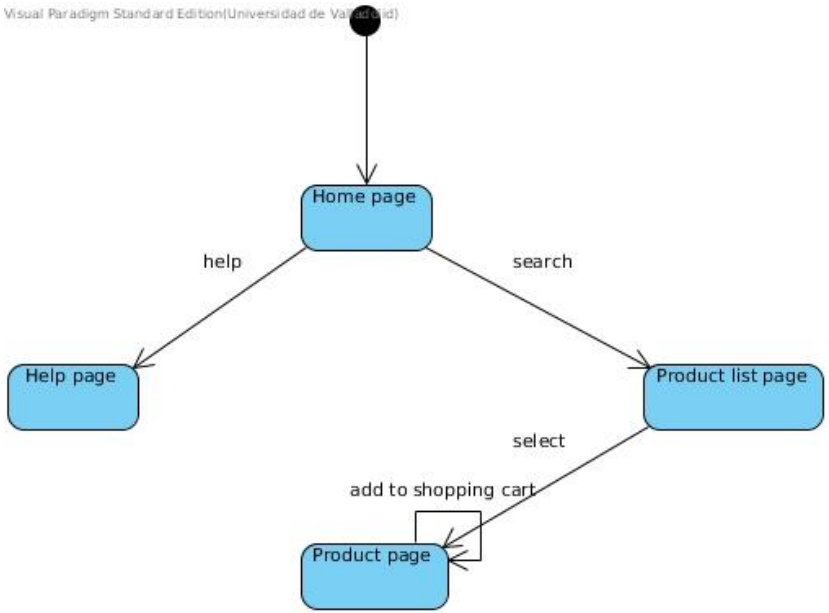
Gestionando múltiples vistas

Interacción Persona Computadora 2014-2015

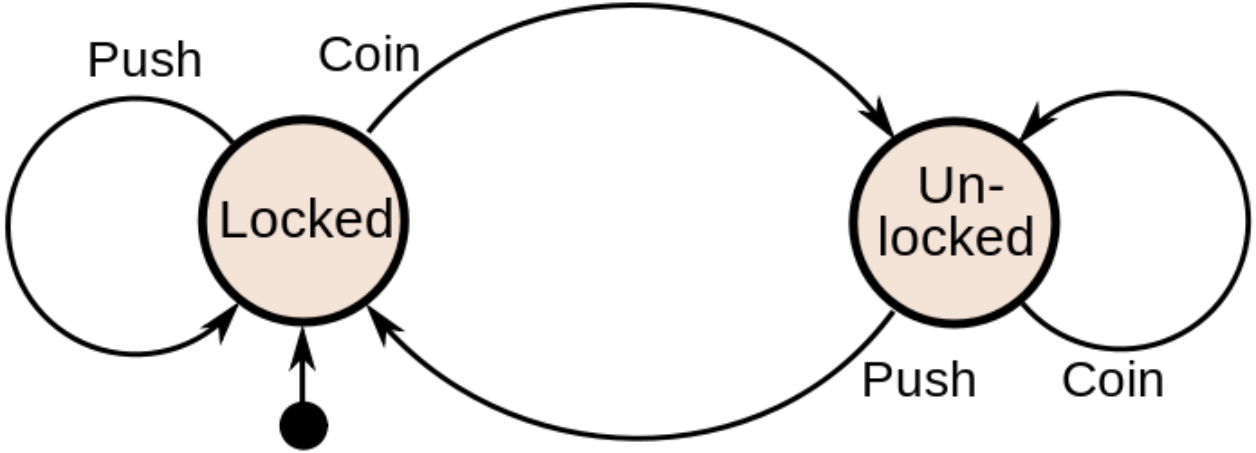
Modelando la navegación en la interfaz de usuario como máquinas de estados



- Algunas aplicaciones, tienen un flujo de vistas complicado.
- Un caso especial son las interfaces de usuario de aplicaciones web.
- A muchos desarrolladores les ha preocupado cómo gestionar mejor el código de la interfaz de usuario
- Una solución son las **máquinas de estados**

¿Qué es una máquina de estados?

- No es un objetivo entrar en profundidad en este tema.
- Definición: es un modelo de computación que consta de
  - Estados (un estado especial llamado Estado inicial)
  - Transiciones o eventos
  - Entrada + Estado actual ⇨ Estado siguiente (transición)
- Máquina de estados finitos o Autómata finito
- La máquina se encuentra en UNO de un conjunto finito de estados.
- Ejemplos habituales en la vida diaria: máquinas de vending, ascensores, semáforos en un cruce de vías.

