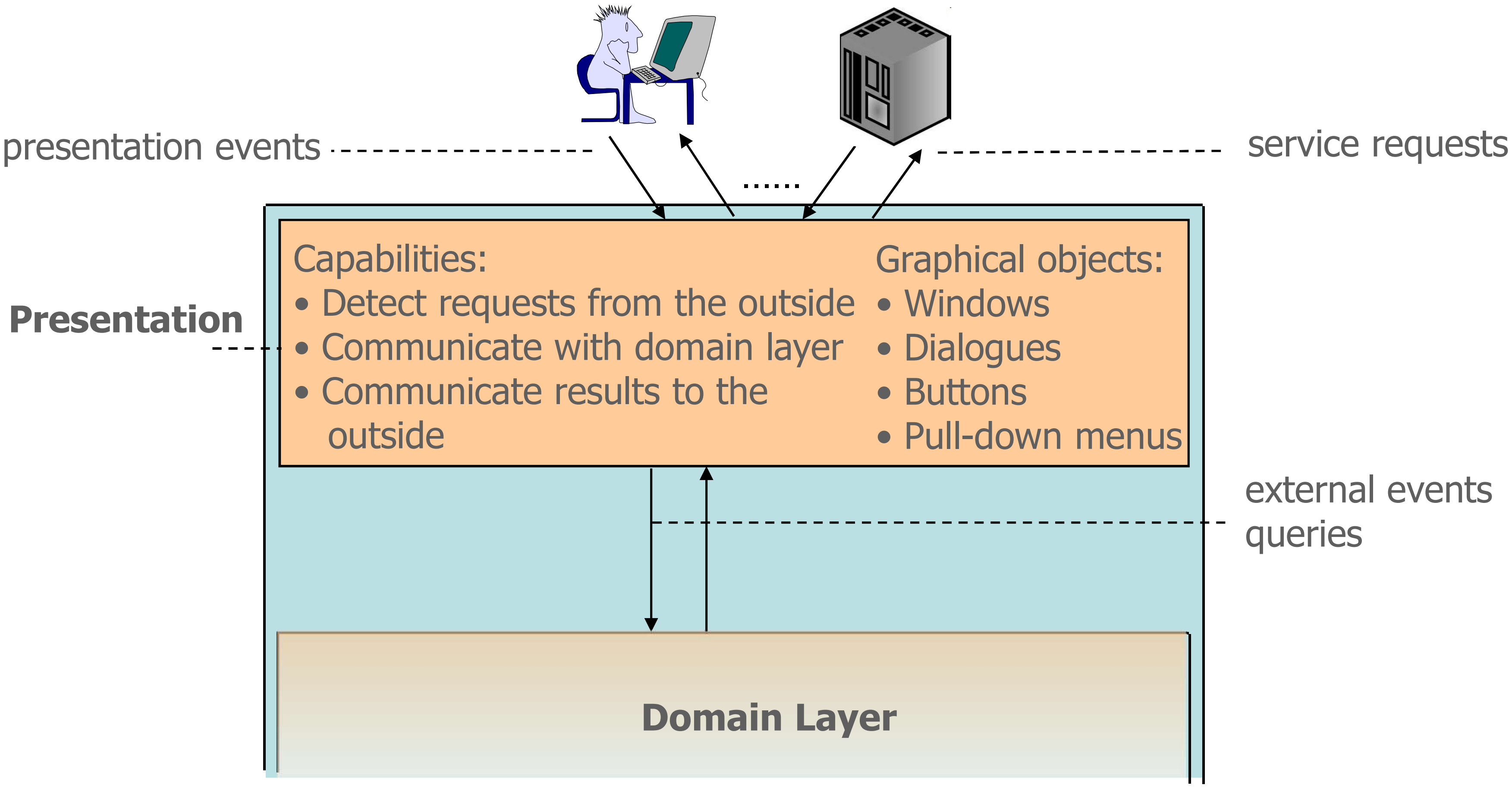
Presentation Layer Design

Presentation Layer Design

- Introduction
- External Design of the Presentation Layer
  - Navigational Maps
- Internal Design of the Presentation Layer
  - Model-View-Controller Pattern
- Presentation Layer Patterns
- References
- Annex

Introduction



presentation events

service requests

......

Presentation

Capabilities:
- Detect requests from the outside
- Communicate with domain layer
- Communicate results to the outside

Graphical objects:
- Windows
- Dialogues
- Buttons
- Pull-down menus

external events
queries

Domain Layer

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

External Design of the Presentation Layer

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:

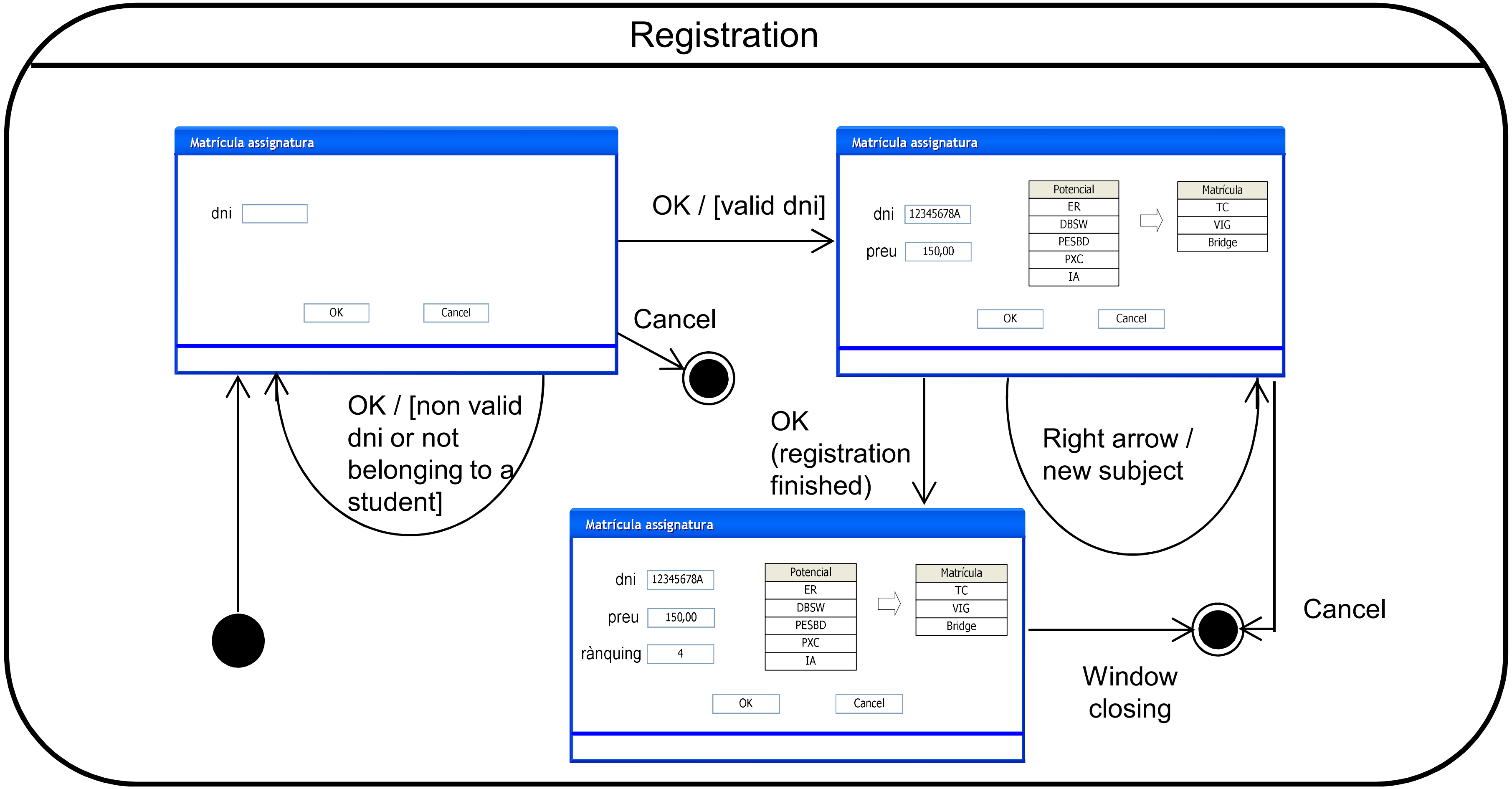
- Human factors:
  - Green background, red letters: very nice, but 7% of male population is daltonic!
- Social or cultural factors:
  - 7/6/2005: June the 7th or July the 6th?

External Design of the Presentation Layer:
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

External Design of the Presentation Layer:
Navigational Maps

- Use Case Navigational Maps: High level description



Registration

Matrícula assignatura

dni

OK / [valid dni]

Cancel

OK / [non valid dni or not belonging to a student]

Matrícula assignatura

dni 12345678A

preu 150,00

Potencial
ER
DBSW
PESBD
PXC
IA

Matrícula
TC
VIG
Bridge

OK (registration finished)

Right arrow / new subject

Matrícula assignatura

dni 12345678A

preu 150,00

rànquing 4

Potencial
ER
DBSW
PESBD
PXC
IA

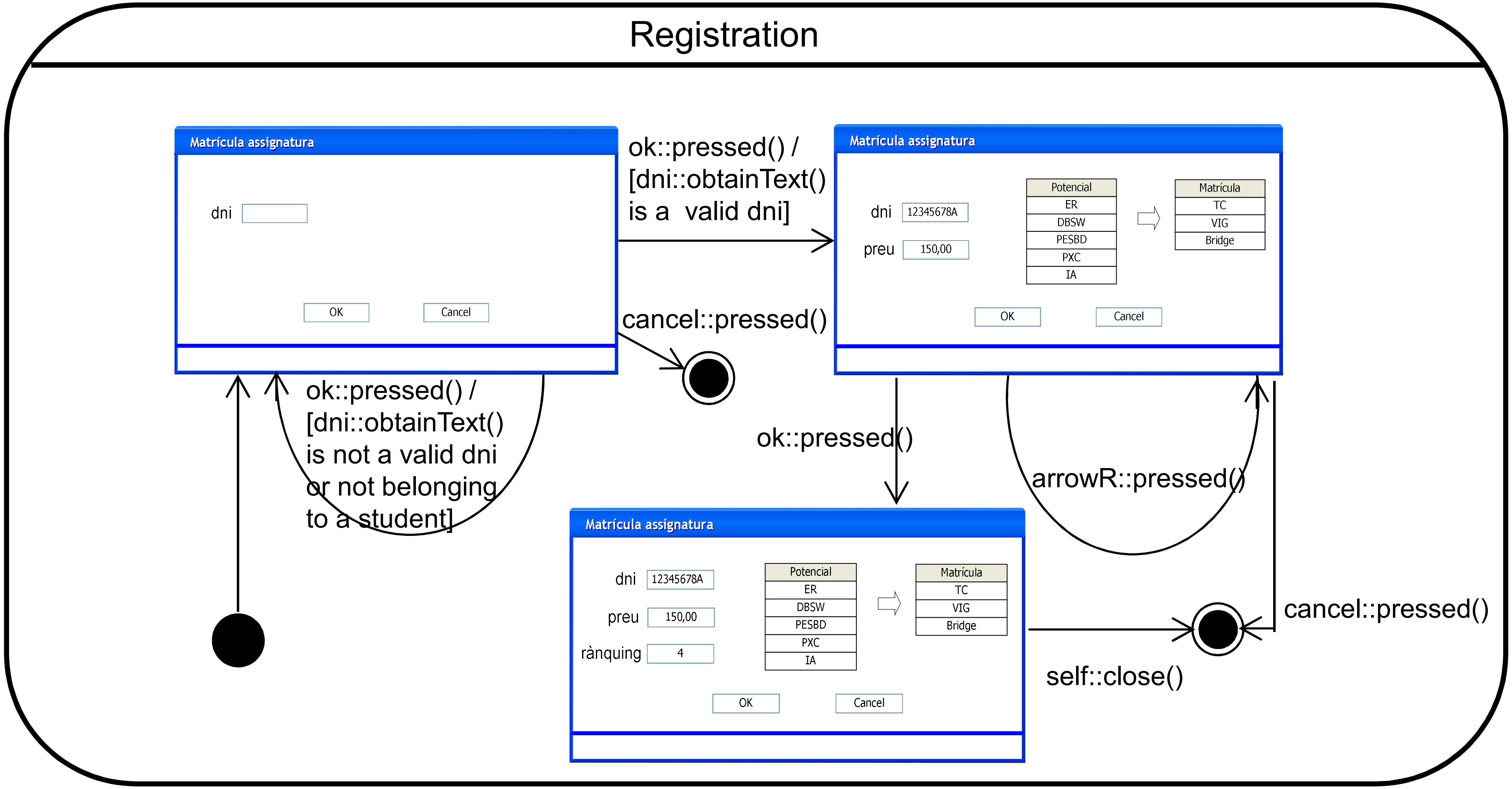
Matrícula
TC
VIG
Bridge

Window closing

Cancel

External Design of the Presentation Layer:
Navigational Maps

- Use Case Navigational Maps: Detailed description



Registration

Matrícula assignatura

dni

ok::pressed() / [dni::obtainText() is a valid dni]

cancel::pressed()

ok::pressed() / [dni::obtainText() is not a valid dni or not belonging to a student]

Matrícula assignatura

dni 12345678A

preu 150,00

Potencial
ER
DBSW
PESBD
PXC
IA

Matrícula
TC
VIG
Bridge

ok::pressed()

arrowR::pressed()

Matrícula assignatura

dni 12345678A

preu 150,00

rànquing 4

Potencial
ER
DBSW
PESBD
PXC
IA

Matrícula
TC
VIG
Bridge

self::close()

cancel::pressed()

