

APLICATIVO WEB PARA LA CORRECCIÓN AUTOMÁTICA DE MODELOS UML/OCL

MEMORIA

UNIVERSIDAD OBERTA DE CATALUNYA

**PROYECTO FINAL DE CARRERA DE LOS ESTUDIOS DE
INGENIERIA INFORMÁTICA**

**Alumno : Pedro Reverte Morales
Consultor: Xavier Ferró García
25 de junio de 2008**

RESUMEN

Aplicativo web para la corrección automática de modelos UML/OCL.

El Proyecto de Fin de Carrera (PFC) es la culminación de los estudios de Ingeniería Informática y como tal es un trabajo de síntesis de los conocimientos adquiridos en la misma. El presente proyecto está centrado en la tecnología Java y en la arquitectura J2EE que se han constituido en un estándar en el mundo de la industria para el desarrollo de aplicaciones empresariales distribuidas en Internet. Se pretende profundizar en estas tecnologías mediante el desarrollo de un trabajo práctico, un aplicativo web para la corrección automática de modelos de clases UML/OCL, que hará uso de una herramienta desarrollada por el grupo de investigación GRES-UOC: *UMLtoCSP* y que permitirá a los estudiantes de un entorno virtual mejorar sus habilidades de modelización de datos (análisis). El aplicativo se ha desarrollado siguiendo el modelo arquitectónico más empleado para este tipo de aplicaciones: cliente/servidor estructurado en capas y se apoya en uno de los frameworks de código abierto más utilizados en los últimos años como alternativa al modelo de Enterprise Javabeans, *Spring*. Para la persistencia se utiliza el gestor de BBDD *MySQL* mediante un framework de mapeo objeto-relacional (ORM) *Hibernate*. Finalmente la integración con la herramienta *UMLtoCSP* se realiza mediante un Web Service, tecnología en auge y que aporta gran independencia entre la aplicación que usa el servicio y el propio servicio que podrá ser utilizado por otras aplicaciones. Todo ello junto con el uso de patrones arquitectónicos y de diseño permite alcanzar un conocimiento amplio de algunas de las tecnologías utilizadas para desarrollos en la plataforma J2EE y obtener como conclusión un aplicativo funcional que puede servir como punto de partida para su uso en un entorno virtual real como el de la UOC.

INDICES

Índice de contenidos.

INDICES	3
Índice de contenidos.	3
Índice de figuras.	4
INTRODUCCIÓN.....	6
Contexto.	6
El proyecto.....	6
Objetivos.....	7
Alcance del proyecto.	7
Enfoque y método seguido.	8
Plan de trabajo.	8
Fase inicial.....	9
Análisis	9
Diseño	9
Implementación.....	9
Fase Final.....	9
Planificación.	10
Calendario.....	10
Diagrama de Gantt.....	11
Productos obtenidos.....	12
Descripción del resto de capítulos.	12
1 ANÁLISIS	13
1.1 Casos de uso.	13
1.1.1 Identificación de actores.	13
1.1.2 Diagrama de casos de uso.....	14
1.1.3 Descripción de los casos de uso.	14
1.2 Especificación de las clases de análisis.	20
1.2.1 Identificación de las clases de entidad.....	20
1.2.2 Especificación de los atributos de las clases de entidad.	21
1.2.3 Diagrama de clases de entidad	21
1.2.4 Identificación de las clases de frontera y control y de las operaciones. ..	22
1.2.5 Especificación formal de los casos de uso.	29
2 DISEÑO	33
2.1 Diseño de la arquitectura de la aplicación.	33
2.1.1 Diseño de alto nivel.	33
2.1.2 Diseño arquitectónico detallado.	34
2.1.3 Integración con la herramienta UMLtoCSP.	42
2.1.4 Diagrama de despliegue.	43
2.1.5 Clases estáticas de diseño.	45
2.2 Diseño de la persistencia.	47
2.3 Diseño de la interfaz de usuario.	49
3 IMPLEMENTACIÓN	52

3.1	Decisiones de implementación.....	52
3.2	Software utilizado.....	54
3.3	Despliegue y configuración.....	54
3.3.1	Configuración del aplicativo UMLtoCSPWeb.....	55
3.3.2	Configuración del Web Service UMLtoCSPWS.....	56
3.3.3	Configuración de un entorno para pruebas.....	56
VALORACION FINAL.....		58
Conclusiones.....		58
Mejoras para futuras versiones.....		58
GLOSARIO.....		59
BIBLIOGRAFIA.....		61

Índice de figuras.

