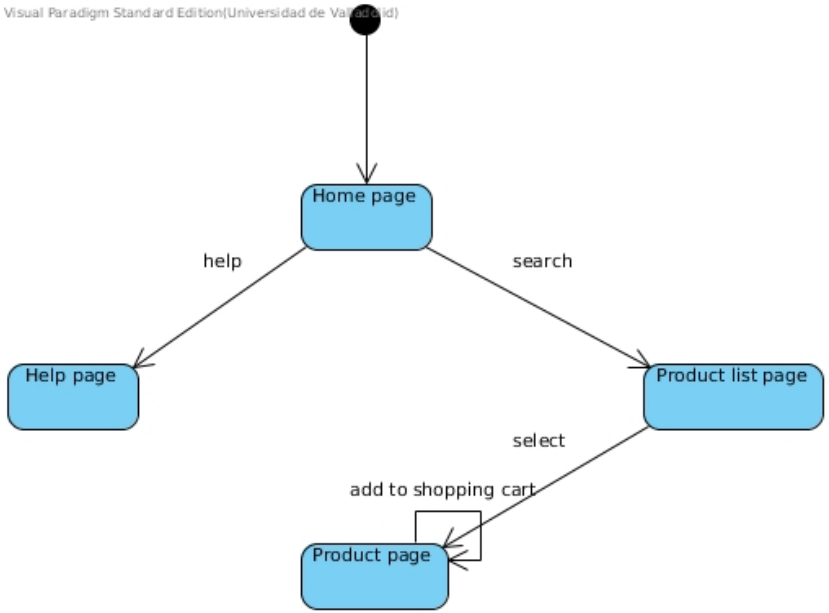
Gestionando múltiples vistas

Interacción Persona Computadora 2018-2019

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/diagramas de estados**

¿Qué es una máquina de estados?

No es un objetivo entrar en profundidad en este tema.

Es un modelo de computación que consta de

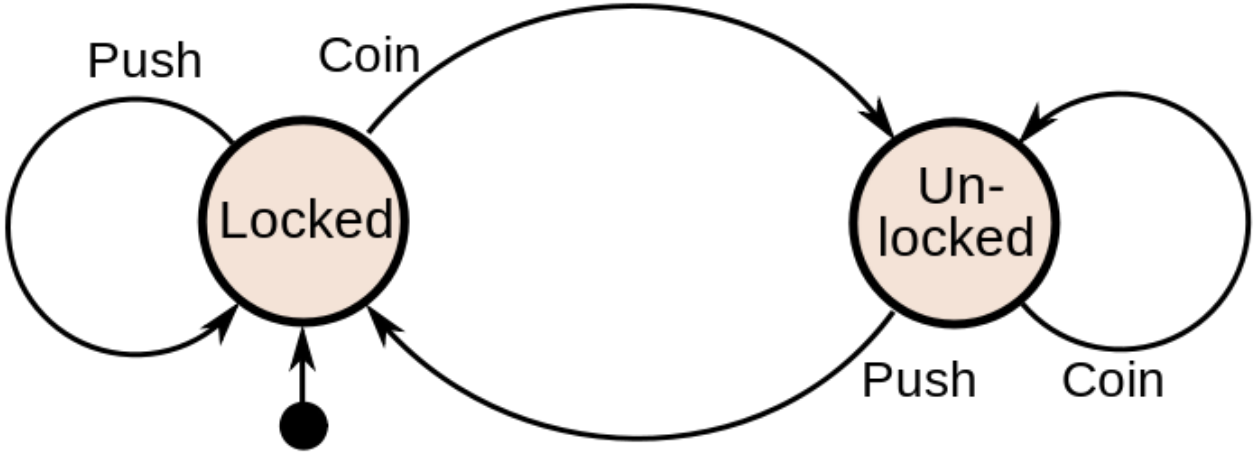
Estados (un estado especial llamado Estado inicial)

Transiciones o eventos

De manera que

Estado actual + Evento ⇨ Estado siguiente (transición)

- 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

Máquina de estados

- **Estados**
  - Se describen por sus efectos y relaciones.
- **Eventos**
  - Tipo de ocurrencia significativa, localizada en un tiempo y en un espacio.
- **Transiciones**
  - Define la respuesta del objeto. Tiene un evento que la dispara, una condición de guarda y un estado destino.

