*2015/2016*


# Memoria

*Eduardo Vela Galindo*

*Fernando Rivilla Bravo*

*Iván María Paredes*

*Rodrigo de Miguel González*

*Rubén Barrado González*

*Tomás Muñoz Testón*


*Universidad Complutense*

***Facultad de Informática***

# ÍNDICE

# 1. introducción.

La memoria del proyecto School Mannager Software (SMS) contiene una descripción breve de éste, la documentación y el trabajo realizado durante estos 2 meses por los integrantes del grupo. Vamos a detallar uno a uno los documentos que vamos a entregar, a parte de este documento, que son:

- Patrones de Diseño
- Especificación de requisitos software (SRS)
- Diagramas de UML (Proyecto IBM Rational Software Architect for WebSphere)

# 2. Patrones de diseño.

Nuestro “ActionListener” hace una función similar al patrón de Controlador Frontal, por lo tanto no tenemos presente este patrón como tal.

### 2.1. Controldor de aplicación.

Su función es centralizar y modularizar la gestión de acciones, con el fin de mejorar la extensibilidad del manejo de peticiones, usando el patrón “Command” para ello.

### 2.2. Contexto.

Lo utilizamos para evitar utilizar la información específica del sistema fuera de su contexto y por tanto evitar así un fuerte acoplamiento.

### 2.3. Dispatcher view.

Se desea que una vista maneje una petición y genere una respuesta, gestionando una cantidad limitada de procesamiento de negocio.

La vista necesita servicios limitados de negocio o acceso a datos, aprovechando marcos y librerías.

### 2.4. Transfer.

Para que los datos fluyan entre la capa de negocio y otras capas, y para encapsular los datos en objetos con el fin de ser independientes de otras capas .

### 2.5. Singleton.

Lo utilizamos para garantizar que una clase tenga solo una única instancia de la misma, normalmente lo utilizamos en el controlador, en las factorías y sobre todo en las ventanas o vistas

### 2.6. Servicio de aplicación (SA).

Este patrón hace de intermediario entre la capa de servicios (integración) y la capa de negocio, permitiendo la encapsulación de la capa de negocio fuera de esta misma capa además permite reutilización de código

### 2.7. Abstract Factory.

Nos proporciona interfaces para crear familias de objetos relacionados sin especificar clases concretas aislando estas de sus clientes para conseguir una mayor consistencia entre productos

### 2.8. Almacén del dominio.

Nos permite separar la persistencia del modelo de objetos cuyo objetivo principal es reducir la complejidad mejorando la estrategia de persistencia. Está estrategia desacopla los objetos del negocio del almacén nombrado así resuelven, entre otras, los problemas de persistencia, carga dinámica, gestión de transacción y concurrencia.

### 2.9. Objetos del negocio

Para evitar que la aplicación pueda permitir un acceso directo a la capa de datos (por ejemplo, un DAO), se tiene un modelo conceptual conteniendo objetos del negocio interrelacionados y compuestos. Además, se tiene un modelo conceptual con lógica de negocio, reglas de validación y reglas de negocio sofisticadas. De esta

manera, estos objetos encapsulan y manejan datos del negocio, comportamiento y persistencia.

## 3. Especificación de requisitos del software.

Describe las acciones de nuestro programa, y las funcionalidades de las que dispone el usuario que va a usar nuestra herramienta, que le facilitara la gestión del centro educativo desde una interfaz gráfica muy intuitiva.

La implementación del programa se realizara en lenguaje Java y no existe la necesidad de identificarse antes de entrar a la aplicación, accediendo directamente en modo Administrador.

## 4. Planificación temporal.

Nuestra planificación se basa en tres fases; inicial, de diseño y de implementación.