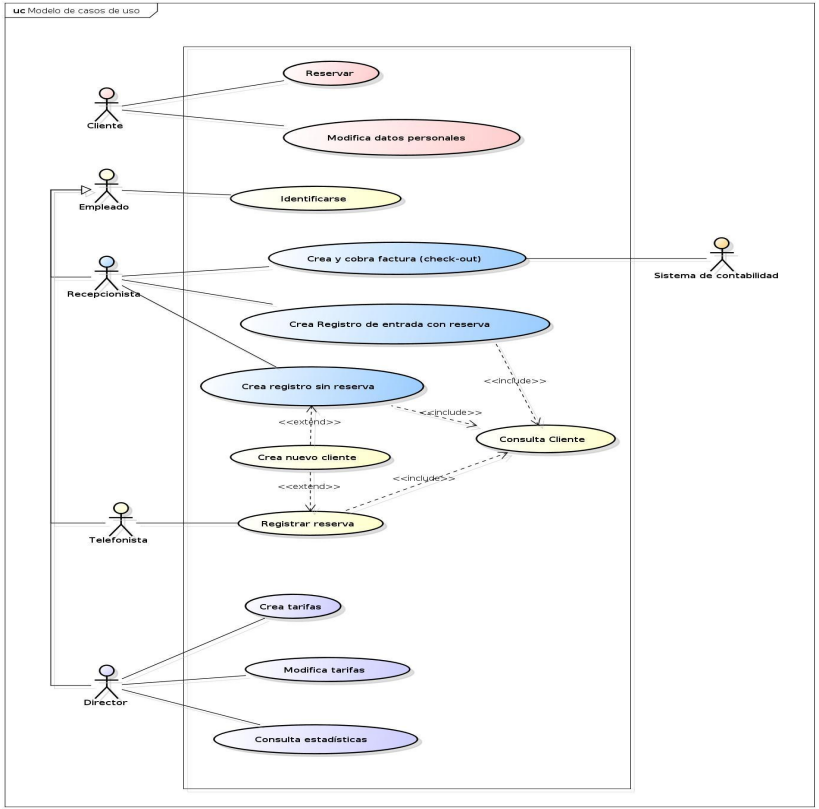


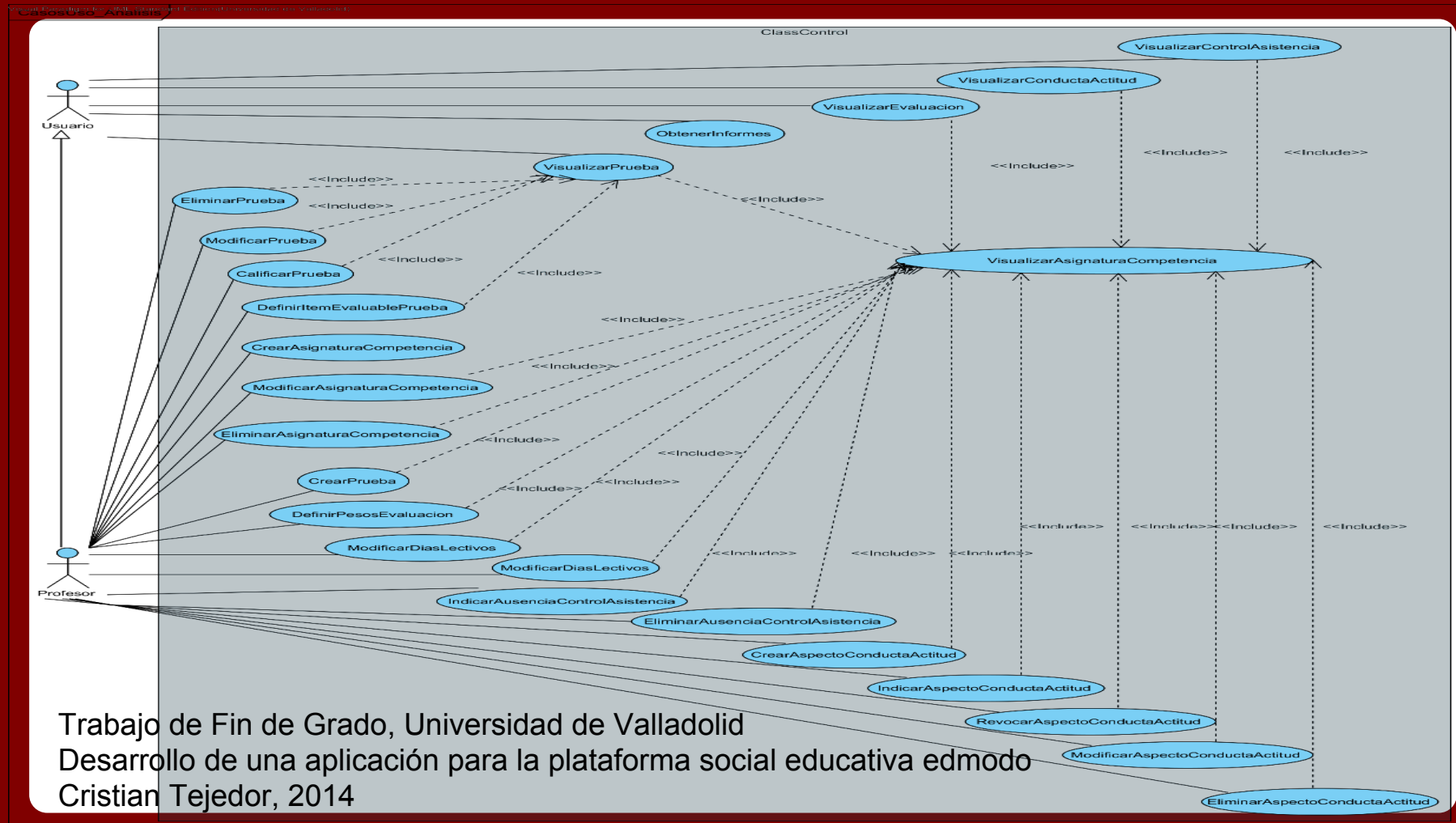
"Torniqueterevolution" by Sebasgui - Own work. Licensed under GFDL via Wikimedia Commons - http://commons.wikimedia.org/wiki/File:Torniqueterevolution.jpg#/media/File:Torniqueterevolution.jpg