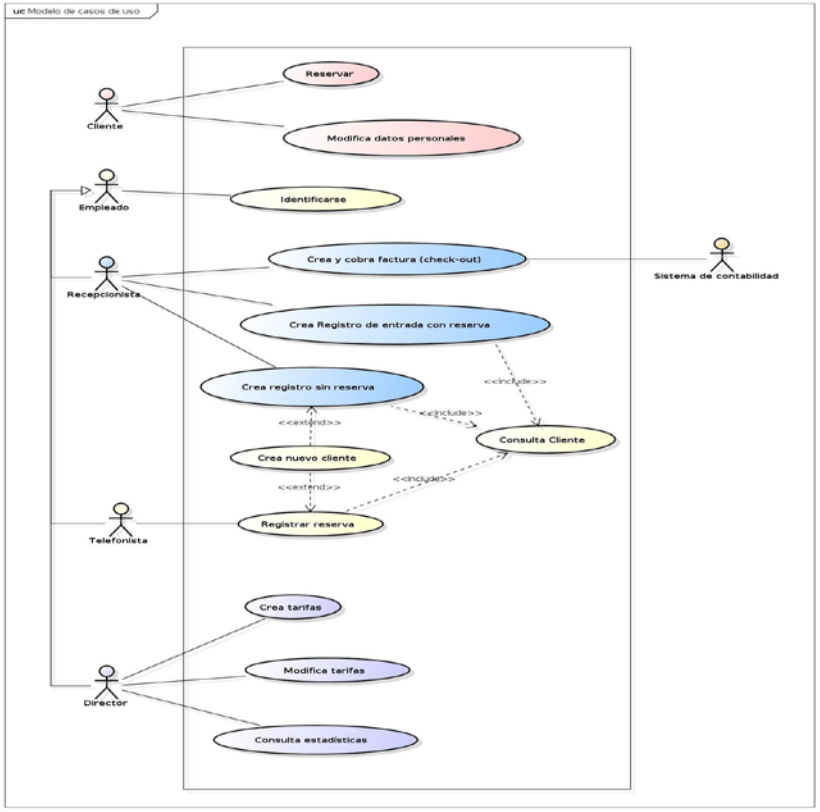
Máquina de estados

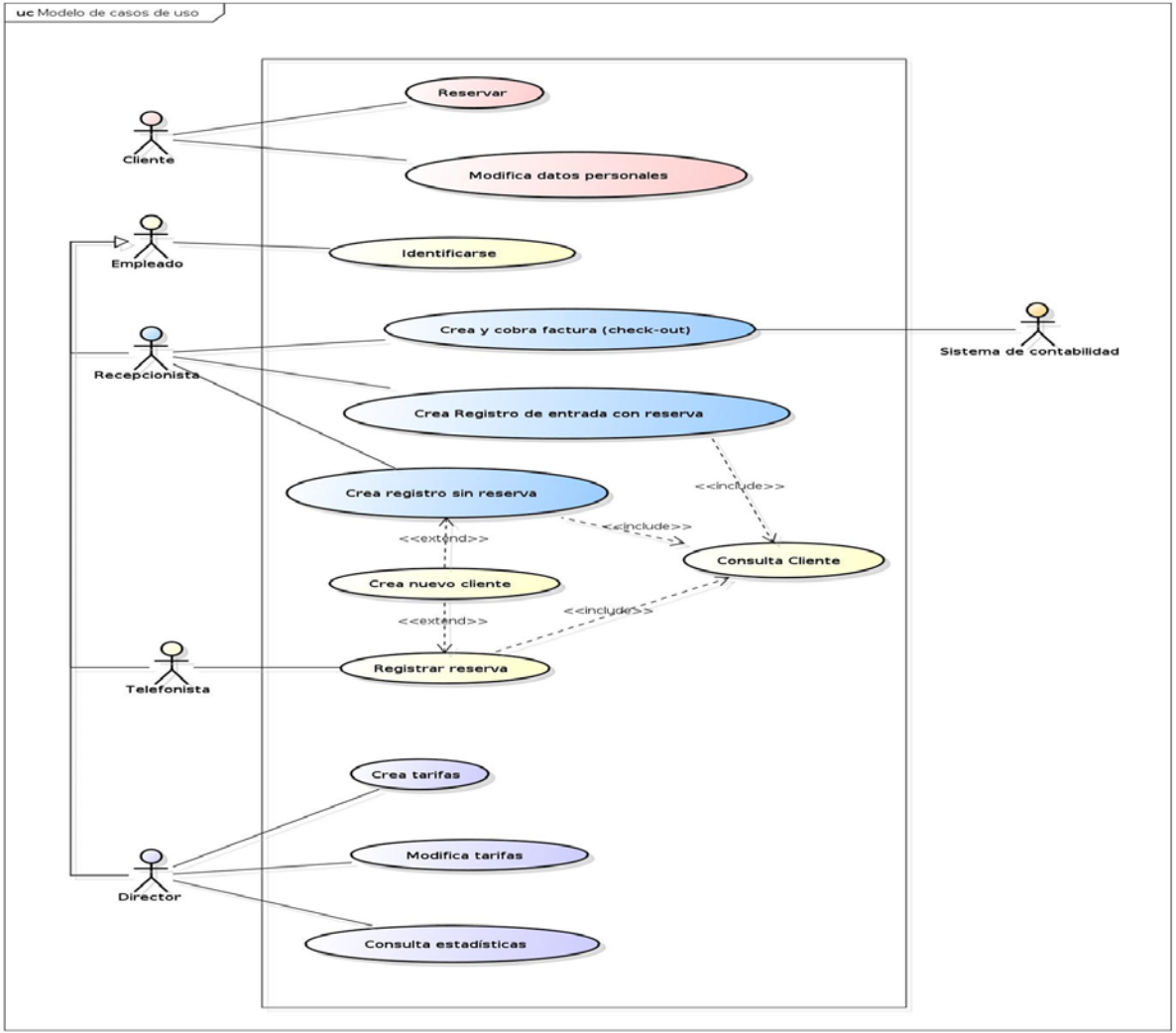
Transiciones:

- Ponen en marcha “actividad” en la clase (cuando está en ese ESTADO).
  - Actividad de entrada: lo que se hace al entrar en el estado
  - Actividad de salida: lo que se hace cuando se sale del estado

Casos de uso y escenarios

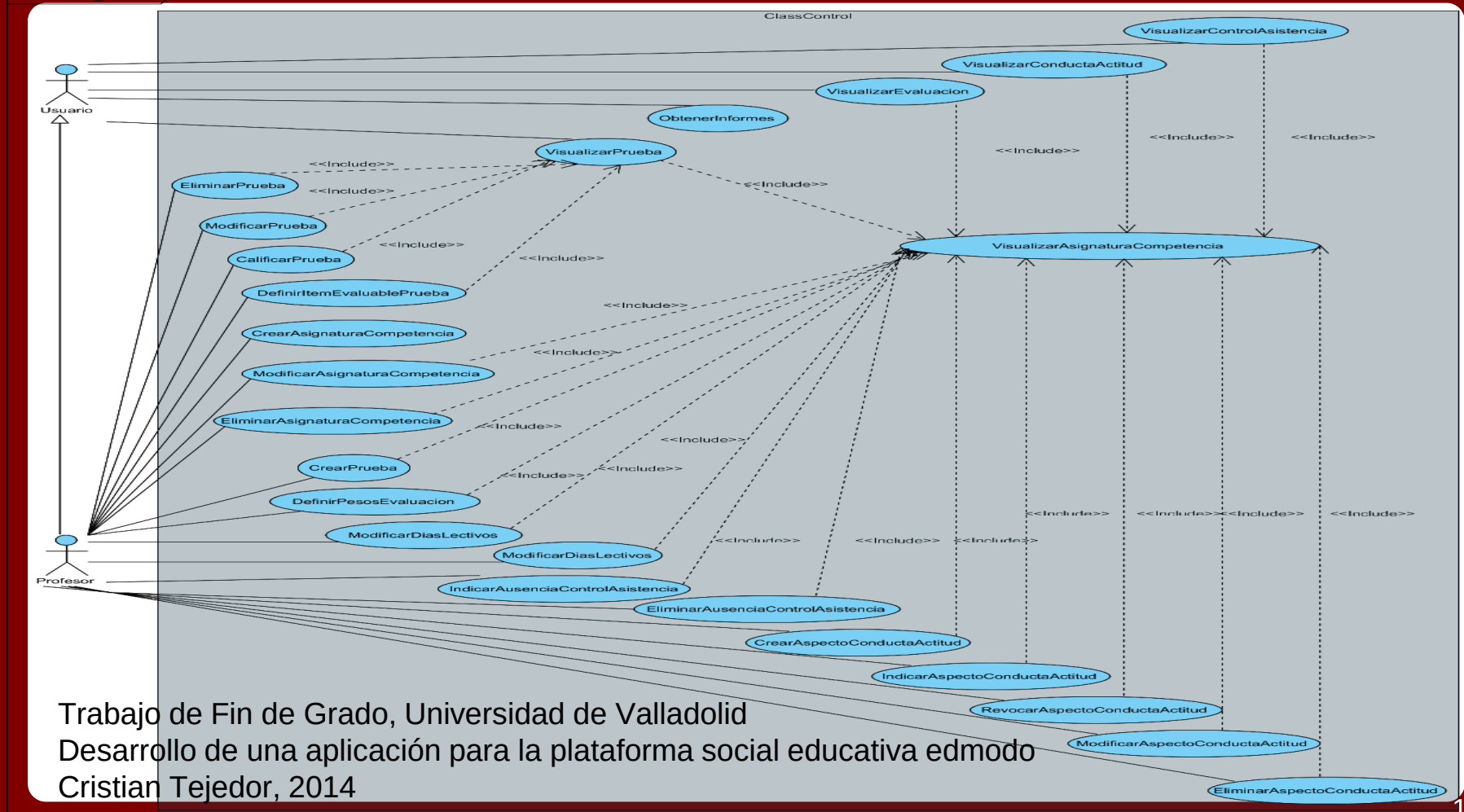
- El diseño de una aplicación interactiva está basado en casos de uso (o user stories)
- En un caso de uso puede tener diferentes vistas