Fig. 1	Herencia de actores.....	13
Fig. 2	Diagrama de casos de uso.....	14
Fig. 3	Diagrama de clases de entidad.....	21
Fig. 4	Caso de uso <i>login</i>	22
Fig. 5	Caso de uso <i>verificar modelo</i>	22
Fig. 6	Caso de uso <i>cambiar password</i>	23
Fig. 7	Caso de uso <i>consultar ejercicios</i>	23
Fig. 8	Caso de uso <i>ver enunciado</i>	23
Fig. 9	Caso de uso <i>resolver ejercicio</i>	24
Fig. 10	Caso de uso <i>obtener solucion</i>	24
Fig. 11	Caso de uso <i>crear colección</i>	24
Fig. 12	Caso de uso <i>consultar colecciones</i>	25
Fig. 13	Caso de uso <i>eliminar colección</i>	25
Fig. 14	Caso de uso <i>modificar colección</i>	25
Fig. 15	Caso de uso <i>crear ejercicio</i>	26
Fig. 16	Caso de uso <i>eliminar ejercicio</i>	26
Fig. 17	Caso de uso <i>modificar ejercicio</i>	26
Fig. 18	Caso de uso <i>alta usuario</i>	27
Fig. 19	Caso de uso <i>alta usuarios múltiple</i>	27
Fig. 20	Caso de uso <i>baja usuario</i>	27
Fig. 21	Caso de uso <i>modificar usuario</i>	28
Fig. 22	Caso de uso <i>buscar usuario</i>	28
Fig. 23	Diagrama de secuencia caso de uso <i>login</i>	29
Fig. 24	Diagrama de secuencia caso de uso <i>resolver ejercicio</i>	29
Fig. 25	Diagrama de secuencia caso de uso <i>consultar ejercicios</i>	30
Fig. 26	Diagrama de secuencia caso de uso <i>obtener solución</i>	30
Fig. 27	Diagrama de secuencia caso de uso <i>crear ejercicio</i>	31
Fig. 28	Diagrama de secuencia caso de uso <i>ver enunciado</i>	31
Fig. 29	Diagrama de secuencia caso de uso <i>cambiar password</i>	32
Fig. 30	Diagrama de secuencia caso de uso <i>eliminar colección</i>	32
Fig. 31	Diagrama de secuencia caso de uso <i>alta usuario</i>	32
Fig. 32	Diagrama de diseño de alto nivel.....	33

Fig. 33 Diagrama de diseño arquitectónico detallado	34
Fig. 34 Modelo MVC-2.....	35
Fig. 35 Diagrama del componente <i>autenticaciónUsuarios</i>	36
Fig. 36 Diagrama del componente <i>presentaciónUsuarios</i>	36
Fig. 37 Diagrama del componente <i>presentaciónEjercicios</i>	37
Fig. 38 Diagrama del componente <i>presentaciónValidador</i>	37
Fig. 39 Patrón Facade	38
Fig. 40 Diagrama del componente <i>NegocioUsuarios</i>	38
Fig. 41 Diagrama del componente <i>NegocioEjercicios</i>	38
Fig. 42 Patrón proxy	39
Fig. 43 Diagrama del componente <i>NegocioValidador</i>	39
Fig. 44 Diagrama del componente <i>NegocioRepositorio</i>	40
Fig. 45 Patrón DAO	40
Fig. 46 Diagrama del componente <i>integraciónUsuarios</i>	41
Fig. 47 Diagrama del componente <i>integraciónEjercicios</i>	41
Fig. 48 Diagrama del componente <i>integraciónRepositorio</i>	41
Fig. 49 Diagrama del Web Service.....	43
Fig. 50 Diagrama de despliegue	44
Fig. 51 Diagrama de clases estáticas de entidad.....	45
Fig. 52 Diagrama de clases estáticas de gestión e integración.....	46
Fig. 53 Diagrama entidad-relación.....	47
Fig. 54 Pantalla <i>Inicio</i>	49
Fig. 55 Pantalla <i>Colecciones estudiante</i>	49
Fig. 56 Pantalla <i>cambio password</i>	49
Fig. 57 Pantalla <i>Enviar Modelo</i>	49
Fig. 58 Pantalla <i>Colecciones profesor</i>	49
Fig. 59 Pantalla <i>ver enunciado</i>	49
Fig. 60 Pantalla <i>resolver ejercicio</i>	50
Fig. 61 Pantalla <i>propiedades modelo</i>	50
Fig. 62 Pantalla crear colección.....	50
Fig. 63 Pantalla <i>ejercicios estudiante</i>	50
Fig. 64 Pantalla <i>ver solucion ejercicio</i>	50
Fig. 65 Pantalla <i>resultado validación</i>	50
Fig. 66 Pantalla crear ejercicio	50
Fig. 67 Pantalla <i>ejercicios profesor</i>	50
Fig. 68 Pantalla <i>Login administrador</i>	51
Fig. 69 Pantalla <i>alta múltiple</i>	51
Fig. 70 Pantalla <i>inicial administrador</i>	51
Fig. 71 Pantalla <i>alta usuario</i>	51
Fig. 72 Framework Spring.....	53

INTRODUCCIÓN

Contexto.