Un torniquete

Casos de uso y escenarios

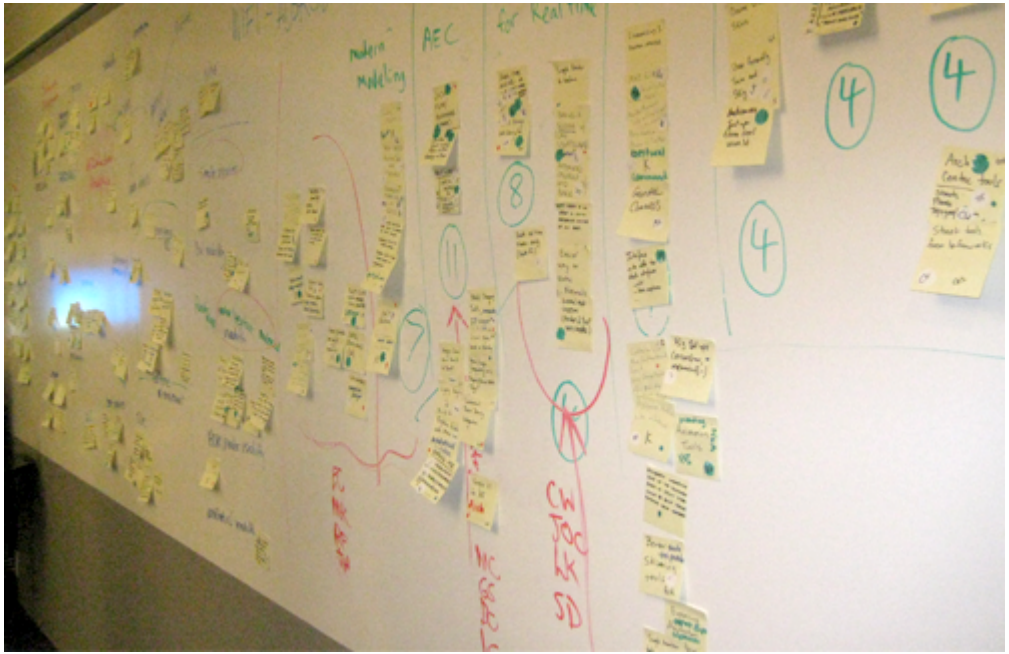
- El diseño de una aplicacion interactiva está basado en casos de uso (o user stories)
- En un caso de uso encontramos varios escenarios
- Cada escenario puede tener diferentes vistas