uc Modelo de casos de uso
Reservar
Cliente
Modifica datos personales
Empleado
Identificarse
Crea y cobra factura (check-out)
Sistema de contabilidad
Recepcionista
Crea Registro de entrada con reserva
Crea registro sin reserva
<<include>>
<<include>>
<<extend>>
Consulta Cliente
Crea nuevo cliente
<<extend>>
<<include>>
Telefonista
Registrar reserva
Crea tarifas
Modifica tarifas
Director
Consulta estadísticas

Casos de uso y escenarios

- Un diagrama de casos de uso de una aplicación real es muy complejo.
- El flujo entre las diferentes ventanas es complejo.
- Nos preguntamos cómo llevar control del mismo durante el diseño y la implementación.



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

Diseñando vistas con BalsamiQ (I)

Desarrollo de una aplicación para la plataforma social educativa edmodo

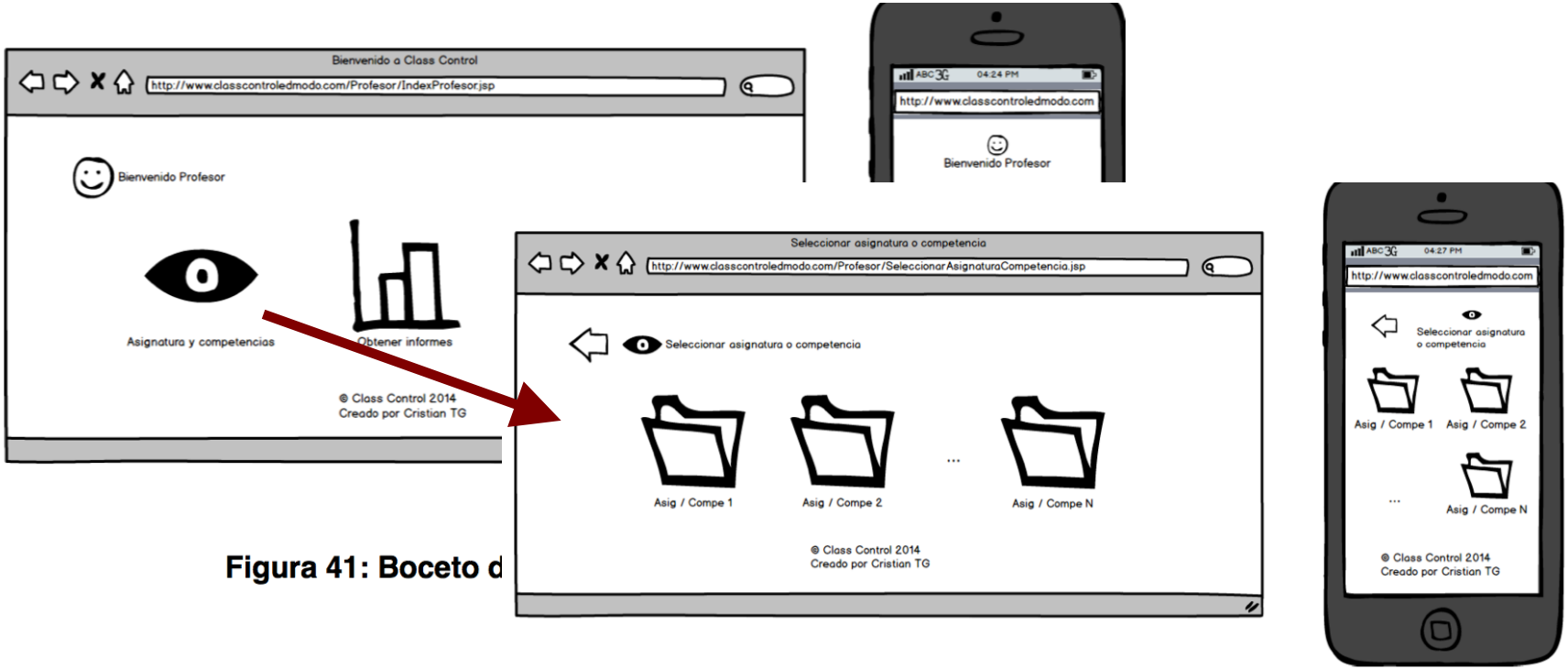


Figura 41: Boceto d

Figura 42: Boceto de Profesor: seleccionar asignatura o competencia

Diseñando vistas con BalsamiQ (II)