Dentro de todo proceso de desarrollo de software, la etapa de modelización de datos (análisis) es una de las fases más importantes. En esta etapa los estudiantes aprenden a definir la información del dominio que la aplicación necesitará para llevar a cabo sus funciones. Esta información se expresa mediante una serie de diagramas (actualmente utilizando el lenguaje UML) más un conjunto de expresiones textuales (por ejemplo en OCL) que los complementan.

La modelización no es un proceso mecánico, sino más bien creativo y donde es muy importante que los alumnos puedan practicar mucho, recibiendo un feedback constante que los permita evaluar su progreso. Esto es especialmente necesario para los alumnos de la UOC, donde por las propias características de una universidad virtual, no existe la opción de que el profesor consultor proporcione un feedback inmediato, constante e individual a los estudiantes.

Es por este motivo que resulta oportuna la introducción de un entorno virtual de autoaprendizaje que permita a los estudiantes mejorar sus habilidades de modelización, una pieza clave en su capacitación como profesionales de la informática adaptados a las demandas de la sociedad actual. Además, este tipo de entornos serán imprescindibles dentro del nuevo Espacio Europeo de Educación Superior (EEES).

El proyecto.

El presente proyecto pretende desarrollar un aplicativo web (aplicación distribuida) que pueda ser utilizado por los estudiantes de la UOC para mejorar su rendimiento en las asignaturas de ingeniería del software. Para ello permitirá, haciendo uso de una herramienta ([UMLtoCSP](#)) desarrollada dentro del grupo de investigación GRES-UOC, comprobar la corrección de modelos UML/OCL (diagramas de clases UML complementados por restricciones OCL) en el sentido de asegurar que cumplen con unos criterios de calidad básicos.

Objetivos.

La motivación de este proyecto es doble:

- El primer objetivo es el estudio y asimilación de las técnicas de desarrollo y producción basadas en la plataforma J2EE.

La tendencia actual en el ámbito del desarrollo de programas empresariales es la de proporcionar marcos de trabajo multicapa destinados a obtener aplicaciones distribuidas seguras, escalables y portables. A ese efecto, Sun Microsystems creó la arquitectura Java 2 Enterprise Edition (J2EE) que se ha convertido en un estándar de facto para esos desarrollos. El presente proyecto permitirá al autor el estudio de tecnologías concretas relacionadas con el mundo J2EE que son ampliamente utilizadas en la empresa: Servlets, JSPs, EJBs, JAAS, JDBC, SGBDs relacionales etc.

- El segundo objetivo es el desarrollo de un ejemplo típico de aplicación distribuida (aplicativo web para uso a través de Internet) que sirva como base para el mencionado estudio de la plataforma J2EE y al mismo tiempo tenga una utilidad real dentro de un entorno virtual como es el de la educación a distancia.

La aplicación permitirá a los usuarios comprobar la validez de ciertas propiedades (satisfactibilidad fuerte o débil, liveness y ausencia de contradicciones o subsunciones en las restricciones) de un modelo de clases UML. Además se proporcionará un servicio de administración de los usuarios de la aplicación.

El aplicativo constará de una parte servidora implementada en un servidor central y un cliente ligero correspondiente a los usuarios que accederán desde Internet con cualquier navegador convencional.

Alcance del proyecto.

El resultado final del proyecto será un aplicativo web diseñado para su aplicación en un entorno distribuido cuya funcionalidad a grandes rasgos será la siguiente:

- Acceso anónimo. Usuarios anónimos podrán acceder al sistema y comprobar la validez de un modelo UML/OCL. Para ello deberán suministrar un archivo xmi con el modelo de clases UML y opcionalmente un archivo de texto con las restricciones OCL, y seleccionar una propiedad a comprobar (de un conjunto predefinido). Estos elementos se pasarán a la herramienta UMLtoCSP para su análisis y se presentará al usuario el resultado de la comprobación.
- Acceso controlado. Existirán 3 roles a la hora de acceder: Administrador, Profesor y Estudiante.