External Design of the Presentation Layer:
Navigational Maps

System Navigational Map

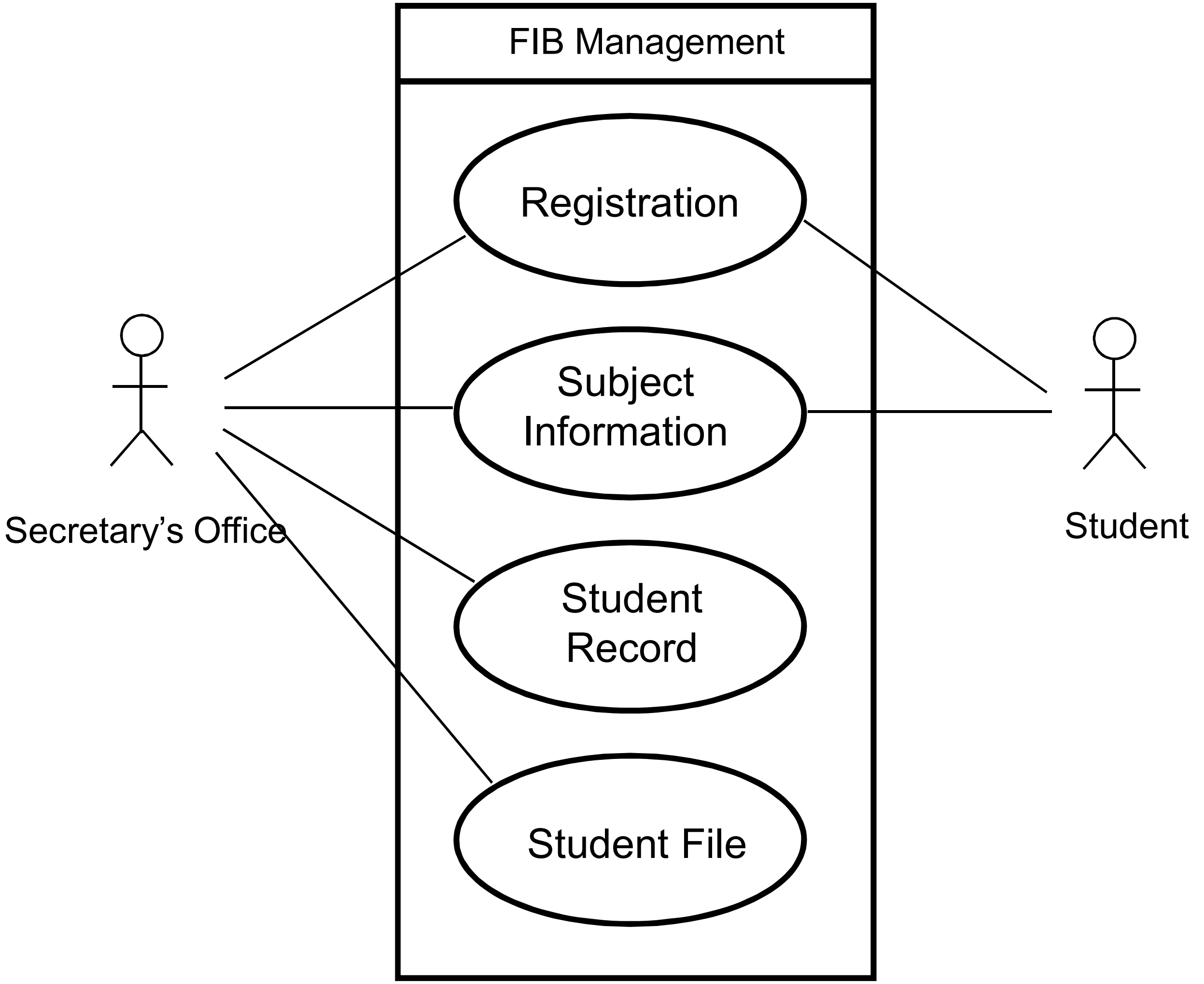
- We need a general perspective of the system
  - Generally, for usability purposes, we define many valid transitions between diverse screens from different use cases
    - ✓ new elements in the interface (i.e., facilitating the use of the right button of the mouse)
  - Use cases show the relation between 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

External Design of the Presentation Layer:

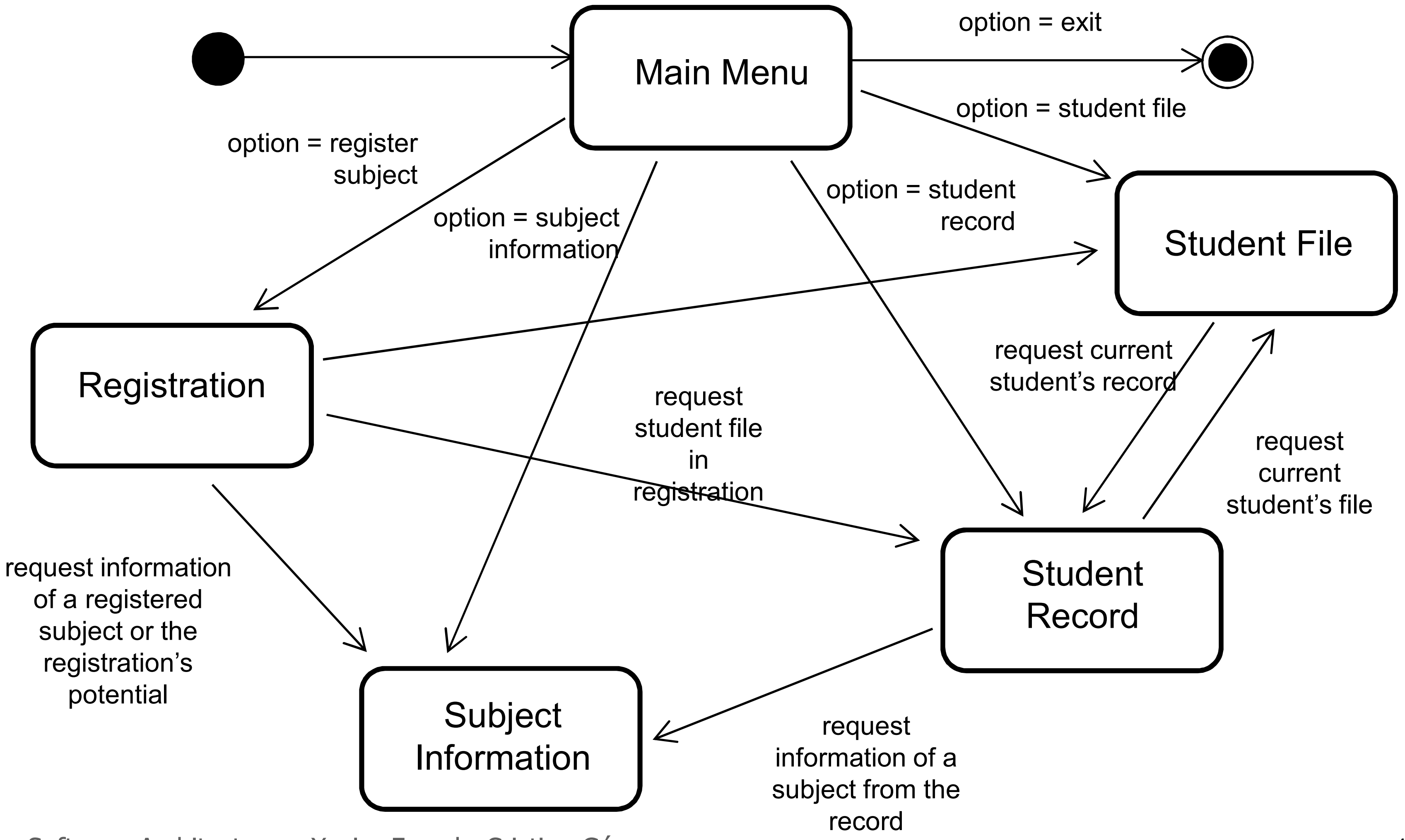
Navigational Maps

- Use Case Diagram of the FIB's system



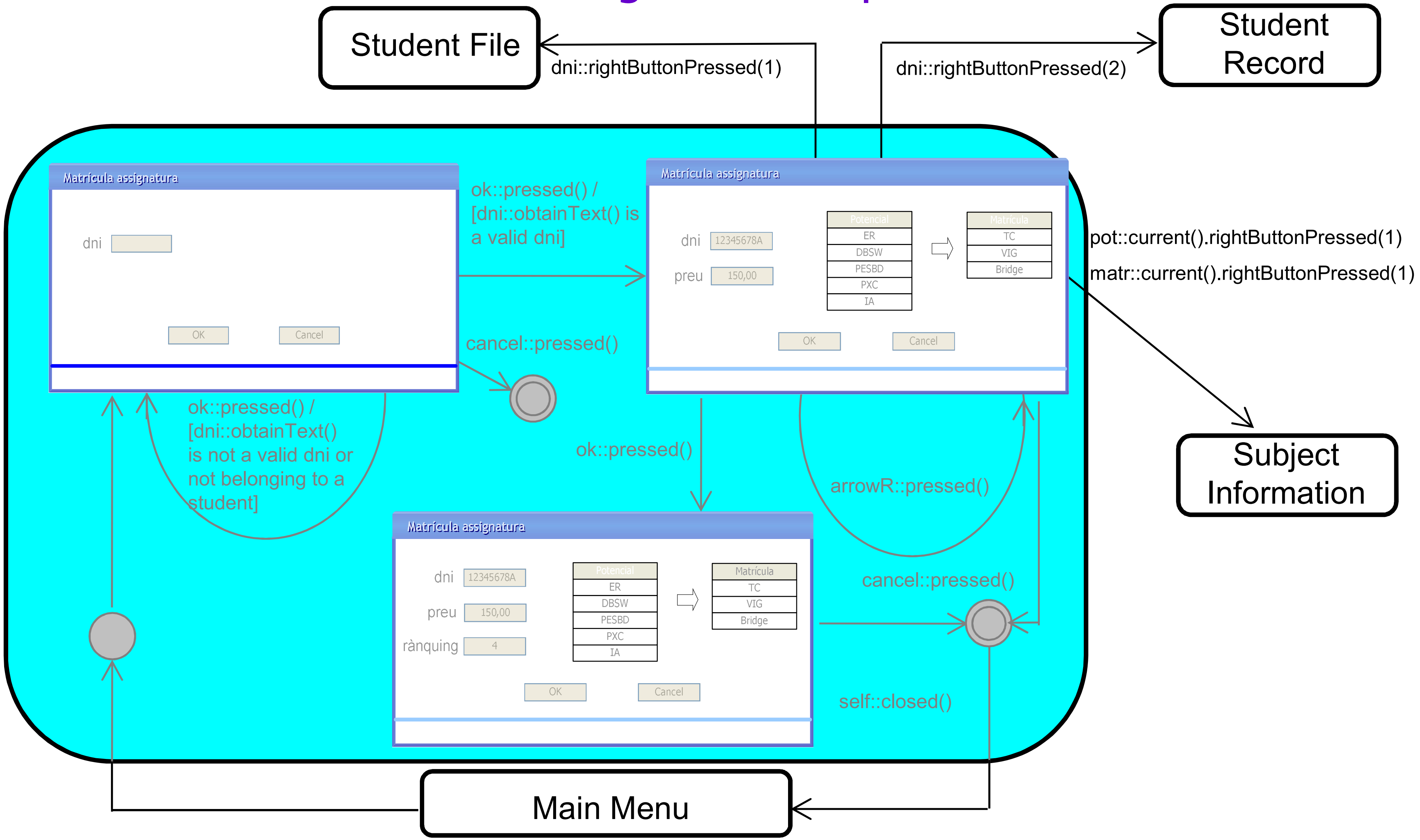
External Design of the Presentation Layer:
Navigational Maps

- System Navigational Maps: High level description



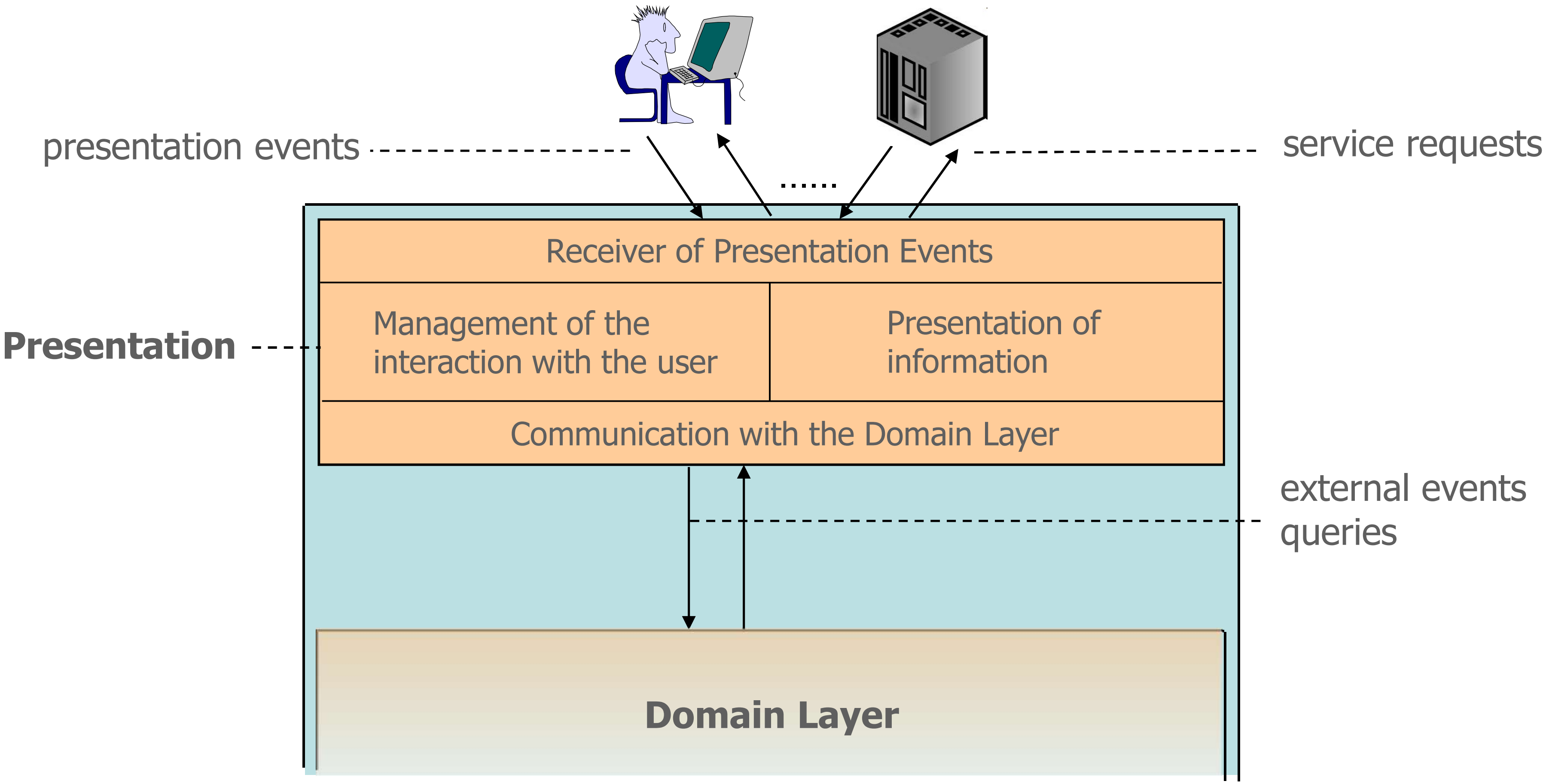
Main Menu
option = exit
option = student file
option = register subject
option = subject information
option = student record
Student File
Registration
request current student's record
request student file in registration
request current student's file
Student Record
request information of a registered subject or the registration's potential
Subject Information
request information of a subject from the record