Desarrollo de una aplicación para la plataforma social educativa edmodo

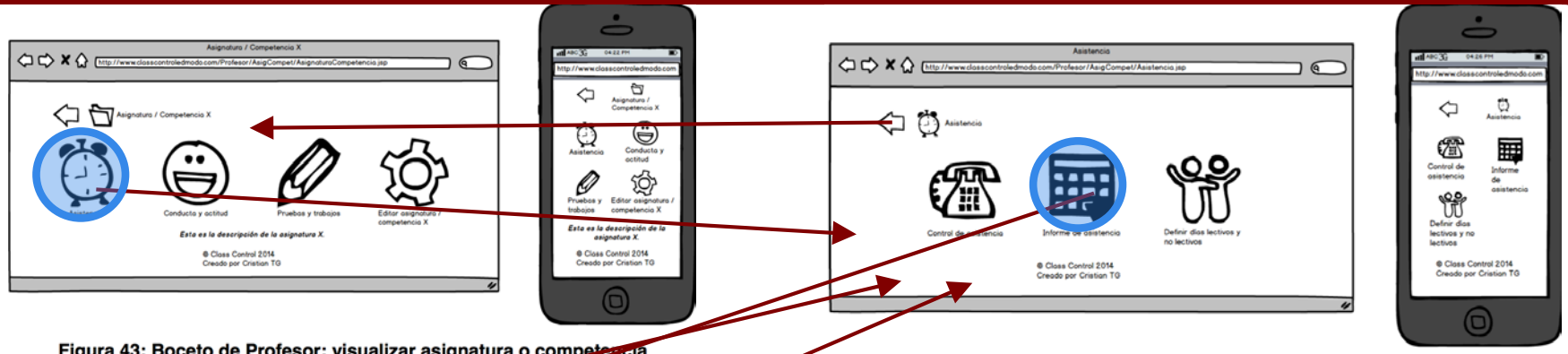


Figura 43: Boceto de Profesor: visualizar asignatura o competencia

Figura 51: Boceto de Profesor: asistencia

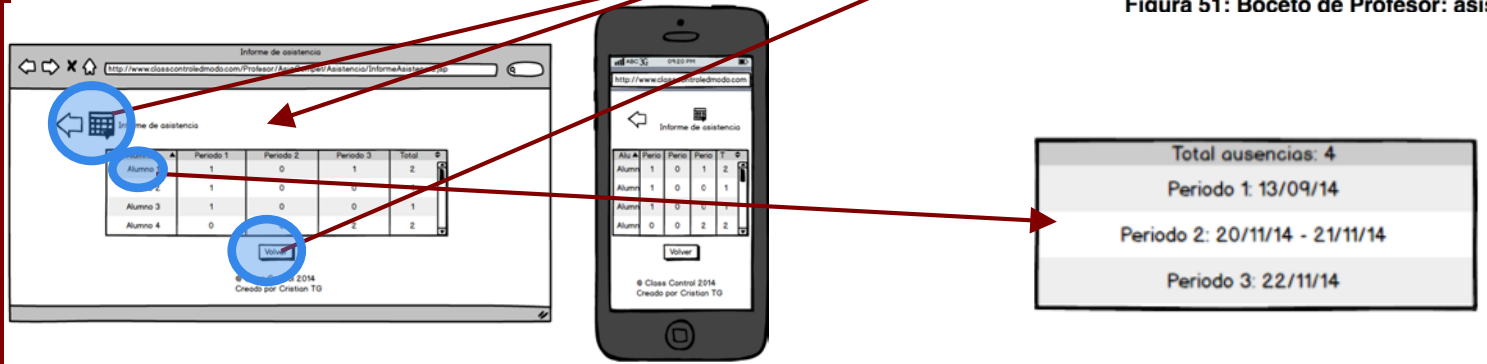


Figura 54: Boceto de Profesor: visualizar informe asistencia

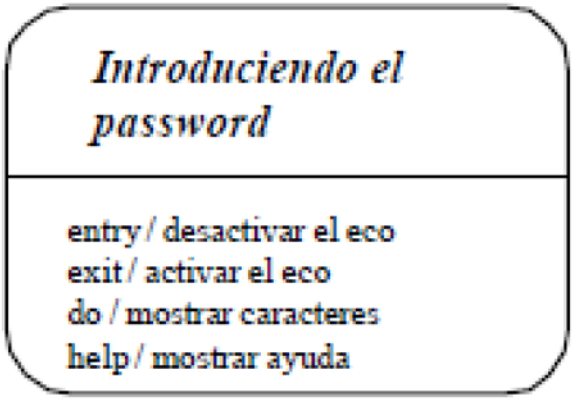
Con este flujo de vistas….