- Administrador. Será el encargado de dar de alta en el sistema a los participantes con los roles de profesor y estudiante. Los datos a utilizar para ello serán: un nombre, un login y un password. Los estudiantes se podrán dar de alta de forma múltiple a partir de un archivo de texto separado por comas que contenga una lista de nombres y sus logines correspondientes. Cada usuario dado de alta en el sistema recibirá un email de notificación.
- Profesor. Podrá definir colecciones de ejercicios. Cada colección está formada por un conjunto x de ejercicios y cada ejercicio tiene un número de orden dentro de la colección. Un ejercicio consiste en un enunciado textual más dos archivos que representan la solución (un modelo UML y opcionalmente un archivo de restricciones OCL). Podrá cambiar su password.
- Estudiante. Podrá resolver los ejercicios predefinidos. Para ello seleccionará una colección y un ejercicio dentro de la colección y se le presentará el enunciado correspondiente. El estudiante podrá aportar su solución que será comprobada con el herramienta UMLtoCSP y una vez realizado esto podrá acceder a la solución oficial del ejercicio. El estudiante podrá cambiar su password.
- Repositorio. Cada vez que se envíe un modelo a la herramienta UMLtoCSP para comprobar su validez se guardará en una BBDD junto con el usuario que ha enviado el modelo, la fecha, la propiedad concreta a verificar y un indicador de si ha resultado exitosa o no.

Enfoque y método seguido.




Dada la naturaleza del presente proyecto, condicionado a un plazo de entrega muy estricto y con un alcance muy definido desde el inicio, se utilizará para su desarrollo un ciclo de vida clásico basado en las etapas de definición de requisitos, análisis, diseño, implementación y pruebas.

Plan de trabajo.

La planificación del proyecto viene condicionada por las fechas de los entregables y se ha definido en 5 fases.


Fase inicial

Periodo 28 febrero a 14 de marzo

-  Estudio de la problemática a realizar.
-  Localización del software necesario para la realización del proyecto.
-  Desarrollo y entrega del documento Plan de Trabajo.



Análisis

Periodo 14 marzo a 29 de marzo

-  Especificación de la aplicación a desarrollar.






Diseño

Periodo 29 marzo a 14 de abril

-  Diseño de la arquitectura, persistencia e interfaz de usuario de la aplicación.
-  Entrega del documento de análisis y diseño (PAC2).




Implementación

Periodo 14 abril a 11 de junio

-  Estudio de la plataforma J2EE.
-  Instalación del entorno de trabajo.
-  Implementación del aplicativo.
-  Pruebas de funcionamiento
-  Entrega de la versión Alpha (PAC3).

Fase Final

Periodo 11 junio a 25 de junio

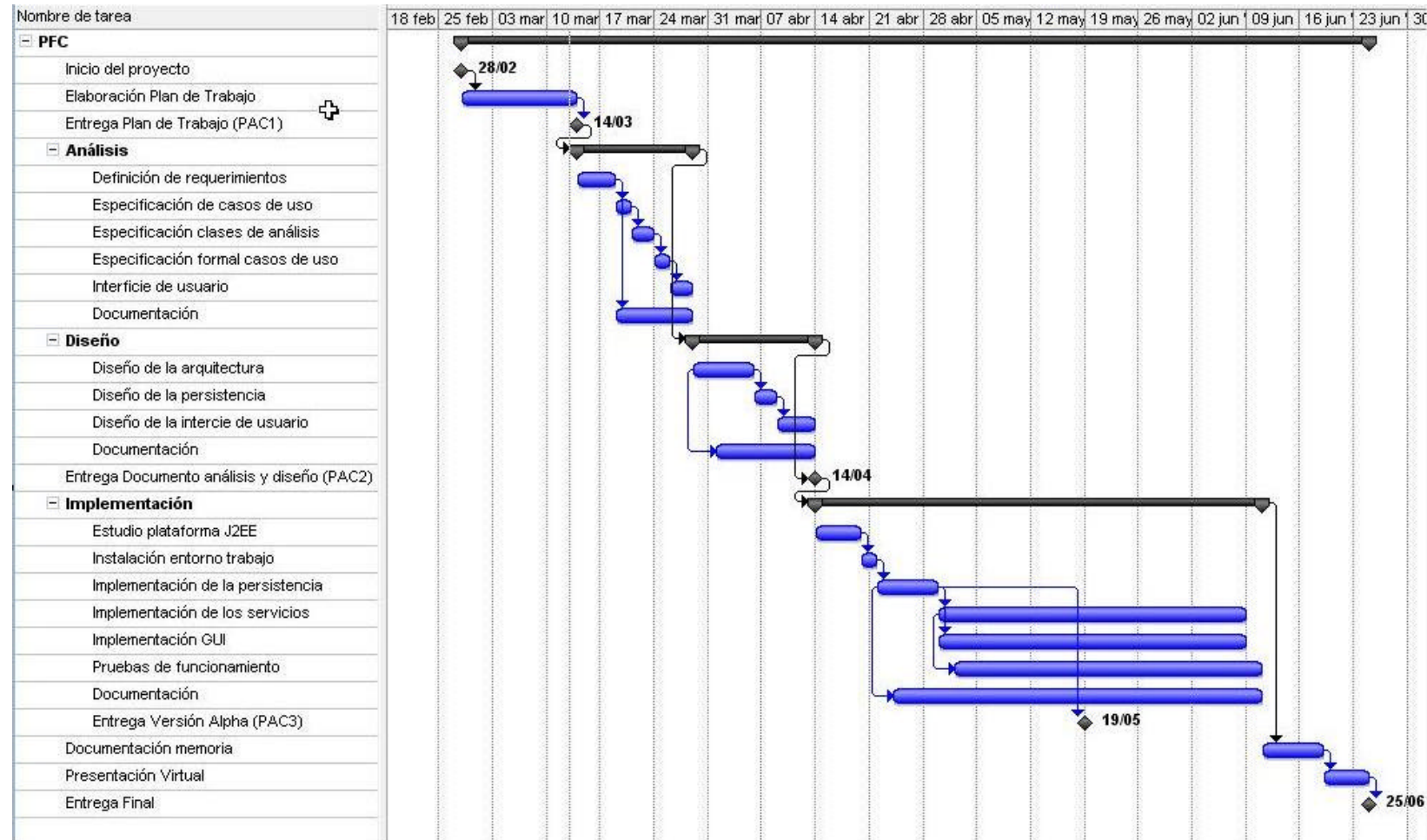
-  Redacción final de la memoria.
-  Realización de la presentación virtual.
-  Entrega final.