Trabajo de Fin de Grado, Universidad de Valladolid
Desarrollo de una aplicación para la plataforma social educativa edmodo
Cristian Tejedor, 2014

Diseñando vistas con post-its



Diseño de la interfaz (I)

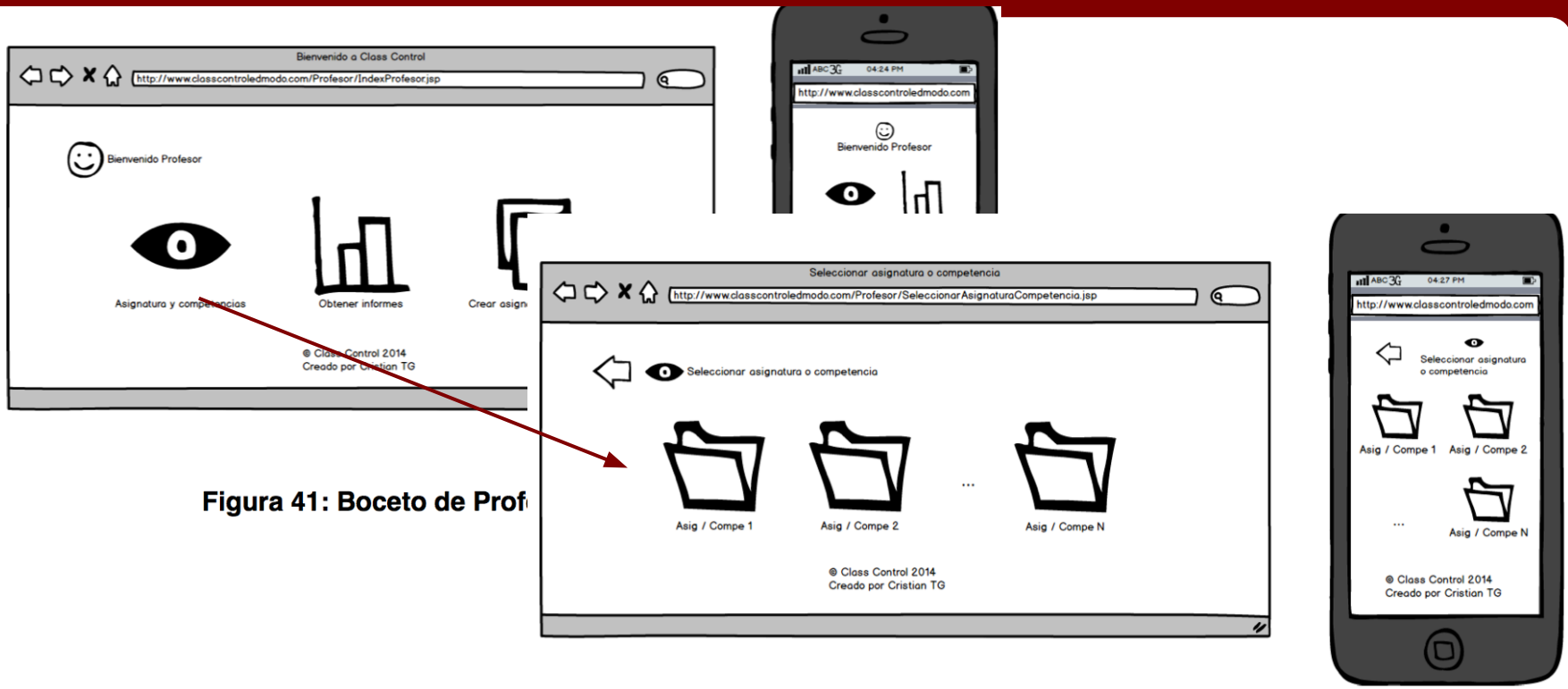


Figura 41: Boceto de Prof

Figura 42: Boceto de Profesor: seleccionar asignatura o competencia

Diseño de la interfaz (II)