External Design of the Presentation Layer:
Navigational Maps



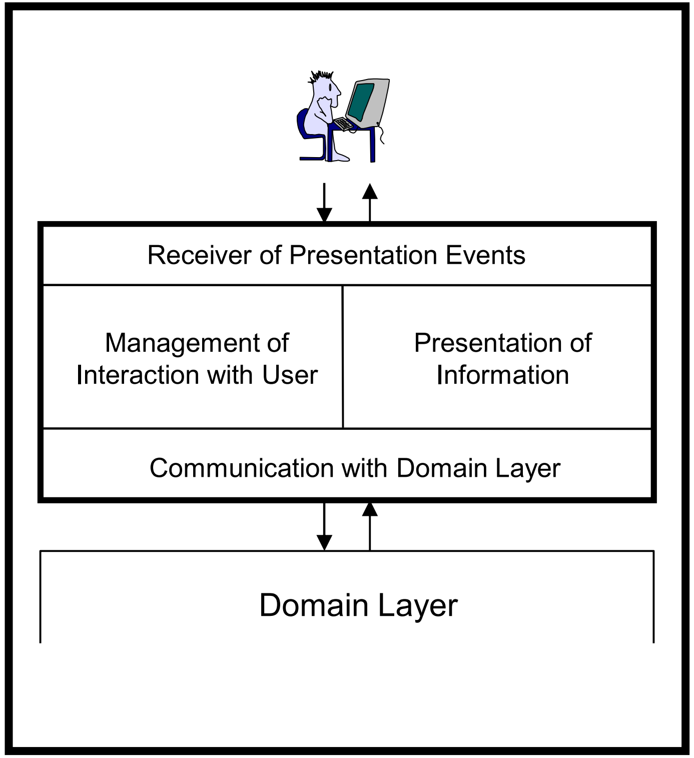
Internal Design of the Presentation Layer

Logic structure of the Presentation Layer:


presentation events
service requests
......
Receiver of Presentation Events
Presentation
Management of the interaction with the user
Presentation of information
Communication with the Domain Layer
external events
queries
Domain Layer

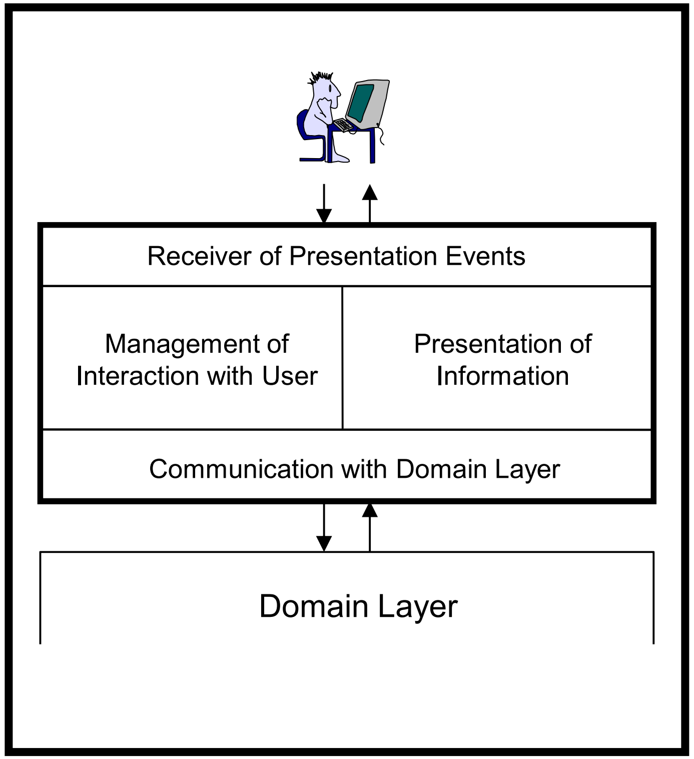
Internal Design of the Presentation Layer

- 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 Presentation Events
Management of Interaction with User
Presentation of Information
Communication with Domain Layer
Domain Layer

Internal Design of the Presentation Layer

- Receiver of the Presentation Events
  - The elements of the UI are modeled as objects (see Annex 1)
- 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**-Controller Pattern.


Receiver of Presentation Events
Management of Interaction with User
Presentation of Information
Communication with Domain Layer
Domain Layer

Internal Design of the Presentation Layer:
Model-View-Controller Pattern

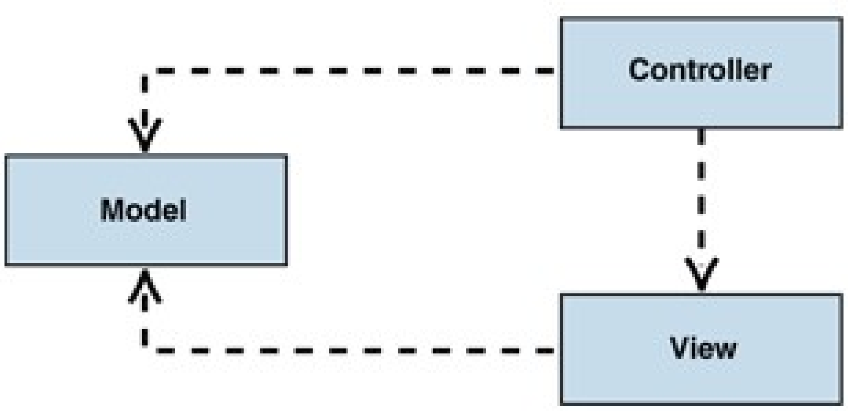
- Context
  - Interactive software systems with flexible user interfaces
- Problem
  - How to modularize user interface functionality of a software system so that it can easily modified?
- Forces to balance
  - User interface logic tends to change more frequently than business logic.
  - It is necessary 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.

Internal Design of the Presentation Layer:
Model-View-Controller Pattern

- 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.

Internal Design of the Presentation Layer:
Model-View-Controller Pattern

Static View



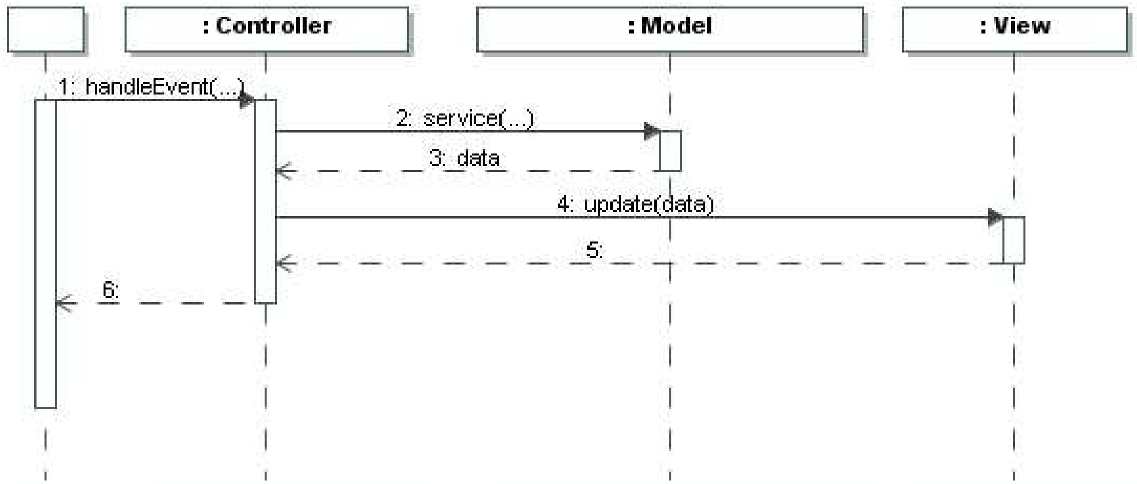
The *active model* variant of the MVC may be found in the Annex 2

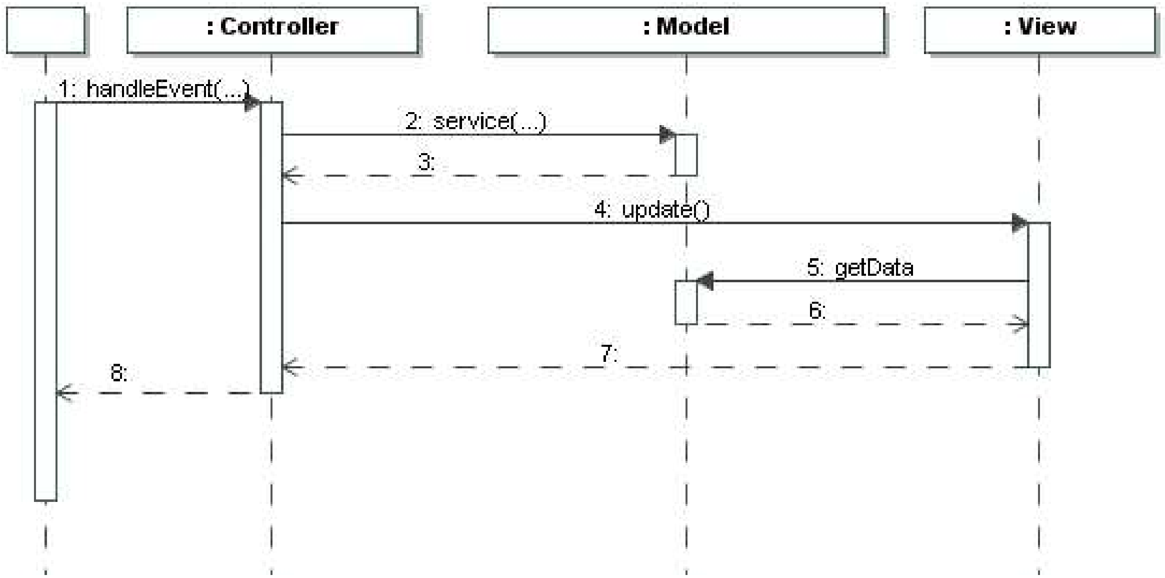
Internal Design of the Presentation Layer:

Model-View-Controller Pattern

Dynamic View (two options)

We will use this option

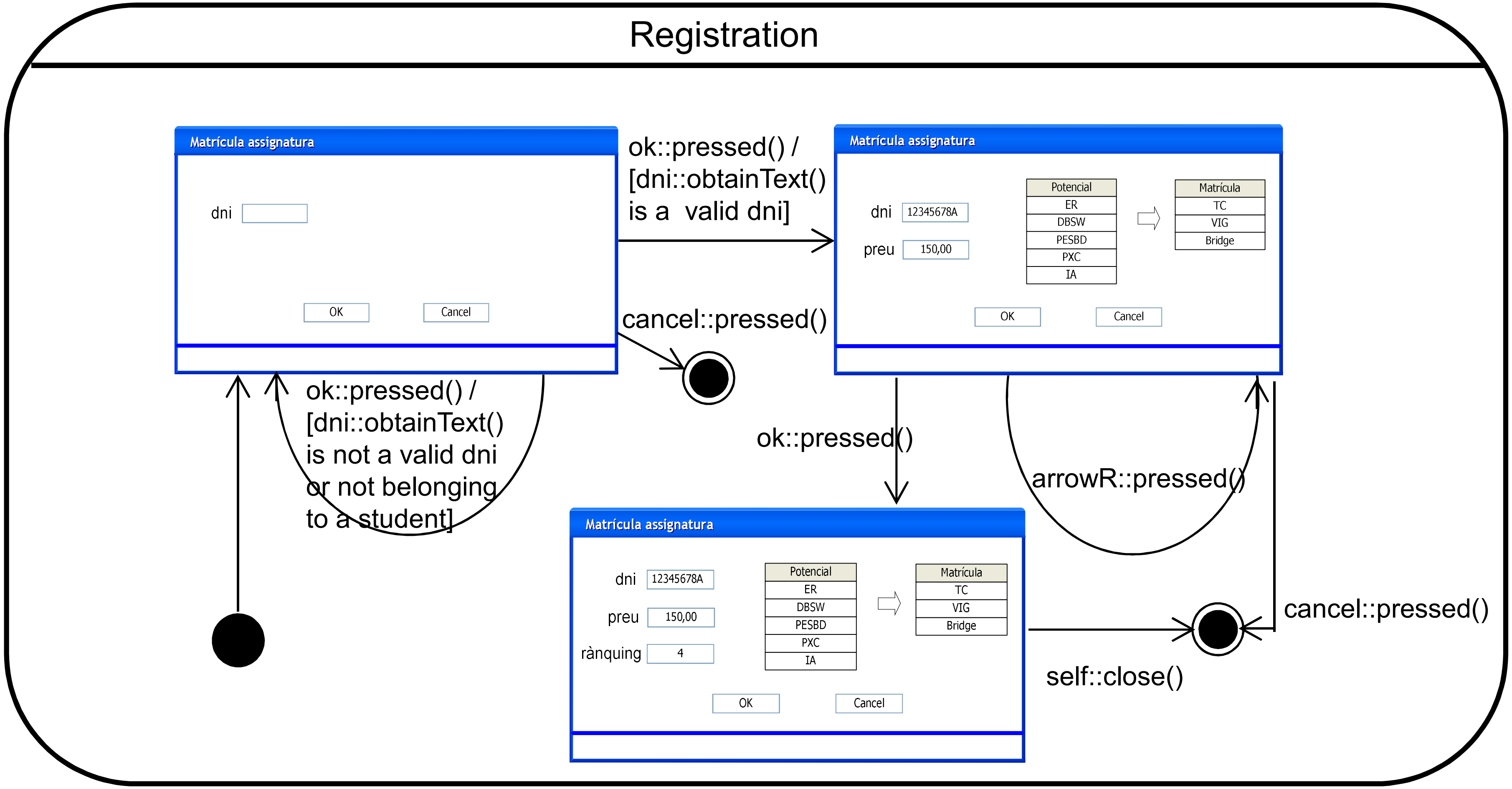




Internal Design of the Presentation Layer:

Model-View-Controller Pattern

- Example



Registration

ok::pressed() / [dni::obtainText() is a valid dni]

cancel::pressed()

ok::pressed() / [dni::obtainText() is not a valid dni or not belonging to a student]

ok::pressed()

arrowR::pressed()

cancel::pressed()

self::close()

dni 12345678A

preu 150,00

rànquing 4

Internal Design of the Presentation Layer:

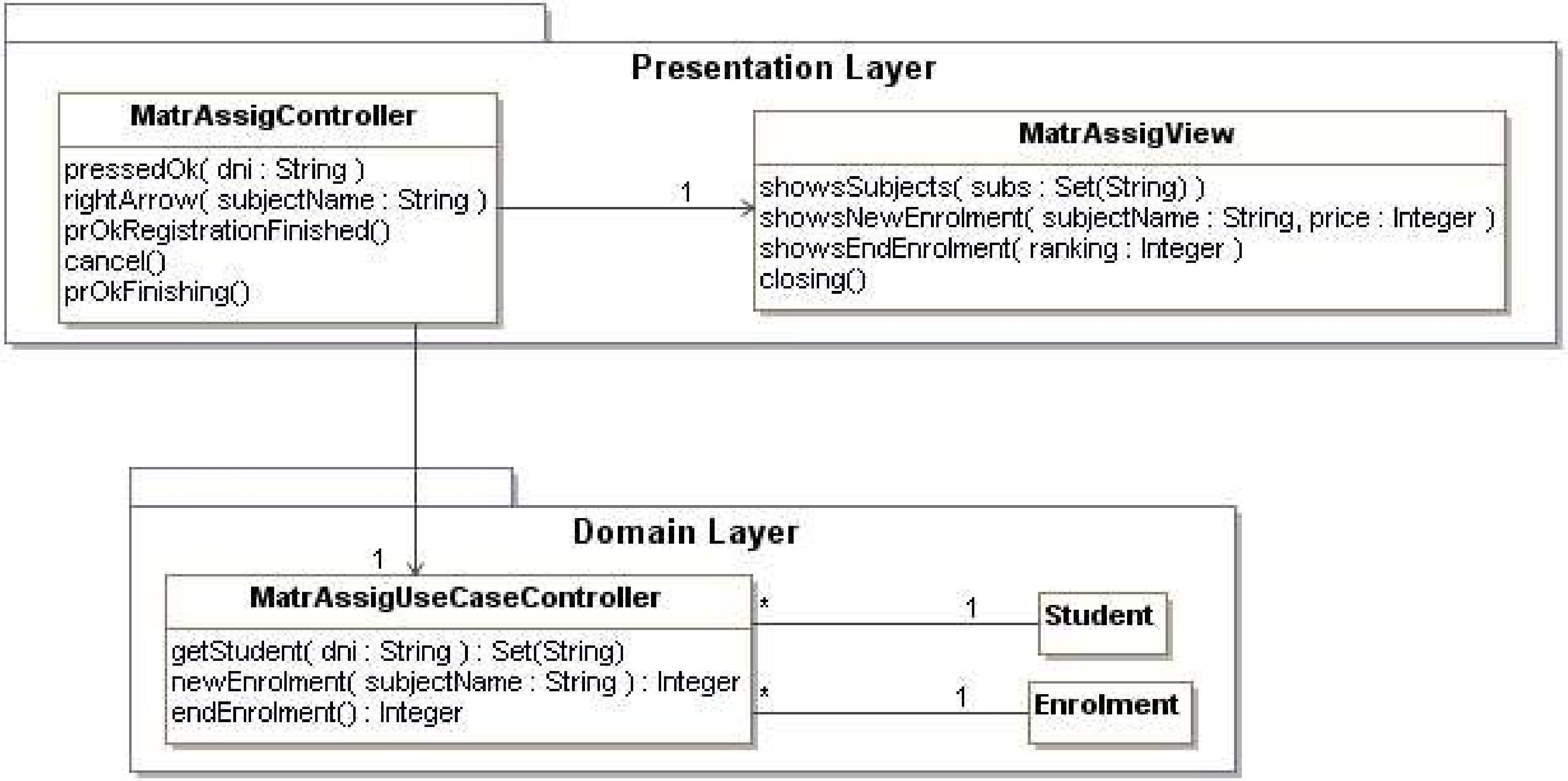
Model-View-Controller Pattern

- 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()

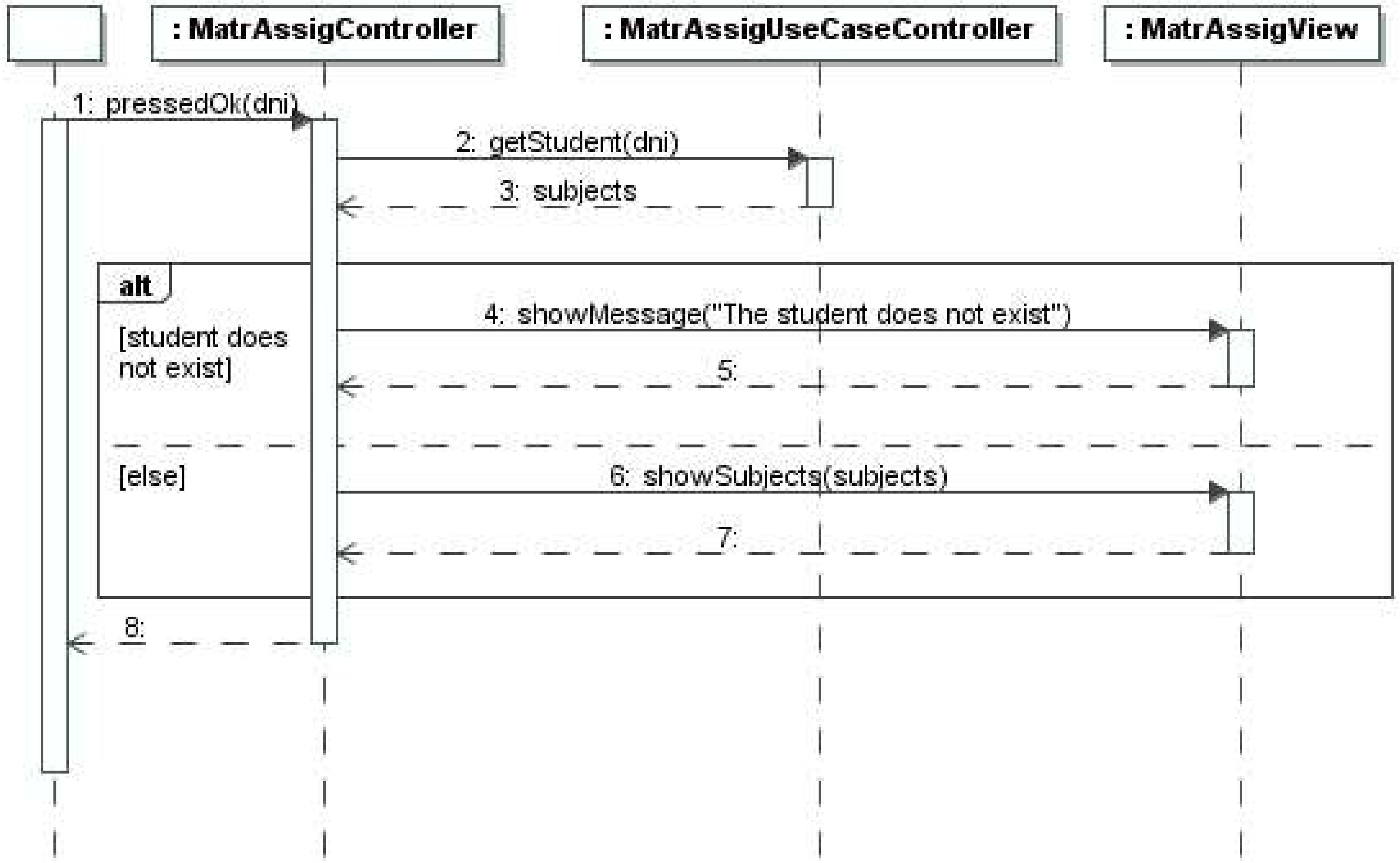
Internal Design of the Presentation Layer:
Model-View-Controller Pattern

- Example


Presentation Layer
MatrAssigController
pressedOk(dni : String)
rightArrow(subjectName : String)
prOkRegistrationFinished()
cancel()
prOkFinishing()
1
MatrAssigView
showsSubjects(subs : Set(String))
showsNewEnrolment(subjectName : String, price : Integer)
showsEndEnrolment(ranking : Integer)
closing()
Domain Layer
1
MatrAssigUseCaseController
getStudent(dni : String) : Set(String)
newEnrolment(subjectName : String) : Integer
endEnrolment() : Integer
*
1
Student
*
1
Enrolment

Internal Design of the Presentation Layer:
Model-View-Controller Pattern

- Example



The rest of the sequence diagrams are similar.

Internal Design of the Presentation Layer:
Model-View-Controller Pattern

- **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
  - …

Presentation Layer Patterns

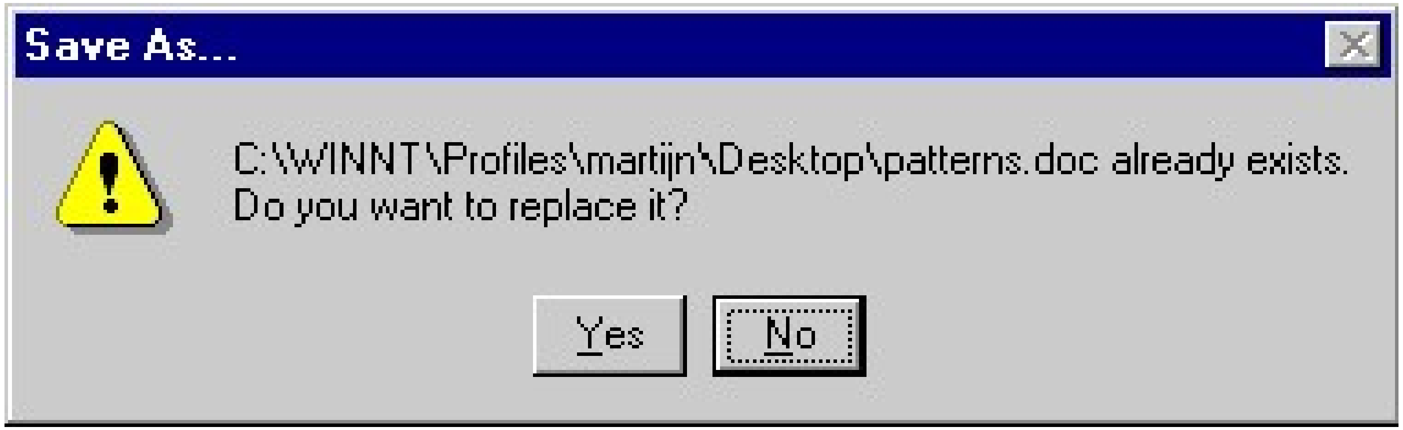
Some categories:

- 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

Presentation Layer Patterns:
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

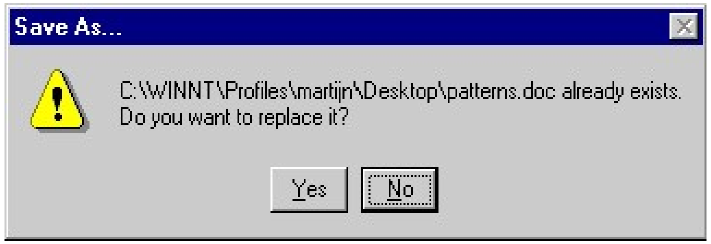
Presentation Layer Patterns:
Protection Pattern



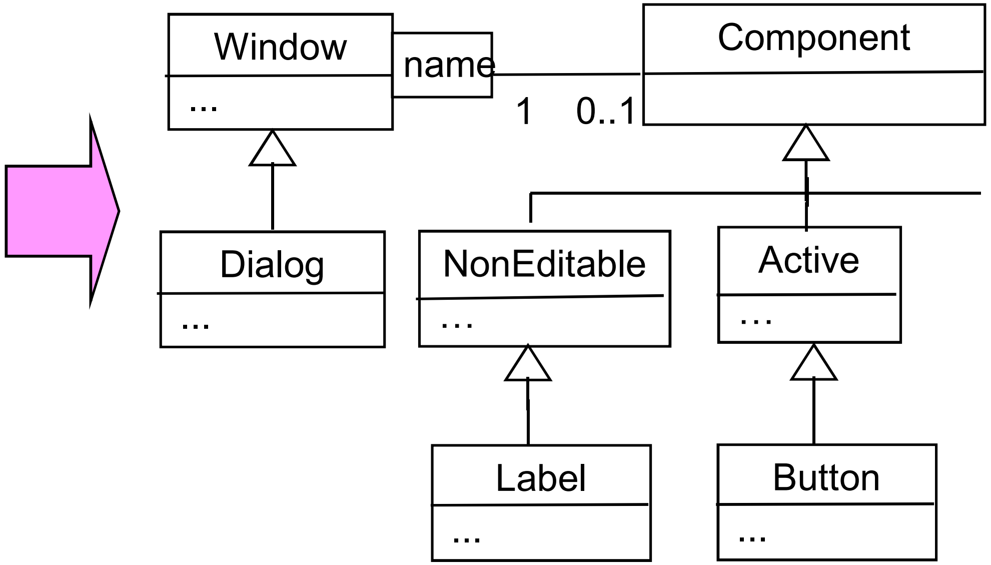
Save As...