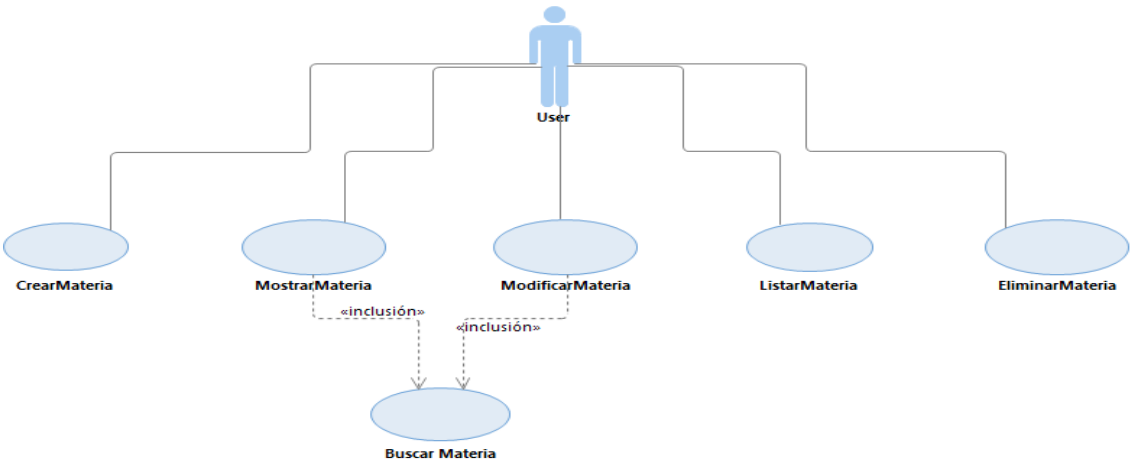
En la primera se realizó la planificación temporal y el análisis de requisitos. En la segunda se elaboró documentación, como el diseño UML de los módulos más sencillos de la aplicación para una vez validados con el profesor y pasados al software IBM Rational, comenzamos la implementación del primer módulo, seguidamente pasados al software IBM Rational los diagramas del segundo módulo comenzamos su implementación y de igual manera para nuestro 3 modulo, por tanto, el equipo ha trabajado en todos los aspectos del proyecto; en la última fase se revisaron errores, realizando pruebas intensivas con el fin de encontrar posibles fallos pasados por alto durante la fase de desarrollo.

# 5. Diagramas UML.

Aquí describimos todos los diagramas que vamos a entregar

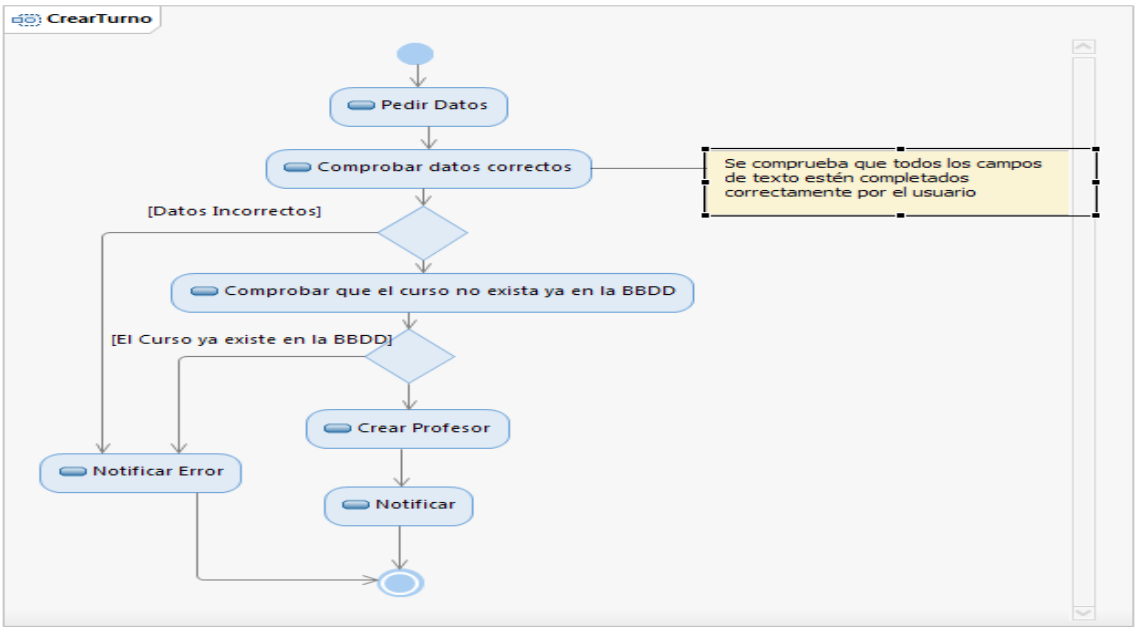
## 5.1. Diagramas de casos de uso.

Contiene todos los casos de uso del programa, las relaciones entre ellos y los datos o precondiciones que necesitan para ejecutarse.