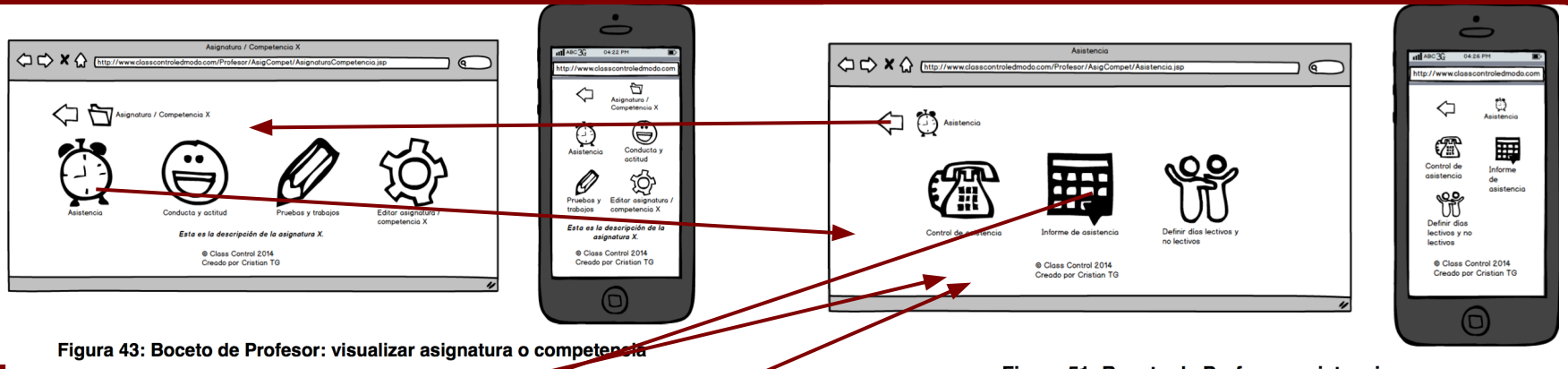


Figura 43: Boceto de Profesor: visualizar asignatura o competencia

Figura 51: Boceto de Profesor: asistencia