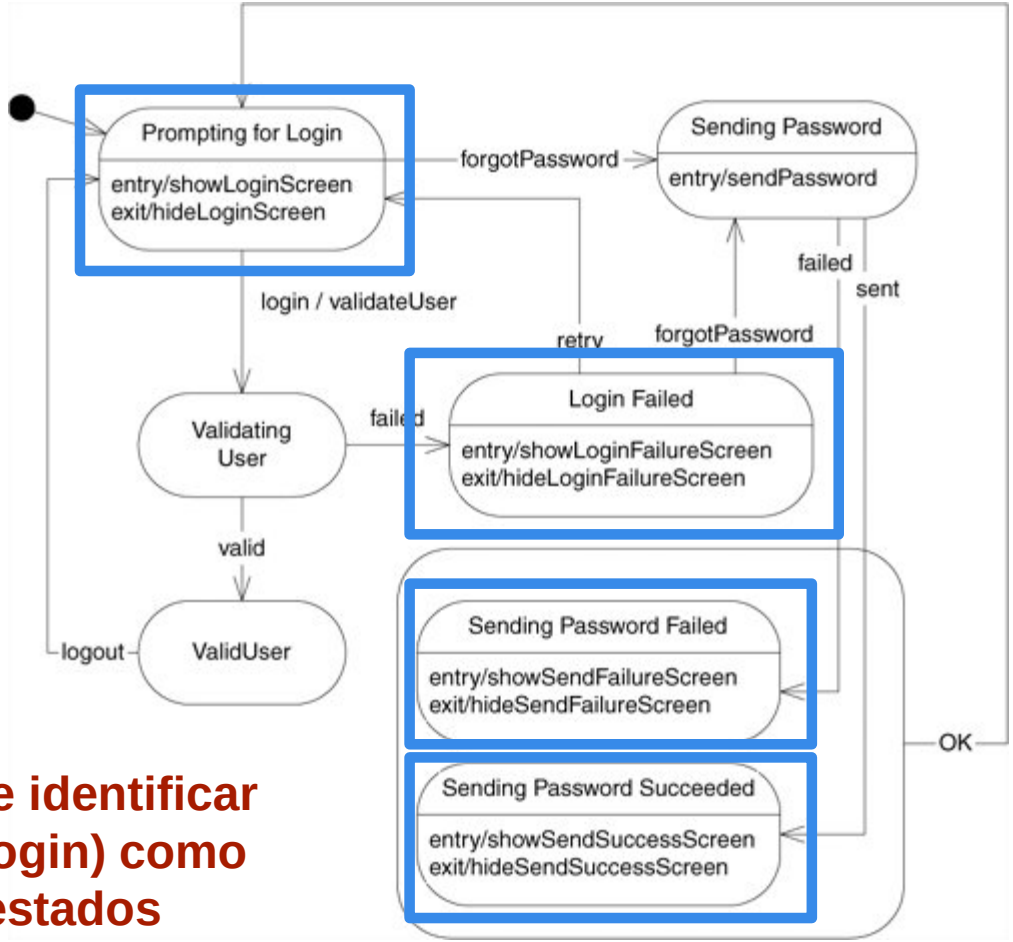
¿cómo podemos construir una máquina de estados que gestione este flujo?

Máquina de estados

Un estado se representa mediante de un rectángulo con las esquinas redondeadas

- Puede tener de forma opcional uno o más compartimientos.
- Nombre
- Transiciones internas