Planificación.

Calendario.

	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras
1	PFC	118 días	jue 28/02/08	mié 25/06/08	
2	Inicio del proyecto	0 días	jue 28/02/08	jue 28/02/08	
3	Elaboración Plan de Trabajo	15 días	jue 28/02/08	vie 14/03/08	2
4	Entrega Plan de Trabajo (PAC1)	0 días	vie 14/03/08	vie 14/03/08	3
5	Análisis	15 días	vie 14/03/08	sáb 29/03/08	4
6	Definición de requerimientos	5 días	vie 14/03/08	mié 19/03/08	
7	Especificación de casos de uso	2 días	mié 19/03/08	vie 21/03/08	6
8	Especificación clases de análisis	3 días	vie 21/03/08	lun 24/03/08	7
9	Especificación formal casos de uso	2 días	lun 24/03/08	mié 26/03/08	8
10	Interficie de usuario	3 días	mié 26/03/08	sáb 29/03/08	9
11	Documentación	10 días	mié 19/03/08	sáb 29/03/08	6
12	Diseño	16 días	sáb 29/03/08	lun 14/04/08	5
13	Diseño de la arquitectura	8 días	sáb 29/03/08	dom 06/04/08	
14	Diseño de la persistencia	3 días	dom 06/04/08	mié 09/04/08	13
15	Diseño de la intercie de usuario	5 días	mié 09/04/08	lun 14/04/08	14
16	Documentación	13 días	mar 01/04/08	lun 14/04/08	13CC+3 días
17	Entrega Documento análisis y diseño (PAC2)	0 días	lun 14/04/08	lun 14/04/08	12
18	Implementación	58 días	lun 14/04/08	mié 11/06/08	17
19	Estudio plataforma J2EE	6 días	lun 14/04/08	dom 20/04/08	
20	Instalación entorno trabajo	2 días	dom 20/04/08	mar 22/04/08	19
21	Implementación de la persistencia	8 días	mar 22/04/08	mié 30/04/08	20
22	Implementación de los servicios	40 días	mié 30/04/08	lun 09/06/08	21
23	Implementación GUI	40 días	mié 30/04/08	lun 09/06/08	21
24	Pruebas de funcionamiento	40 días	vie 02/05/08	mié 11/06/08	22CC+2 días
25	Documentación	48 días	jue 24/04/08	mié 11/06/08	21CC+2 días
26	Entrega Versión Alpha (PAC3)	0 días	lun 19/05/08	lun 19/05/08	21
27	Documentación memoria	8 días	mié 11/06/08	jue 19/06/08	18
28	Presentación Virtual	6 días	jue 19/06/08	mié 25/06/08	27
29	Entrega Final	0 días	mié 25/06/08	mié 25/06/08	28

Diagrama de Gantt.



Productos obtenidos.

El producto obtenido es el aplicativo *UMLtoCSPWeb* desarrollado según los estándares J2EE.

Esta aplicación ha generado los siguientes documentos y archivos:

- PFC_prevertem.war. Contiene el aplicativo web listo para desplegar en un servidor web como Tomcat.
- UMLtoCSPWS.war Implementación del Web Service desarrollado para la herramienta UMLtoCSP que deberá ser desplegado en un servidor web en la misma máquina en la que resida dicha herramienta.
- Umltocspweb_creación.sql. Script de creación de la base de datos necesaria para el funcionamiento del aplicativo. Se añade además un usuario administrador con login: admin y password: admin
- Usuarios_pruebas.csv y umltocspweb_pruebas.sql. Necesarios para la creación de un entorno de pruebas con datos definidos.
- PFC_prevertem.doc. Memoria del proyecto.