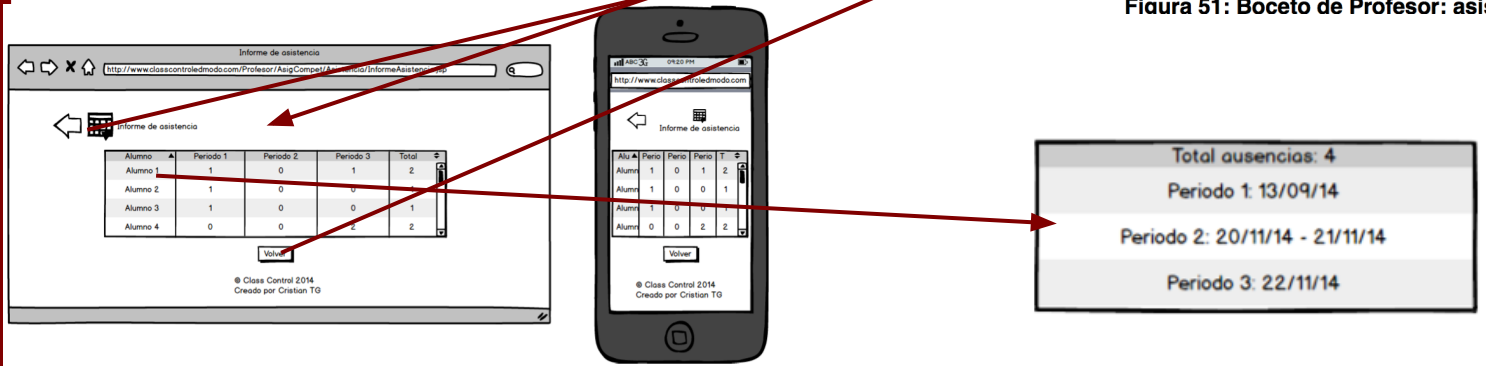
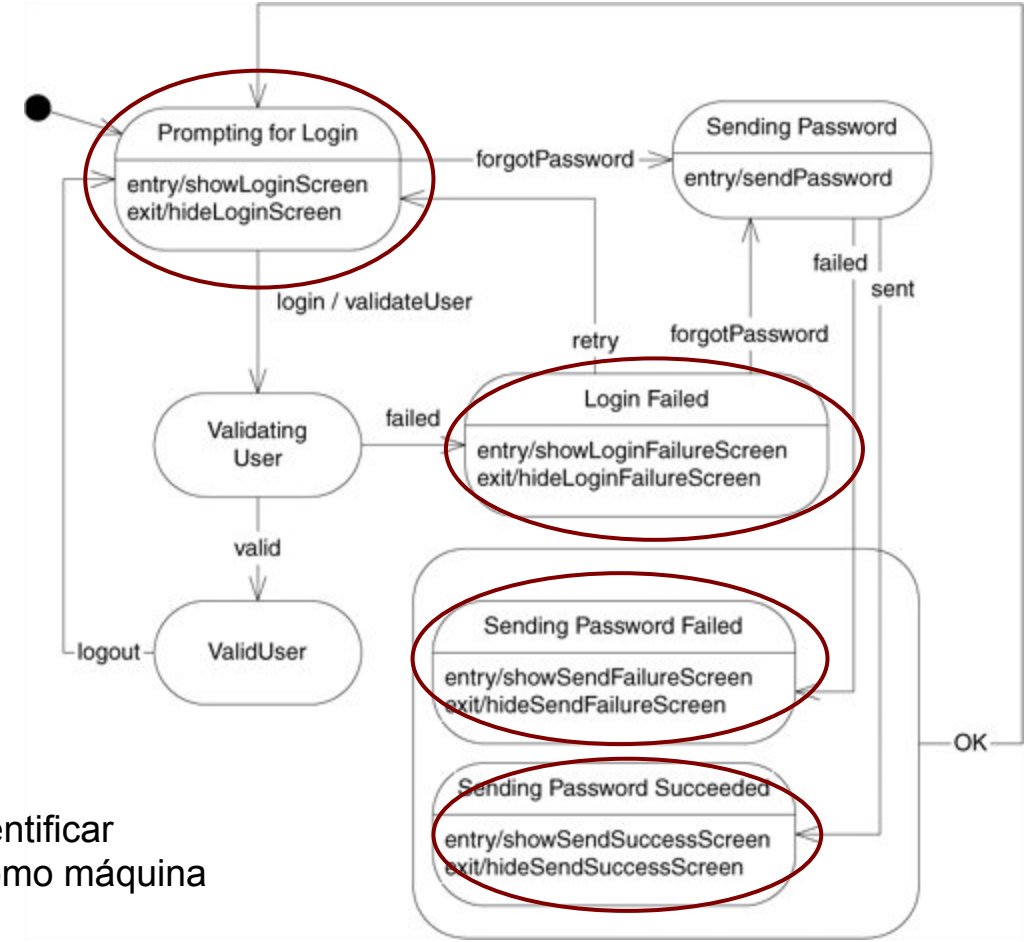
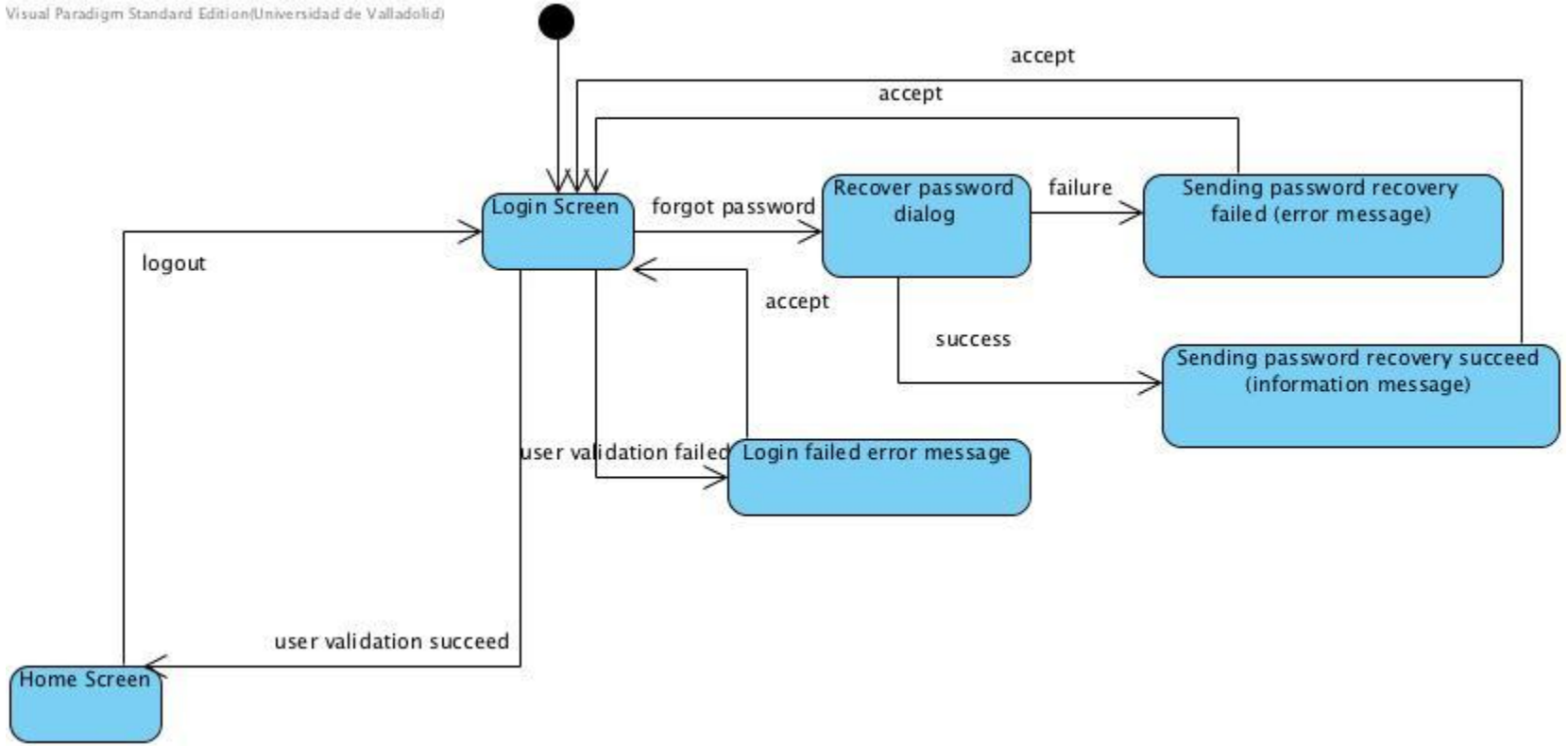