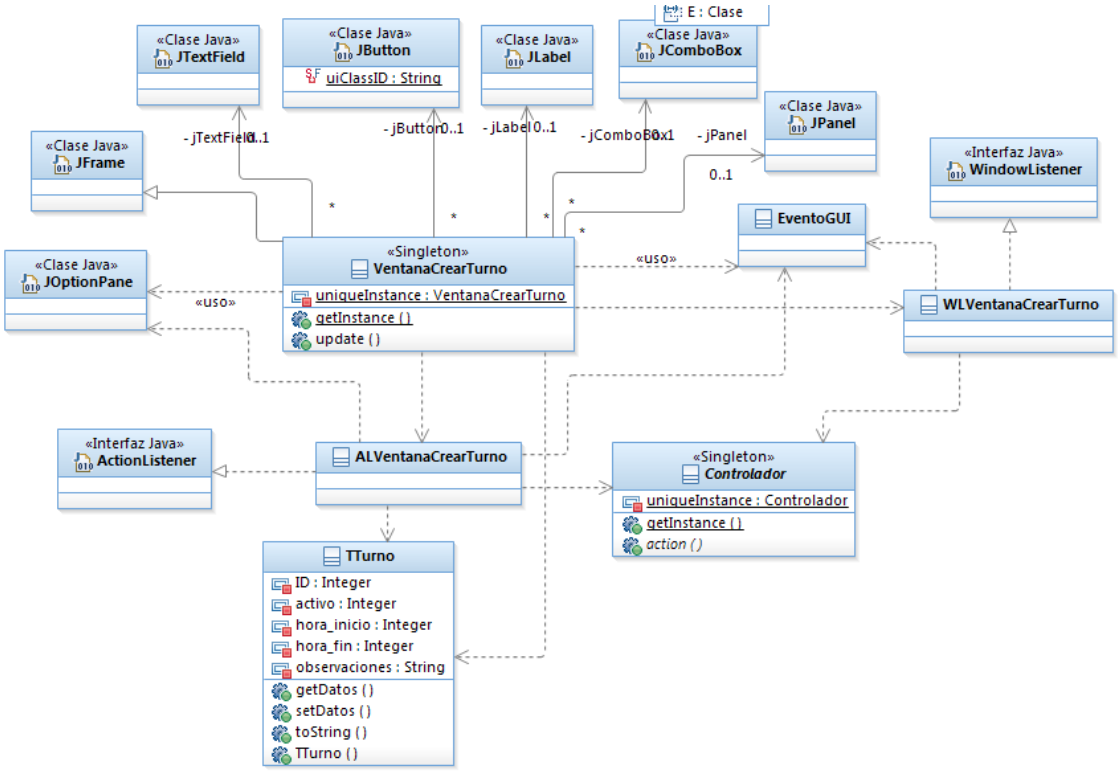

## 5.2. Diagramas de actividades.

Muestra los pasos y las ejecuciones alternativas debidas a fallos para todas las actividades del programa desde sus estados iniciales a los finales, además de incluir el flujo de eventos.



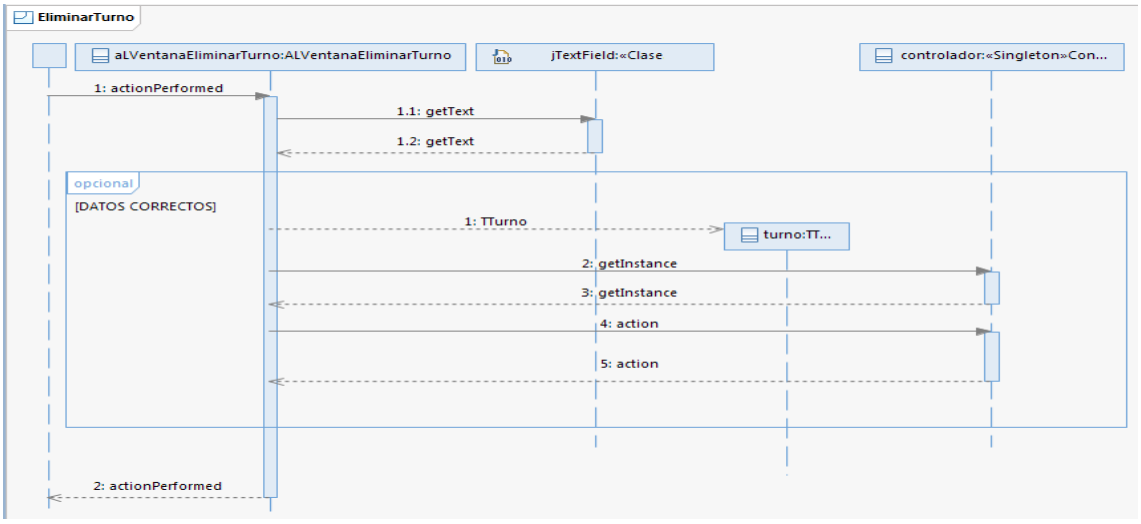
## 5.3. Diagramas de clases.