Descripción del resto de capítulos.

El resto de capítulos de la memoria proporcionan una visión detallada del proceso seguido en el desarrollo del proyecto y son los siguientes:

- Análisis. Este capítulo describe las labores de análisis: identificación de actores, casos de uso, descripción de los casos de uso, identificación de las clases de entidad, frontera y control, diagramas de colaboración y secuencia de los principales casos de uso.
- Diseño. Se describe el diseño en capas de la arquitectura de la aplicación, con la identificación de los componentes a usar y los patrones arquitectónicos y de diseño, el diseño de la persistencia necesaria y el diseño de la interfaz de usuario.
- Implementación. Describe las decisiones de implementación adoptadas, con los frameworks empleados, el software utilizado para el desarrollo del proyecto y la forma de desplegar y configurar el aplicativo para su funcionamiento correcto.
- Valoración final. Se ofrecen las conclusiones obtenidas y las posibles mejoras para próximas versiones.

1 ANÁLISIS

1.1 Casos de uso.

1.1.1 Identificación de actores.

Los actores del sistema vienen dados por los diferentes perfiles descritos en los requisitos funcionales del sistema: *usuario*, *estudiante*, *profesor* y *administrador*.

El *usuario* representa cualquier usuario de la aplicación. No se ha identificado al sistema y tiene acceso únicamente a la parte pública no restringida. Sus funcionalidades son:

- Login
- Verificar modelo UML

El *estudiante* es un usuario identificado ante el sistema de la parte pública. Hereda de usuario y además tiene las siguientes funcionalidades:

- Navegar por las colecciones de ejercicios y ver los enunciados.
- Resolver un ejercicio concreto.
- Acceder a la solución oficial de los ejercicios que haya resuelto.
- Cambiar su password.

El *profesor* es el usuario identificado ante el sistema de la parte pública que tiene más privilegios. Hereda de estudiante y puede además:

- Crear una nueva colección.
- Eliminar/modificar una colección de la que sea propietario.
- Añadir/Eliminar/Modificar un ejercicio en una colección de su propiedad.

El *administrador* es el único usuario con acceso a la parte privada de la aplicación. Su única función será la gestión de usuarios de la aplicación.

En relación a los actores tenemos una herencia por especialización.



Fig. 1 Herencia de actores

4. El sistema valida el modelo con la herramienta UMLtoCLP y presenta la respuesta al usuario.
5. El sistema almacena el modelo validado en un repositorio.
Flujos alternativos
1. El sistema presenta un mensaje de error de archivo no encontrado.
2. El sistema presenta un mensaje de error de archivo no encontrado.
5. En caso de error en la validación no se almacena el modelo en el repositorio.

Caso de uso: “Cambiar password”
Descripción: Un usuario identificado cambia su password.
Papel dentro del trabajo del usuario: Caso de uso secundario.
Actores: estudiante, profesor, administrador
Casos de uso relacionados: <u>Login</u> .
Precondición: El usuario está dado de alta y ha sido identificado por el sistema.
Postcondición: El usuario ha establecido un nuevo password.
Flujo Principal
1. El usuario introduce su password actual.
2. El usuario introduce su nuevo password.
3. El usuario repite su nuevo password.
4. El sistema establece el nuevo password del usuario.
Flujos alternativos
0. Mensaje de error si el password es incorrecto.
3. Mensaje de error si las dos versiones del nuevo password no coinciden.

Caso de uso: “Consultar ejercicios”
Descripción: Un usuario consulta la lista de ejercicios de una colección.
Papel dentro del trabajo del usuario: Caso de uso secundario.
Actores: estudiante, profesor
Casos de uso relacionados: <u>Consultar colecciones</u> , <u>Resolver ejercicio</u> , <u>Ver enunciado</u> , <u>Obtener solución</u> , <u>Eliminar ejercicio</u> , <u>Modificar ejercicio</u> .
Precondición: El usuario está dado de alta, ha sido identificado por el sistema y ha seleccionado una colección de ejercicios.
Postcondición:
Flujo Principal
1. El usuario solicita la lista de ejercicios de una colección.
2. El sistema muestra la lista de ejercicios de la colección seleccionada.
Flujos alternativos
2. Mensaje de error si el usuario no ha seleccionado una colección.