Figura 54: Boceto de Profesor: visualizar informe asistencia

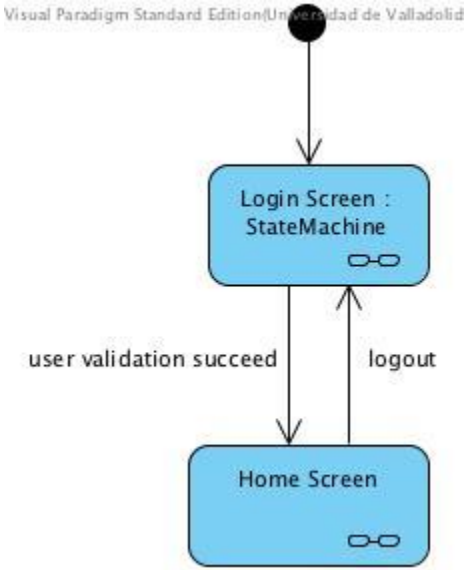


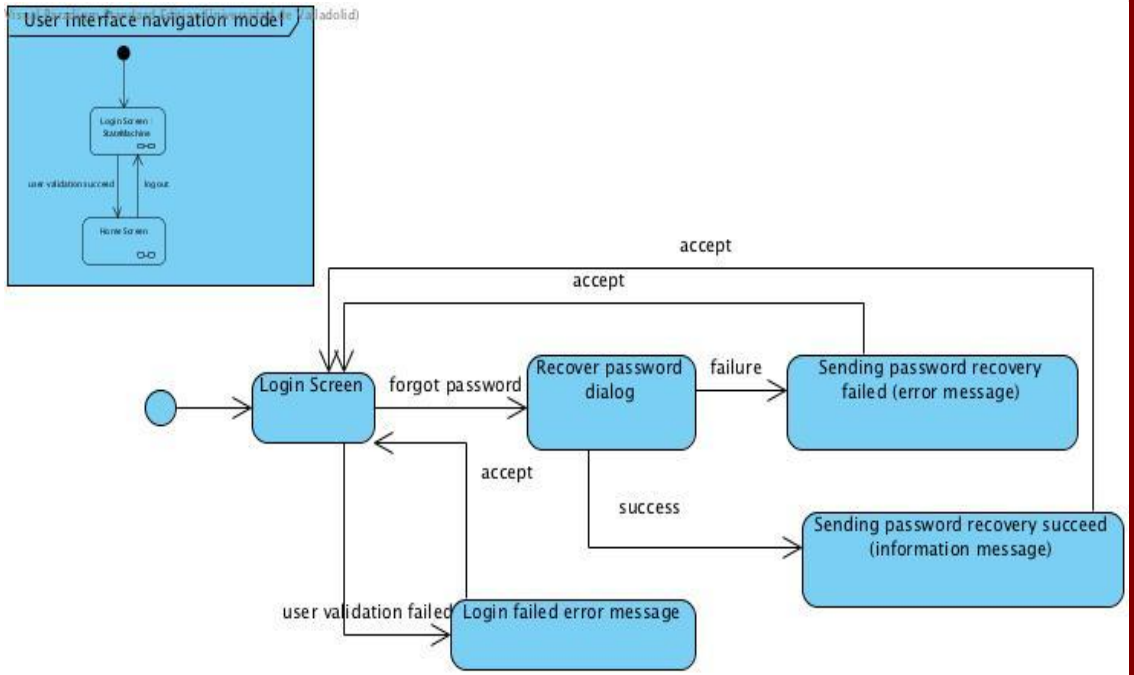
El proceso de identificar usuario (login) como máquina de estados