C:\WINNT\Profiles\martijn\Desktop\patterns.doc already exists.
Do you want to replace it?

Presentation Layer Patterns:
Protection Pattern

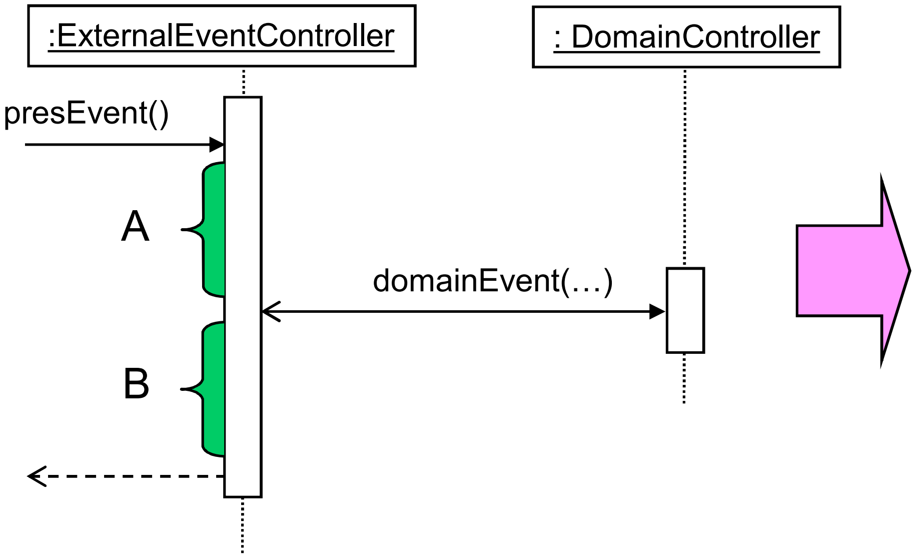


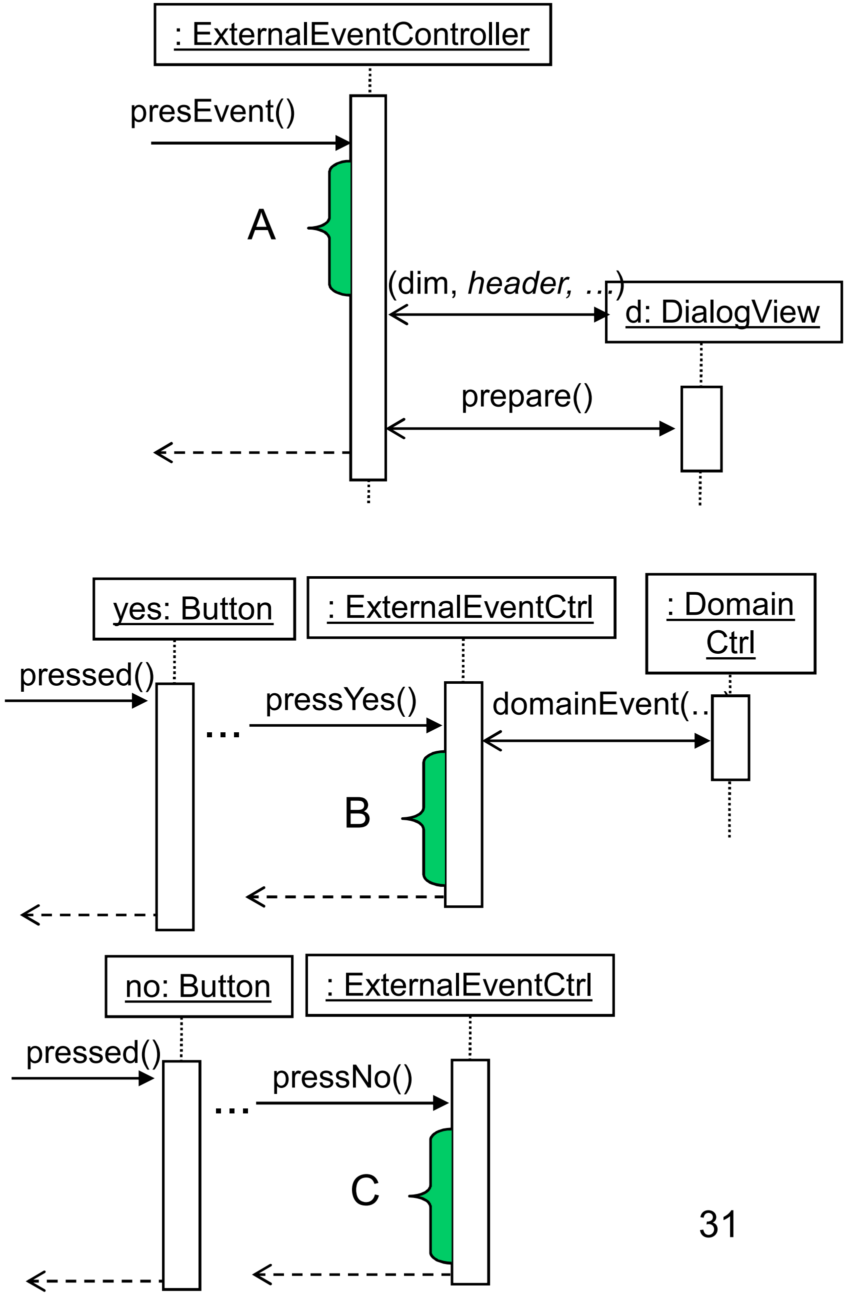
C:\WINNT\Profiles\martijn\Desktop\patterns.doc already exists.
Do you want to replace it?



Presentation Layer Patterns:

Protection Pattern





References

- *Patterns of Enterprise Application Architecture*
  M. Fowler
  Addison-Wesley, 2003

- *Pattern-oriented Software Architecture. A System of patterns*
  F. Buschmann, R. Meunier, H. Rohnert, P.Sommerlad, M. Stal
  John Wiley & Sons, 1996.

- *http://msdn.microsoft.com/en-us/library/ff649643.aspx*

- *Designing Object-Oriented User Interfaces*
  D. Collins
  Benjamin/Cummings Publishing Company, 1995. (chap. 1, 5, 6, 11)

- *Software Engineering. A Practitioner's Approach*
  R.S. Pressman
  McGraw-Hill, 2005 (Seventh edition), chap. 11

- http://www.welie.com/patterns/index.php

Annex 1:
Elements of the User Interface

- Description of the user interface elements:
  - The elements of the user interface are modeled as objects
    - ✓ attributes: mainly physical characteristics (coordinates, colours, etc.)
    - ✓ operations: belonging to each element (button pressed, option selected, close window, etc.)

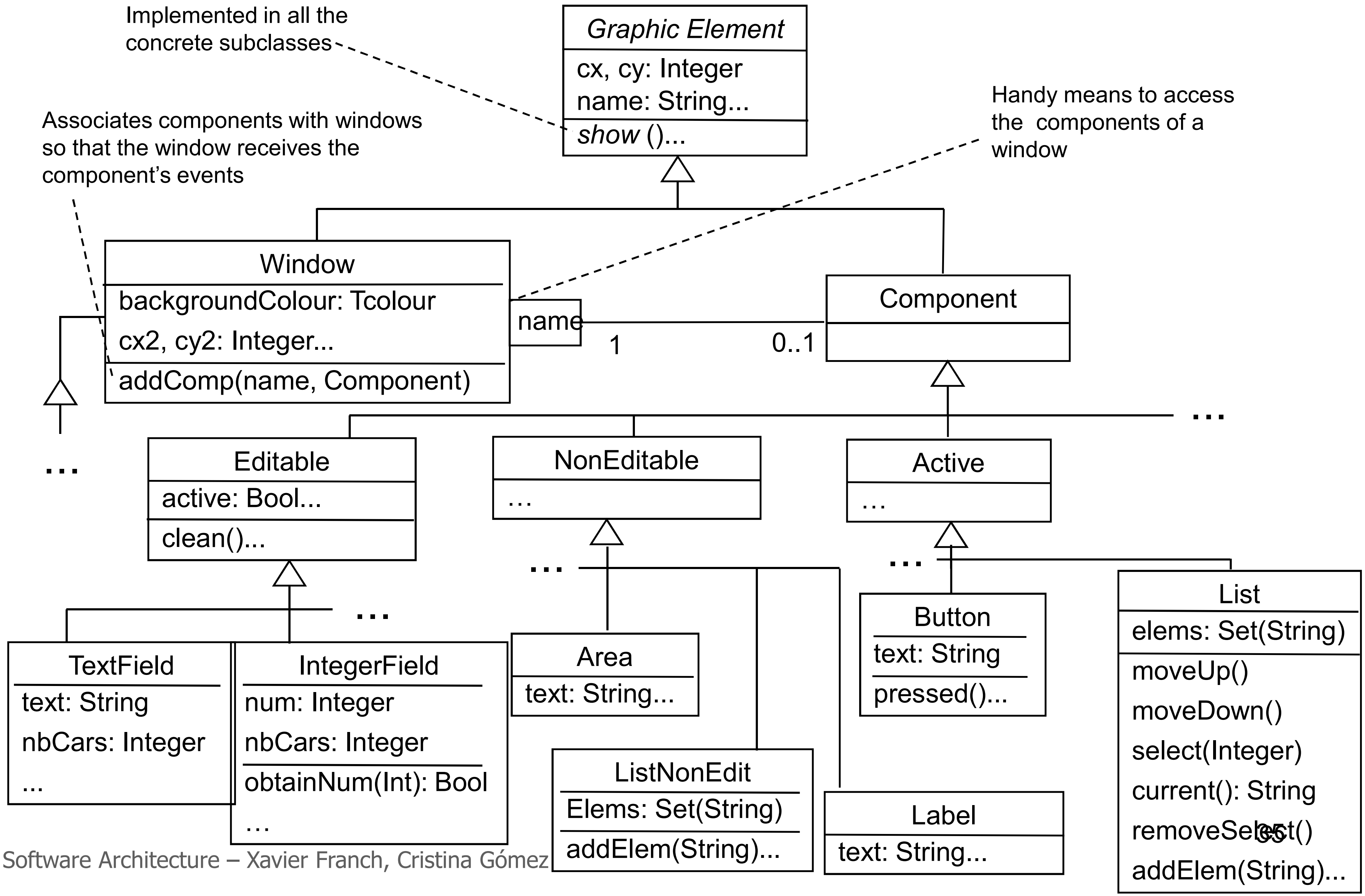
- There are diverse graphic libraries that make things easier when building user interfaces:
  - JFC/Swing, Motif, PowerBuilder...
  - When using these libraries we won't need to work with the level of detail that we'll see in the following slides

Annex 1:
Elements of the User Interface

Structural aspect (simplification / adaptation of the JFC/Swing):

- High level containers: windows
  - Can be of different types: with frame, of dialog, ...
  - Typical operations: close, minimize, scroll, etc.
- Components: associated to the windows
  - Active: generate presentation events
    - ✓ buttons, drop-down menus, etc.
  - Editables: allow the user inserting data
    - ✓ text fields, numeric fields, tables, etc.
  - Non editables: show information to the user
    - ✓ message areas, labels, etc.
  - ...

Annex 1:
Elements of the User Interface



Implemented in all the concrete subclasses

Associates components with windows so that the window receives the component's events

Handy means to access the components of a window

Graphic Element
cx, cy: Integer
name: String...
show ()...

Window
backgroundColour: Tcolour
cx2, cy2: Integer...
addComp(name, Component)

name
1
0..1

Component

...

Editable
active: Bool...
clean()...

NonEditable
…

Active
…

...

TextField
text: String
nbCars: Integer
...

IntegerField
num: Integer
nbCars: Integer
obtainNum(Int): Bool
…

...

Area
text: String...

ListNonEdit
Elems: Set(String)
addElem(String)...