Caso de uso: “Ver enunciado”
Descripción: Un usuario solicita ver el enunciado de un ejercicio.
Papel dentro del trabajo del usuario: Caso de uso principal.
Actores: estudiante, profesor
Casos de uso relacionados: <u>Consultar ejercicios</u> , <u>Resolver ejercicio</u> .
Precondición: El usuario está dado de alta, ha sido identificado por el sistema, ha seleccionado una colección de ejercicios y ha obtenido el listado de sus ejercicios.
Postcondición: El usuario ha obtenido el enunciado de un ejercicio.
Flujo Principal
1. El usuario selecciona un ejercicio para ver su enunciado.
2. El sistema presenta el enunciado del ejercicio.
Flujos alternativos
2. Mensaje de error si el usuario no ha seleccionado un ejercicio.

Caso de uso: “Resolver ejercicio”
Descripción: Un usuario solicita la opción de resolver un ejercicio.
Papel dentro del trabajo del usuario: Caso de uso principal.
Actores: estudiante, profesor
Casos de uso relacionados: <u>Consultar ejercicios</u> , <u>Ver enunciado</u> , <u>Verificar modelo</u> .

Precondición: El usuario está dado de alta, ha sido identificado por el sistema, ha seleccionado una colección de ejercicios y ha obtenido el listado de sus ejercicios.
Postcondición: El usuario ha verificado un modelo UML y el sistema ha marcado el ejercicio como resuelto por el usuario.
Flujo Principal
<ol style="list-style-type: none"> 1. El usuario selecciona un ejercicio de la colección para resolver. 2. El usuario procede a verificar su solución según el caso de uso <u>Verificar modelo</u>. 3. El sistema marca el ejercicio como resuelto por el usuario, si la solución es correcta.
Flujos alternativos
<ol style="list-style-type: none"> 2. Mensaje de error si el usuario no ha seleccionado un ejercicio. 3. En caso de error al verificar el modelo no se marca el ejercicio como resuelto.

Caso de uso: “Obtener solución”
Descripción: Un usuario solicita el modelo UML solución oficial de un ejercicio.
Papel dentro del trabajo del usuario: Caso de uso principal.
Actores: estudiante, profesor
Casos de uso relacionados: <u>Consultar ejercicios</u> , <u>Resolver ejercicio</u> .
Precondición: El usuario está dado de alta, ha sido identificado por el sistema, ha seleccionado una colección de ejercicios y ha obtenido el listado de sus ejercicios.
Postcondición: El usuario ha obtenido el modelo UML solución oficial de un ejercicio.
Flujo Principal
<ol style="list-style-type: none"> 1. El usuario selecciona un ejercicio de la colección para obtener su solución. 2. El sistema verifica que el ejercicio está marcado como resuelto por el usuario. 3. El sistema proporciona el modelo UML solución del ejercicio.
Flujos alternativos
<ol style="list-style-type: none"> 1. Mensaje de error si el usuario no ha seleccionado un ejercicio. 3. Mensaje de pendiente de resolución si el ejercicio no está marcado como resuelto por el usuario.

Caso de uso: “Crear colección”
Descripción: Un usuario de tipo profesor crea una colección de ejercicios nueva.
Papel dentro del trabajo del usuario: Caso de uso principal.
Actores: profesor
Casos de uso relacionados: <u>Modificar colección</u> , <u>Eliminar colección</u> .
Precondición: El usuario está dado de alta y ha sido identificado por el sistema con el tipo profesor.
Postcondición: Se ha creado una nueva colección de ejercicios.
Flujo Principal
<ol style="list-style-type: none"> 1. El usuario introduce el descriptivo para la nueva colección. 2. El sistema crea una nueva colección y asigna como propietario al usuario.
Flujos alternativos
<ol style="list-style-type: none"> 1. Mensaje de error si no se ha introducido ningún identificativo.

Caso de uso: “Consultar colecciones”
Descripción: Un usuario consulta la lista de colecciones.
Papel dentro del trabajo del usuario: Caso de uso secundario.
Actores: estudiante, profesor
Casos de uso relacionados: <u>Consultar colecciones</u> , <u>Eliminar colección</u> , <u>Modificar colección</u> , <u>Crear ejercicio</u> .
Precondición: El usuario está dado de alta y ha sido identificado por el sistema.
Postcondición: El usuario obtiene un listado de colecciones.
Flujo Principal
<ol style="list-style-type: none"> 1. El usuario solicita la lista de colecciones. 2. El sistema muestra la lista de colecciones.
Flujos alternativos

Caso de uso: “Eliminar colección”
Descripción: Un usuario elimina una colección. Papel dentro del trabajo del usuario: Caso de uso principal. Actores: profesor Casos de uso relacionados: Consultar colecciones , Crear colección . Precondición: El usuario está dado de alta, ha sido identificado por el sistema como profesor y ha seleccionado una colección de ejercicios. Postcondición: La colección seleccionada ha sido eliminada.
Flujo Principal
1. El usuario solicita eliminar una colección. 2. El sistema elimina la colección con sus ejercicios asociados.
Flujos alternativos
1. Mensaje de error si el usuario no ha seleccionado una colección.