Login desde el punto de vista de interfaz



En realidad hay varias máquinas de estados





```
public class Main {
        private static LoginStateMachine loginStateMachine;
        private static HomeStateMachine  homeStateMachine;

        public static void main(String args[]) {
                /* Set the Nimbus look and feel */
                loginStateMachine = new LoginStateMachine();
        }
        public static LoginStateMachine getStateMachineLogin() {
                return loginStateMachine;
        }
        public static void loginSucceed() {
                loginStateMachine.close();
                homeStateMachine = new HomeStateMachine();
        }
        public static HomeStateMachine getStateMachineHome() {
                return homeStateMachine;
        }
}
```

La primera máquina y principal tiene dos submáquinas,
un cambio de estado de Login a Home

(añadid un cambio de estado de Home a Login
cuando se haga logout en Home)

Una de las submáquinas de estados (I)

```
public class LoginStateMachine {
        private JFrame currentState;

        public LoginStateMachine() {
          java.awt.EventQueue.invokeLater(
            new Runnable() {
              public void run() {
                currentState = new LoginWindow();
                currentState.setVisible(true);
              }
          });
        } //LoginStateMachine()

        public void recoverPassword() {
          currentState.setVisible(false); // si se desea ocultar
          currentState.dispose();   // si se desea destruir
```

```
    //realiza transición
  java.awt.EventQueue.invokeLater(
    new Runnable() {
      public void run() {
        currentState = new PasswordRecoveryWindow();
        currentState.setVisible(true);
      }
    });
  } //recoverPassword
```

Una de las submáquinas de estados (II)

```
public void help() {
    currentState.setVisible(false); // si se desea ocultar
    currentState.dispose();   // si se desea destruir
    //realiza transición
    java.awt.EventQueue.invokeLater(
      new Runnable() {
        public void run() {
          currentState = new HelpWindow();
          currentState.setVisible(true);
        }
      });
 } //help
```

```
    void close() {
      currentState.setVisible(false); // si se desea ocultar
      currentState.dispose();   // si se desea destruir
    } //close
} //Login class
```

La primera submáquina tiene el estado inicial (a través del constructor) LoginWindow
los estados
- **HelpWindow y**
- **PasswordRecoveryWindow**

(añadid la submáquina de estados Home)

Este código se puede mejorar

- Hay varias cosas duplicadas
- Hay objetos que deben ser accesibles desde todas partes garantizando una única instancia
- Se aplican patrones de diseño
  - Singleton
  - State
- Pero no deben ser objetivo de este curso

Referencias de interés

http://www.visual-paradigm.com/support/documents/vpuserguide/2822/2613/83713_whatisawiref.html
http://www.visual-paradigm.com/support/documents/vpuserguide/2822_wireframe.html

http://www.bennadel.com/blog/2242-treating-user-interface-ui-widgets-like-finite-state-machines.htm

http://lassala.net/2008/02/05/state-machines-and-gui-interaction-part-i/
http://lassala.net/2008/02/19/state-machines-and-gui-interaction-part-ii/

http://www.drdobbs.com/jvm/state-machines-user-interfaces/184405248
http://www.visual-paradigm.com/support/documents/vpuserguide/276/386/28107_generatingst.html