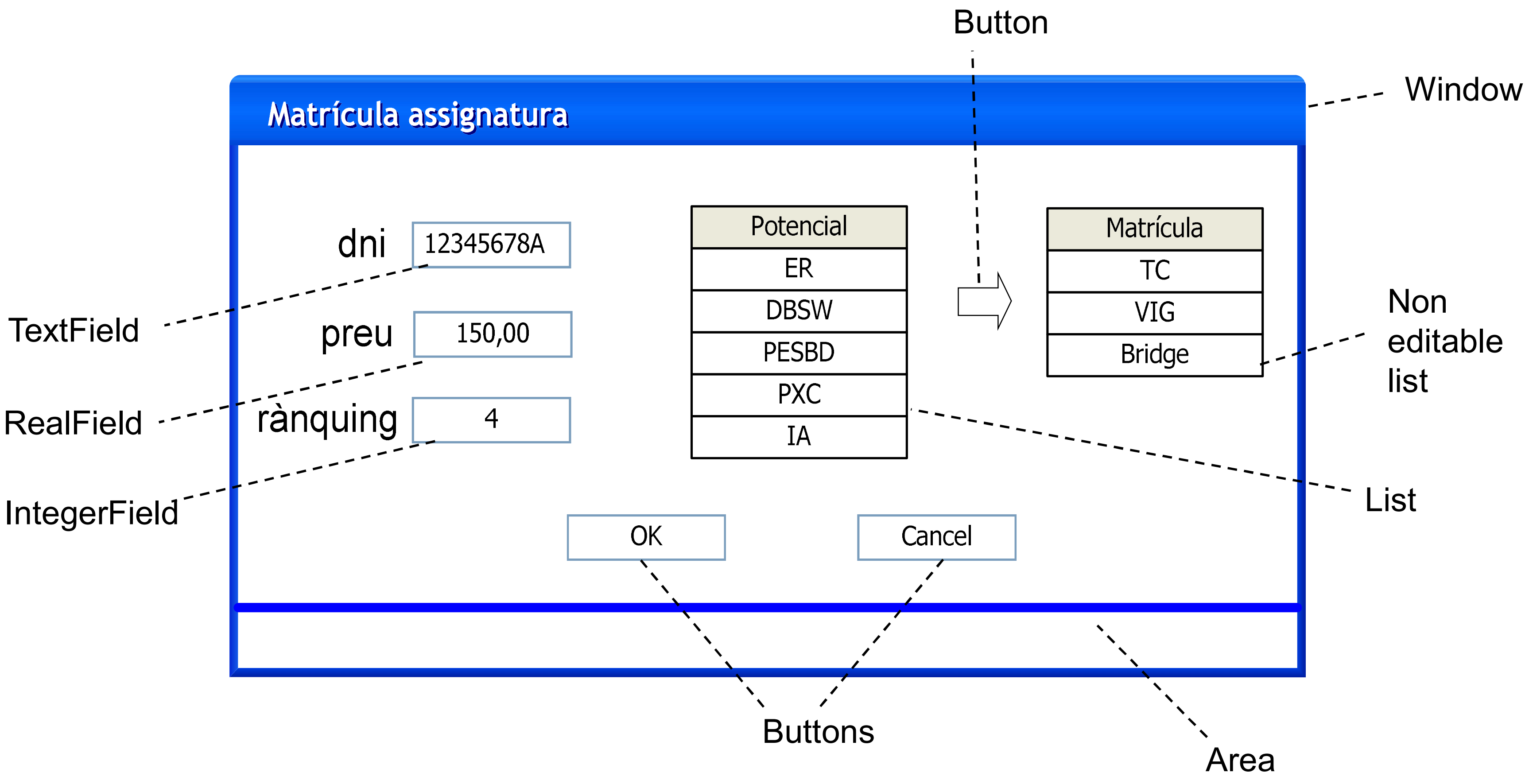
Label
text: String...

...

Button
text: String
pressed()...

List
elems: Set(String)
moveUp()
moveDown()
select(Integer)
current(): String
removeSelect()
addElem(String)...

Annex 1:
Example



Matrícula assignatura

dni 12345678A

preu 150,00

rànquing 4

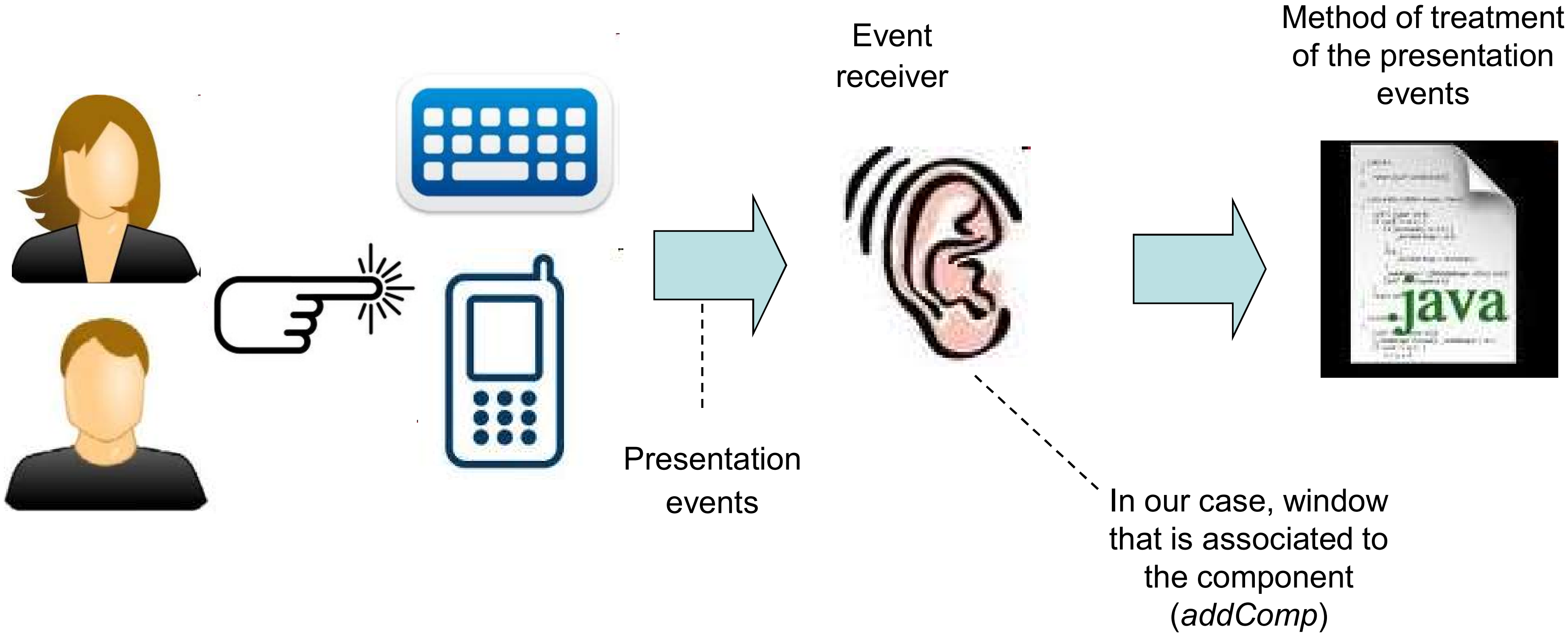
Potencial
ER
DBSW
PESBD
PXC
IA

Matrícula
TC
VIG
Bridge

OK

Cancel

Annex 1:
Management of Presentation Events



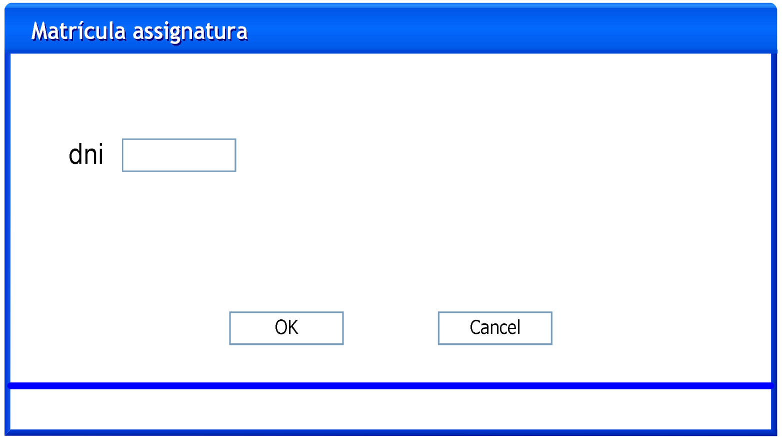
Event receiver

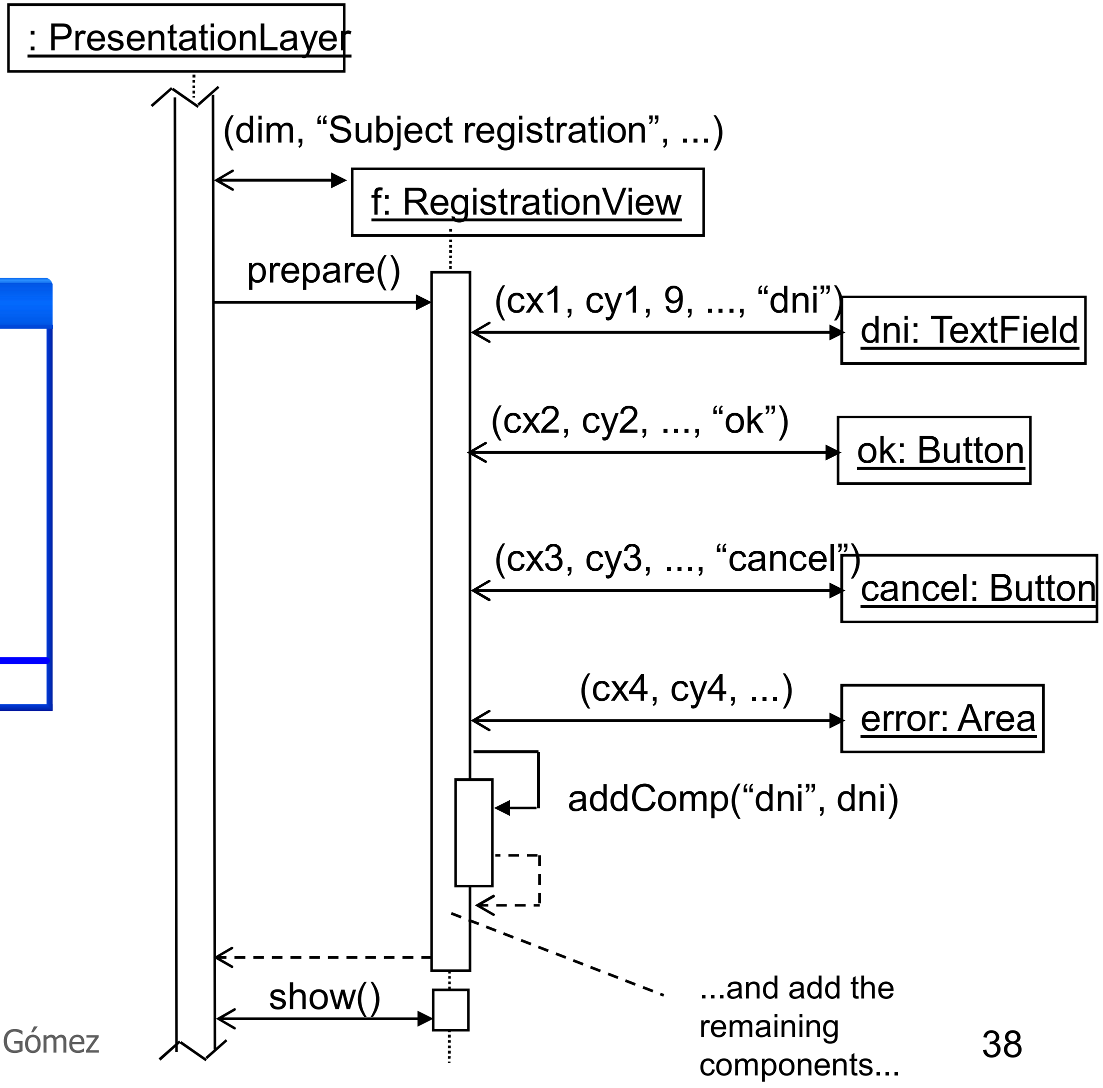
Method of treatment of the presentation events

Presentation events

In our case, window that is associated to the component (*addComp*)

Annex 1:
Example





: PresentationLayer

(dim, “Subject registration”, ...)

f: RegistrationView

prepare()

(cx1, cy1, 9, ..., “dni”)

dni: TextField

(cx2, cy2, ..., “ok”)

ok: Button

(cx3, cy3, ..., “cancel”)

cancel: Button

(cx4, cy4, ...)

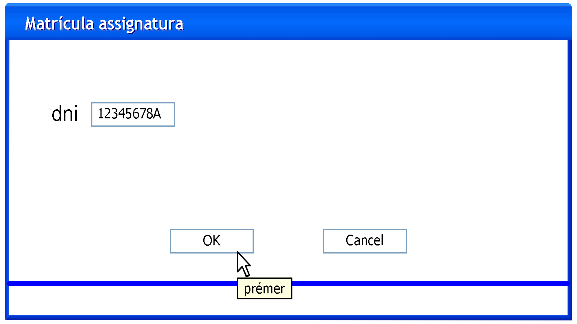
error: Area

addComp(“dni”, dni)

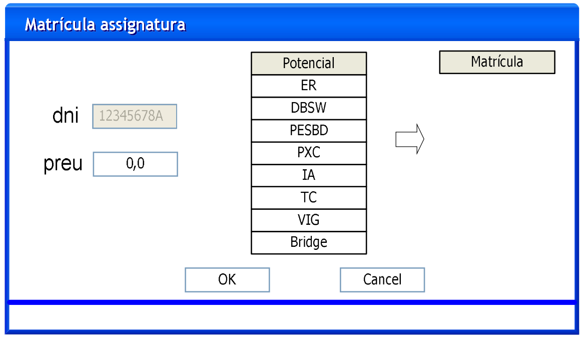
show()

...and add the
remaining
components...