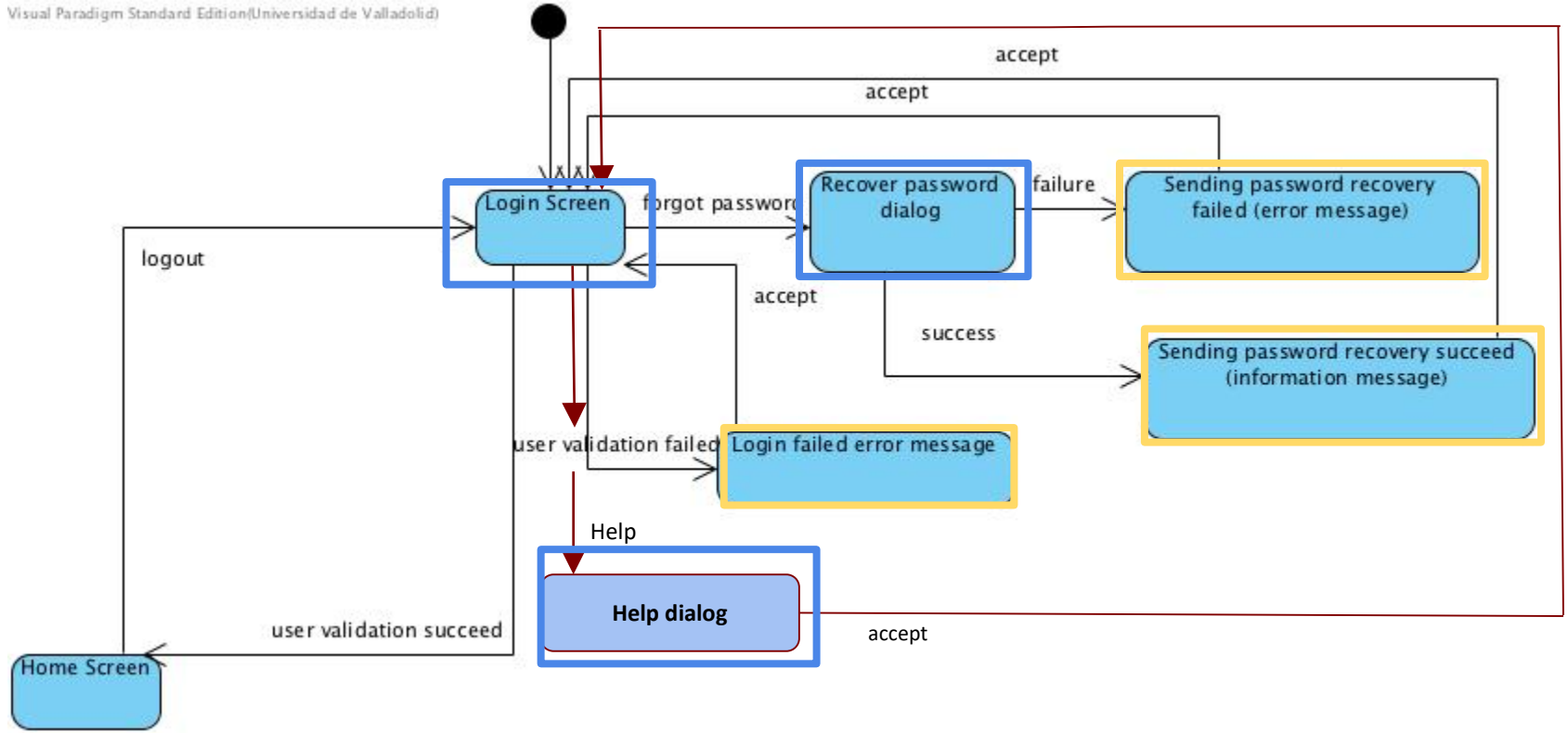




En azul: Estados que “necesitan” una ventana

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

Login desde el punto de vista de interfaz



Máquina de estados

Gestiona la interrelación de las varias ventanas que responden a la misma interacción (caso de uso) con el usuario.

En nuestra interacción de Autentificación, tres ventanas: inicial, ayuda y recuperación de clave.

Main

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

        public static void main(String args[]) {
                loginStateMachine = new LoginStateMachine();
        }
        /*
        * acceso a la máquina de estado
        */
        public static LoginStateMachine getStateMachineLogin()
        {
                return loginStateMachine;
        }
}
```

La máquina de estados se declara como una variable de clase. Es una referencia a la máquina y es única.

La clase Main pone en marcha la máquina de estados LoginStateMachine.

Hay que crear métodos get para obtener la referencia a la máquina

Máquina de estados: LoginStateMachine.

Autentificación

El método	lanza la “ventana”
LoginStateMachine	LoginWindow
help	HelpWindow
recoverPassword	PasswordRecoveryWindow

currentState: Control de qué ventana está activa y visible en cada momento.
Es una variable de clase de tipo JFrame: *private*

LoginStateMachine (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
```