Contiene todas las clases, sus relaciones, atributos y funciones, la forma de organización está según los paquetes de nuestro programa, cada diagrama de clase tiene como raíz la clase más importante de ese paquete.




## 5.4. Diagramas de secuencia.

En ellos se pueden observar las interacciones entre diversos métodos y objetos.

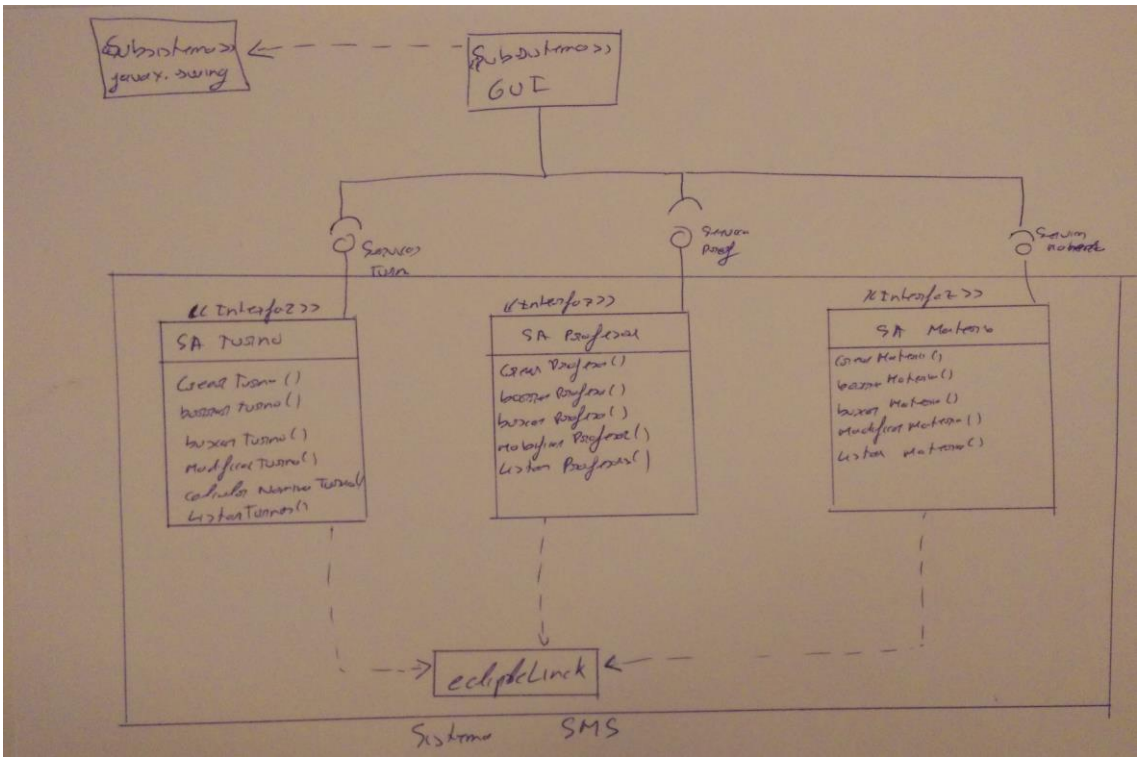


### 5.5. Diagramas de despliegue

Nuestro diagrama es muy sencillo debido a que empleamos únicamente el uso de una interfaz gráfica y una conexión de base de datos interna.

### 5.6. Diagrama de componentes

No hemos sido capaces de hacer el diagrama correspondiente en el IDE IBM RSA. Adjuntamos a papel dicho diagrama.




## 6. Problemas encontrados en el desarrollo.

### 6.1. Referencias en los diagramas.

Al comienzo del desarrollo del proyecto comenzamos a tener problemas cuando intentábamos sincronizar la parte de modelado en el IDE IBM RSA. El problema venía por una configuración a la hora de crear las dependencias entre clases ya que nos buscaba en todos los proyectos guardados en el espacio de trabajo de

cada componente del equipo. Por tanto, cuando se sincronizaban los diagramas no se visualizaban correctamente. La solución fue importar bibliotecas comunes al proyecto y usar todos las mismas.

### 6.2. JPA en IBM.

En un momento concreto del proyecto necesitábamos crear una relación M – N de dos entidades. Para que esta operación se realizará correctamente necesitábamos señalizar a las entidades con la palabra reservada @MapsId la cual no funcionaba correctamente en el IDE IBM RSA. La solución optada por todo el equipo fue migrar la parte de programación del proyecto a otro IDE, en este caso Eclipse.