Annex 1:
Example

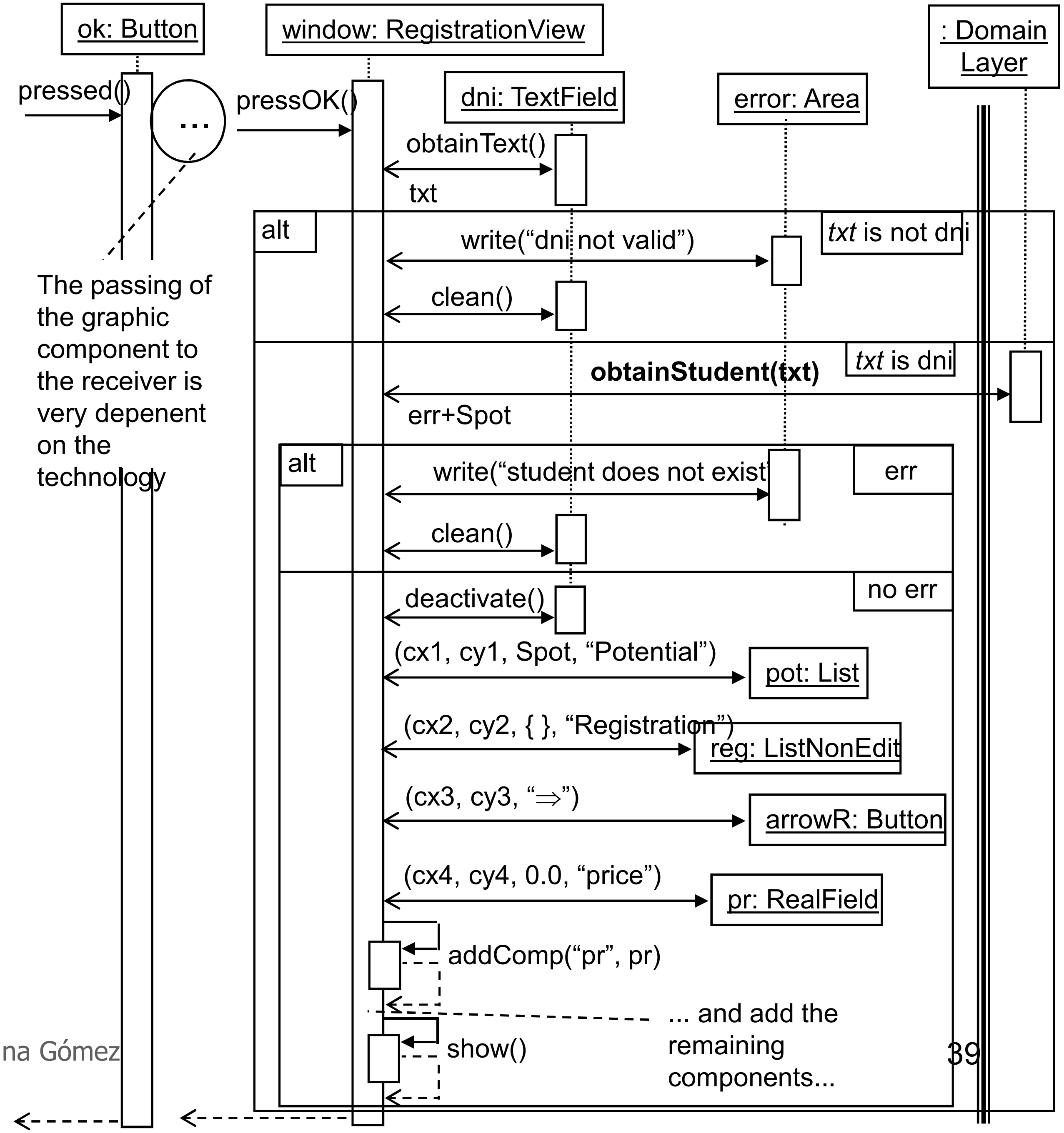

Matrícula assignatura

dni 12345678A


Matrícula assignatura

dni 12345678A

preu 0,0


ok: Button

window: RegistrationView

: Domain Layer

dni: TextField

error: Area

pressed()

...

pressOK()

obtainText()

txt

alt

write(“dni not valid”)

txt is not dni

clean()

obtainStudent(txt)

txt is dni

err+Spot

alt

write(“student does not exist”

err

clean()

no err

deactivate()

(cx1, cy1, Spot, “Potential”)

pot: List

(cx2, cy2, { }, “Registration”)

reg: ListNonEdit

(cx3, cy3, “⇒”)

arrowR: Button

(cx4, cy4, 0.0, “price”)

pr: RealField

addComp(“pr”, pr)

... and add the remaining components...

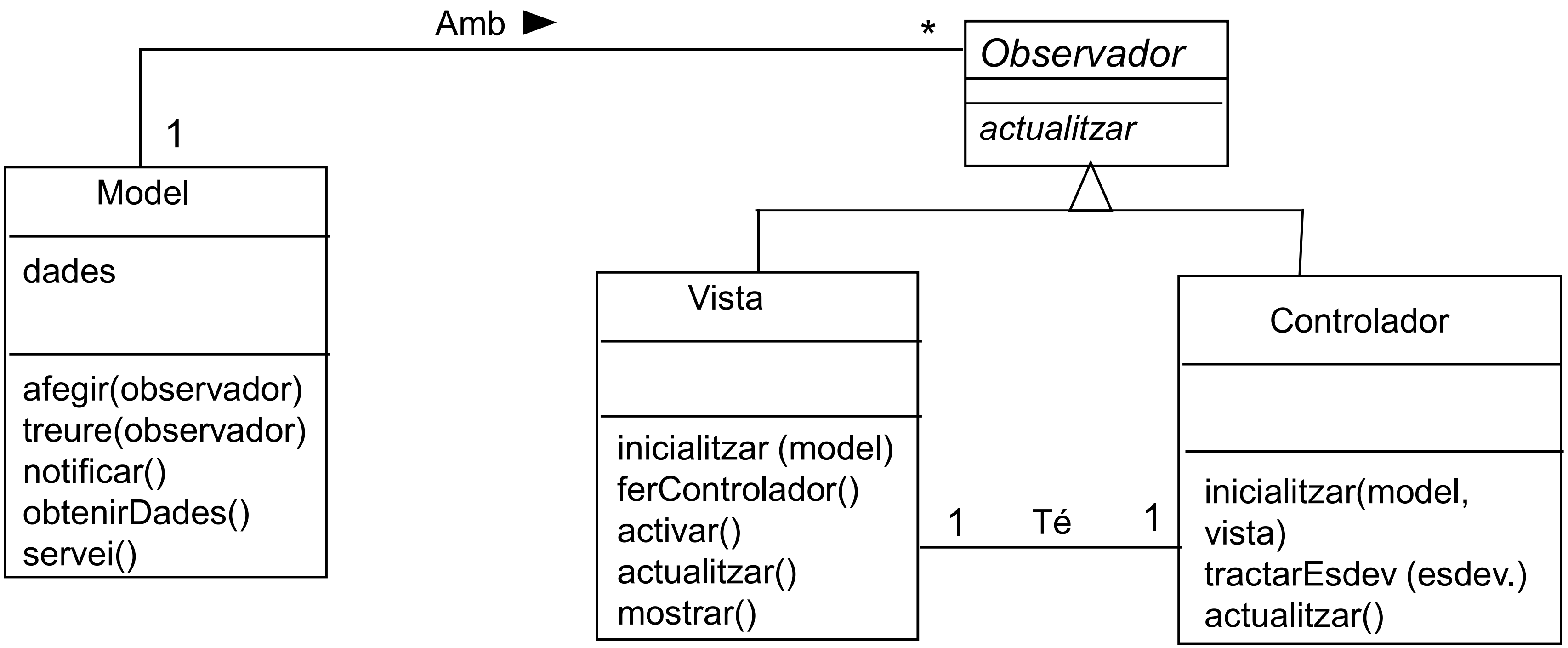
show()

The passing of the graphic component to the receiver is very dependent on the technology

Annex 2:

MVC (active model)

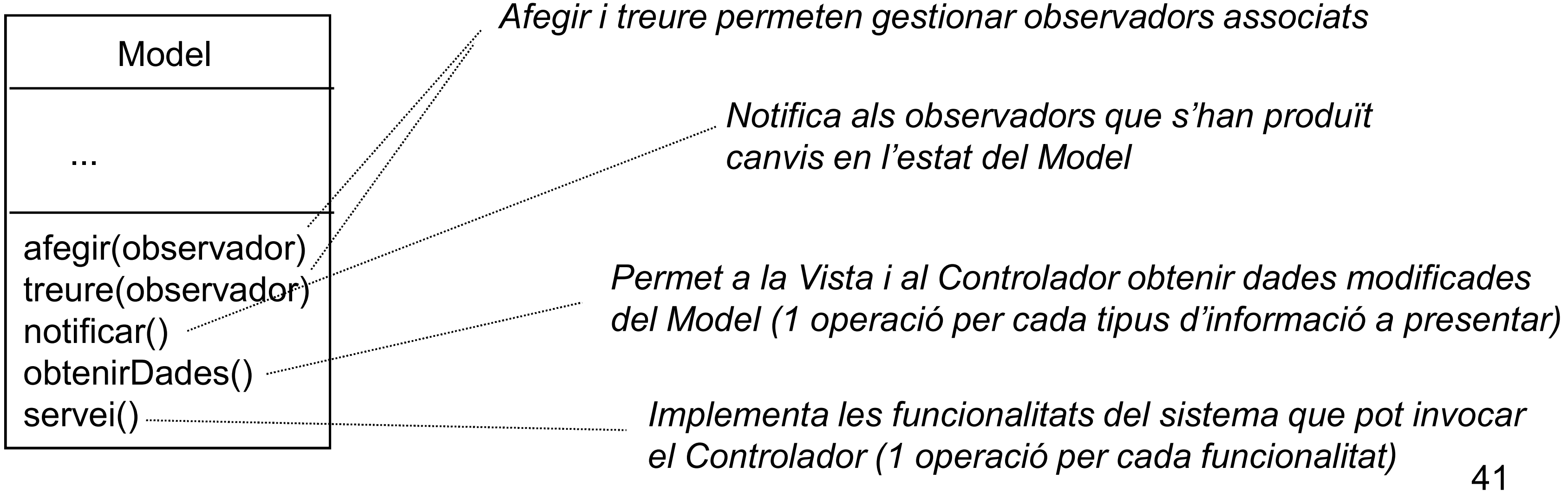
Solució: estructura estàtica



- En general, hi ha *tantes parelles Vista-Controlador com informacions* i formats diferents *es vulguin mostrar* d'un model determinat.
- *Les operacions actualitzar (dels controladors), mostrar i tractarEsdev s'acostumen a redefinir a les subclasses de vista i controlador que es defineixin per a un model determinat*

Annex 2:
MVC (active model)

- **Model**:
  - Encapsula les funcionalitats i les dades del sistema.
  - És independent dels mecanismes de presentació d’informació i d’interacció amb l’usuari.
  - Proporciona al Controlador els serveis per satisfer les peticions de l’usuari.
  - Manté un mecanisme de coordinació amb les Vistes i Controladors associats (patró Observador), per notificar-los qualsevol canvi en el seu estat.



Afegir i treure permeten gestionar observadors associats

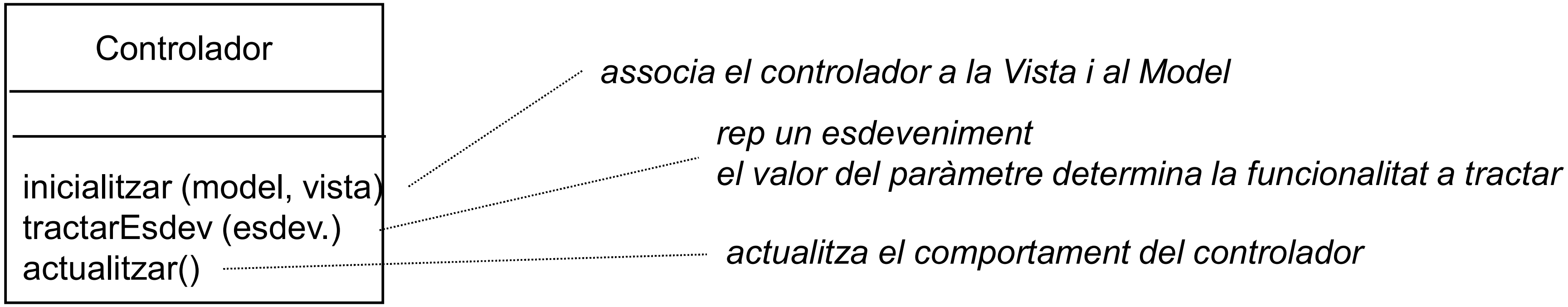
Notifica als observadors que s’han produït canvis en l’estat del Model

Permet a la Vista i al Controlador obtenir dades modificades del Model (1 operació per cada tipus d’informació a presentar)

Implementa les funcionalitats del sistema que pot invocar el Controlador (1 operació per cada funcionalitat)

Annex 2:
MVC (active model)

- **Controlador**:
  - L'usuari interactua amb el sistema únicament mitjançant controladors
  - Gestiona els esdeveniments de presentació i de modificació del model generats per l'usuari.
  - La forma com rep aquests esdeveniments depèn de la plataforma utilitzada per interactuar amb l'usuari (Manipulador d'esdeveniments).
  - Tradueix els esdeveniments de presentació en:
    - invocacions a serveis proporcionats pel Model.
    - peticions de funcionalitats pròpies de la Vista.
  - El comportament del Controlador pot dependre de l'estat del Model.