LoginStateMachine (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
```

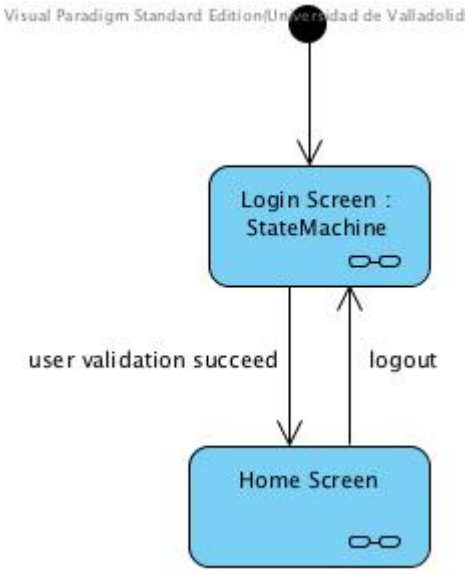
```
    public  void close() {
                currentState.setVisible(false); // si se desea
ocultar
                currentState.dispose();   // si se desea destruir
        } //close
} //Login class
```

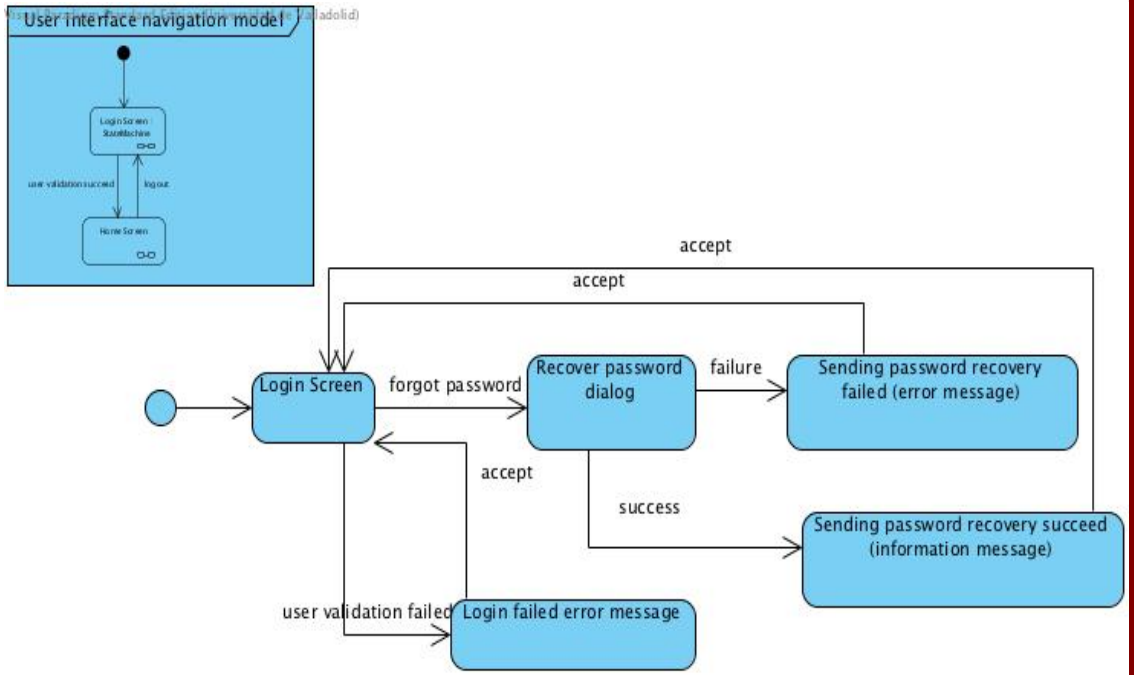
LoginStateMachine (III)

… acabar el ejemplo…

…. Ver los controladores de las vistas….

Y si hay varias máquinas de estados





Main

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

        public static void main(String args[]) {
                loginStateMachine = new LoginStateMachine();
        }
        public static LoginStateMachine getStateMachineLogin() {
                return loginStateMachine;
        }
        public static HomeStateMachine getStateMachineHome() {
                return homeStateMachine;
        }
        public static void loginSucceed() {
                loginStateMachine.close();
                homeStateMachine = new HomeStateMachine();
        }
}
```

La primera máquina y principal (Main) tiene dos submáquinas (LoginStateMachine, HomeStateMachine), y un cambio de estado de Login a Home (loginSucceed)

(Propuesta: añadir un cambio de estado de Home a Login cuando se haga logout en Home)

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

        public static void main(String args[]) {
                loginStateMachine = new LoginStateMachine();
        }
        public static LoginStateMachine getStateMachineLogin() {
                return loginStateMachine;
        }
        public static HomeStateMachine getStateMachineHome() {
                return homeStateMachine;
        }
        public static void loginSucceed() {
                loginStateMachine.close();
                homeStateMachine = new HomeStateMachine();
        }
}
```

Atributos de CLASE. La referencia a la máquina de estados es única.

Hay que crear métodos get para obtener la referencia a la máquina

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)

¿Dónde está el Modelo?

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

        private Modelo model; /* aunque no se puede asegurar, dependerá de la aplicación */

        public static void main(String args[]) {
                /* Creo el modelo*/
                loginStateMachine = new LoginStateMachine(model);
        }
        public static LoginStateMachine getStateMachineLogin() {
                return loginStateMachine;
        }
        public static HomeStateMachine getStateMachineHome() {
                return homeStateMachine;
        }
        public static void loginSucceed() {
                loginStateMachine.close();
                homeStateMachine = new HomeStateMachine(model);
        } }
```

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

Visual Paradigm (2019) Visual Paradigm Users' Guide. Wireframes.
http://www.visual-paradigm.com/support/documents/vpuserguide/2822_wireframe.html

Visual Paradigm (2019) What is a wireframe state? http://www.visual-paradigm.com/support/documents/vpuserguide/2822/2613/83713_whatisawiref.html

ClaudioLassala(n.d.)StateMachines and GUI interaction
Part I : http://lassala.net/2008/02/05/state-machines-and-gui-interaction-part-i/
Part II: http://lassala.net/2008/02/19/state-machines-and-gui-interaction-part-ii/

BrianO'Byrne(2002) StateMachines & UserInterfaces
http://www.drdobbs.com/jvm/state-machines-user-interfaces/184405248