Controlador

inicialitzar (model, vista)
tractarEsdev (esdev.)
actualitzar()

associa el controlador a la Vista i al Model

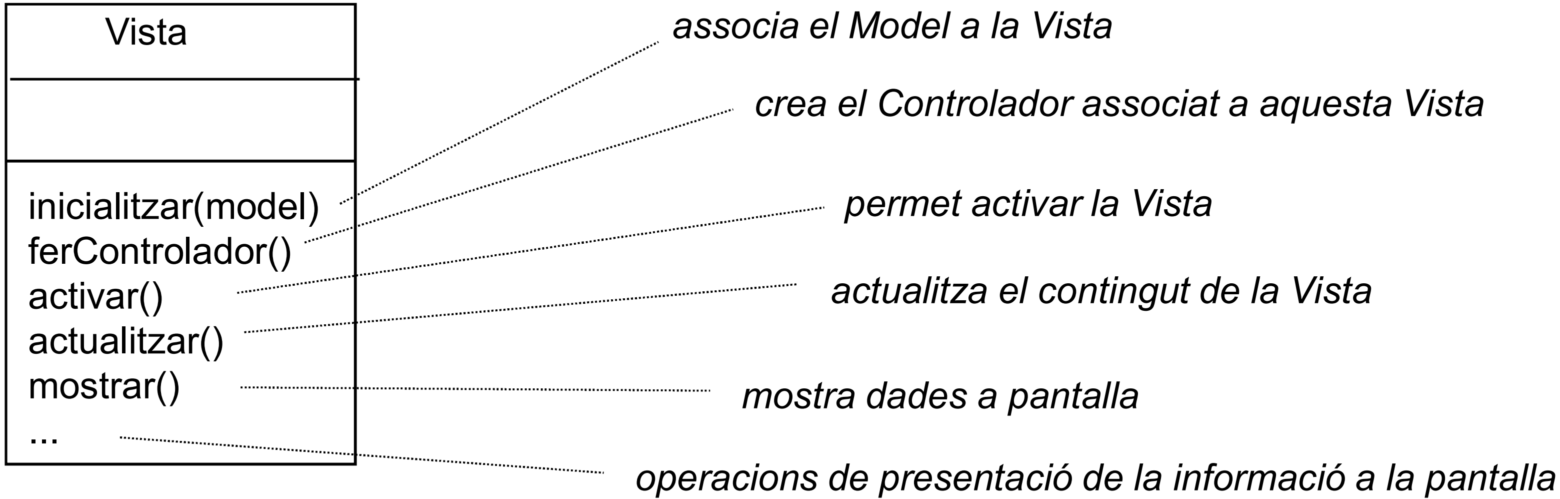
rep un esdeveniment
el valor del paràmetre determina la funcionalitat a tractar

actualitza el comportament del controlador

Annex 2:

MVC (active model)

- **Vista**:
  - Permet presentar informació del model a l’usuari. Hi pot haver diverses vistes d’un mateix model.
  - La informació que mostra es pot veure afectada per canvis en l’estat del Model.
  - Té associat un Controlador que gestiona, si s’escau, els esdeveniments de modificació del Model.
  - Pot proporcionar operacions que permeten que els controladors gestionar la modificació del display (paginació, moure finestra, iconificar, ...).



associa el Model a la Vista

crea el Controlador associat a aquesta Vista

permet activar la Vista

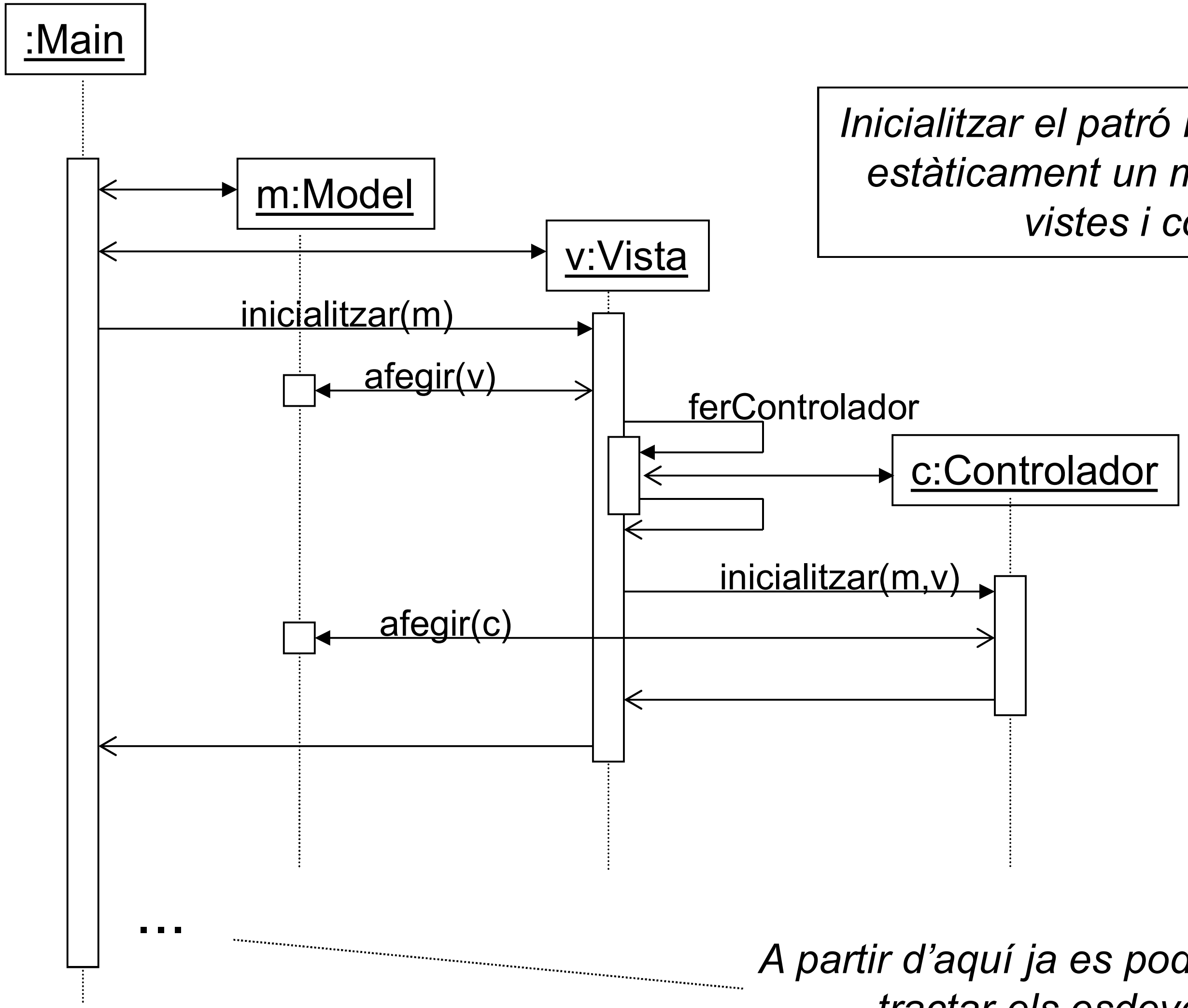
actualitza el contingut de la Vista

mostra dades a pantalla

operacions de presentació de la informació a la pantalla

Annex 2:

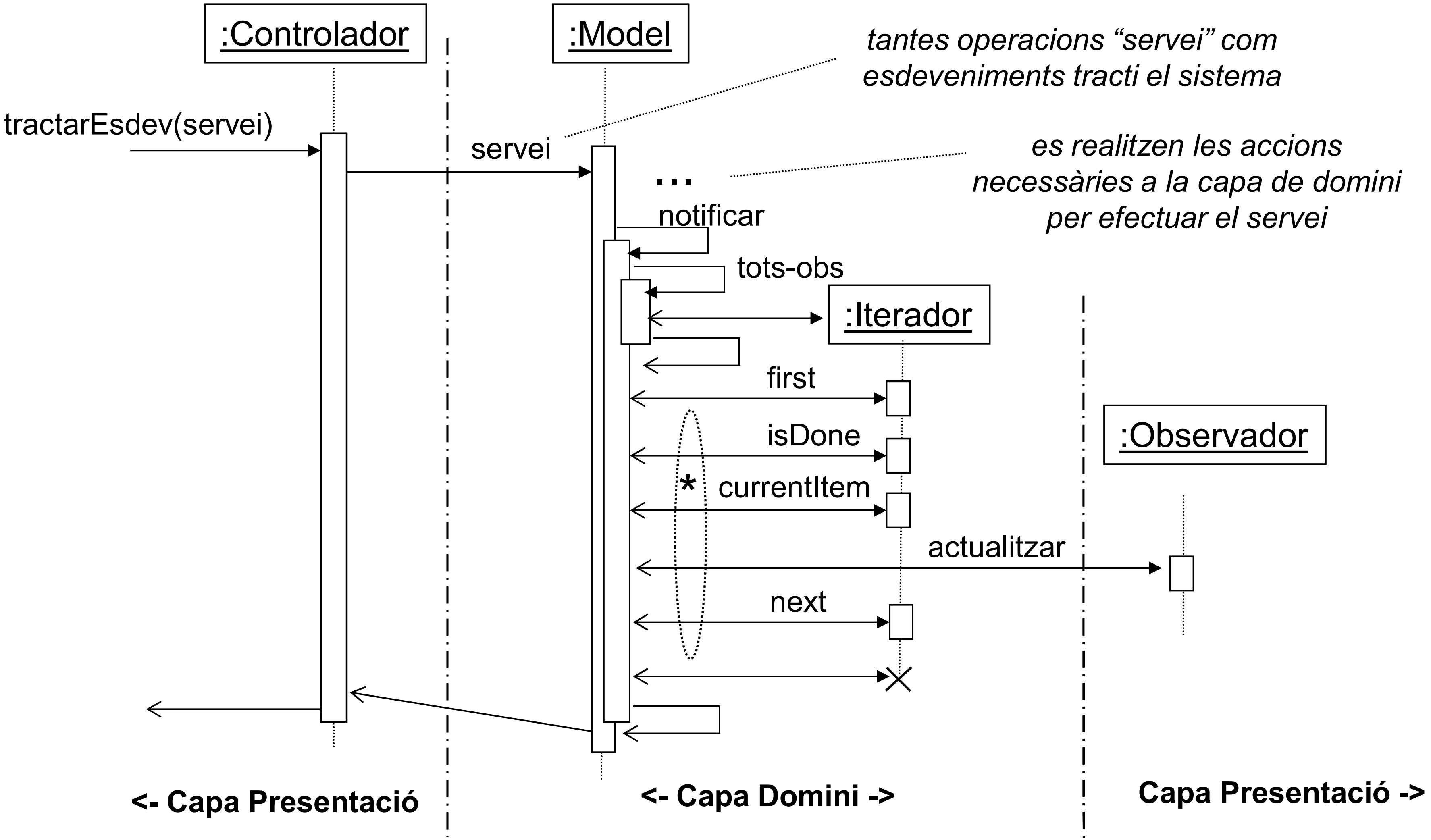
MVC (active model)



Inicialitzar el patró MVC suposa associar estàticament un model amb les seves vistes i controladors.

A partir d'aquí ja es poden començar a tractar els esdeveniments

Annex 2:
MVC (active model)



tantes operacions "servei" com esdeveniments tracti el sistema

es realitzen les accions necessàries a la capa de domini per efectuar el servei

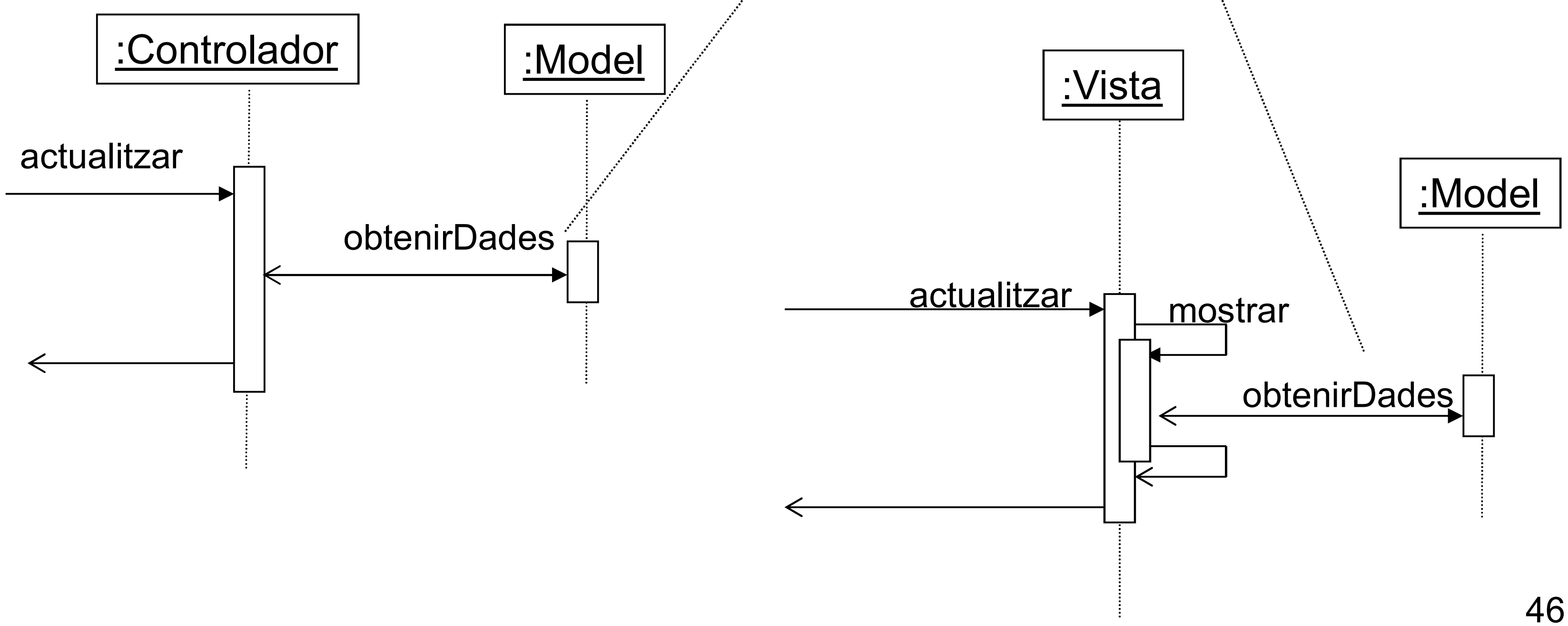
Annex 2:
MVC (active model)

Actualització dels observadors:

- Obté la informació del model, rellevant a la vista i al controlador

- No té perquè ser la mateixa informació i, aleshores, no serà la mateixa operació

Si el controlador modifica el comportament


:Controlador
:Model
actualitzar
obtenirDades
:Vista
:Model
actualitzar
mostrar
obtenirDades