Caso de uso: “Modificar colección”
Descripción: Un usuario modifica el descriptivo de una colección. Papel dentro del trabajo del usuario: Caso de uso principal. Actores: profesor Casos de uso relacionados: Consultar colecciones , Crear colección . Precondición: El usuario está dado de alta, ha sido identificado por el sistema como profesor y ha seleccionado una colección de ejercicios. Postcondición: El descriptivo de la colección seleccionada ha sido modificado.
Flujo Principal
1. El usuario solicita modificar una colección. 2. El usuario introduce el nuevo descriptivo de la colección. 3. El sistema modifica el descriptivo.
Flujos alternativos
1. Mensaje de error si el usuario no ha seleccionado una colección.

Caso de uso: “Crear ejercicio”
Descripción: Un usuario crea un nuevo ejercicio. Papel dentro del trabajo del usuario: Caso de uso principal. Actores: profesor Casos de uso relacionados: Consultar colecciones . Precondición: El usuario está dado de alta, ha sido identificado por el sistema como profesor y ha seleccionado una colección de ejercicios. Postcondición: Se ha añadido un nuevo ejercicio a la colección seleccionada.
Flujo Principal
1. El usuario solicita crear un nuevo ejercicio. 2. El sistema comprueba que el usuario es el propietario de la colección. 3. El usuario introduce el número de orden del nuevo ejercicio. 4. El usuario introduce el descriptivo y el enunciado del ejercicio. 5. El usuario introduce los archivos xmi y ocl (opcional) de la solución. 6. El sistema crea un nuevo ejercicio en la colección.
Flujos alternativos
1. Mensaje de error si el usuario no ha seleccionado una colección. 2. Mensaje de error si el usuario no es propietario de la colección y no puede modificarla. 4. Mensaje de error si el descriptivo o el enunciado están vacíos. 5. Mensaje de error si no se encuentra alguno de los archivos.

Caso de uso: “Eliminar ejercicio”
Descripción: Un usuario elimina un ejercicio de una colección. Papel dentro del trabajo del usuario: Caso de uso principal. Actores: profesor Casos de uso relacionados: Consultar colecciones , Crear colección , Consultar ejercicios . Precondición: El usuario está dado de alta, ha sido identificado por el sistema como profesor y ha seleccionado una colección de ejercicios y un ejercicio concreto. Postcondición: El ejercicio seleccionado ha sido eliminado.

Flujo Principal
<ol style="list-style-type: none"> 1. El usuario solicita eliminar un ejercicio. 2. El sistema comprueba que el usuario es el propietario de la colección. 3. El sistema elimina el ejercicio.
Flujos alternativos
<ol style="list-style-type: none"> 1. Mensaje de error si el usuario no ha seleccionado una colección. 2. Mensaje de error si el usuario no es el propietario de la colección.

Caso de uso: “Modificar ejercicio”
<p>Descripción: Un usuario modifica los datos de un ejercicio de una colección.</p> <p>Papel dentro del trabajo del usuario: Caso de uso principal.</p> <p>Actores: profesor</p> <p>Casos de uso relacionados: <u>Consultar colecciones</u>, <u>Consultar ejercicios</u>.</p> <p>Precondición: El usuario está dado de alta, ha sido identificado por el sistema como profesor y ha seleccionado una colección de ejercicios y un ejercicio concreto.</p> <p>Postcondición: Los datos del ejercicio seleccionado han sido modificados.</p>
Flujo Principal
<ol style="list-style-type: none"> 1. El usuario solicita modificar los datos de un ejercicio. 2. El sistema comprueba que el usuario es el propietario de la colección. 3. El usuario introduce los nuevos datos. 4. El sistema modifica los datos.
Flujos alternativos
<ol style="list-style-type: none"> 1. Mensaje de error si el usuario no ha seleccionado una colección. 2. Mensaje de error si el usuario no es el propietario de la colección. 3. Mensaje de error si alguno de los datos es incorrecto.

Caso de uso: “Alta usuario”
<p>Descripción: El administrador da de alta un nuevo usuario.</p> <p>Papel dentro del trabajo del usuario: Caso de uso principal.</p> <p>Actores: administrador</p> <p>Casos de uso relacionados: <u>Enviar notificación</u>, <u>Alta usuarios multiple</u>.</p> <p>Precondición: El administrador se ha identificado ante el sistema.</p> <p>Postcondición: Un nuevo usuario ha sido dado de alta.</p>
Flujo Principal
<ol style="list-style-type: none"> 1. El administrador solicita dar de alta un usuario. 2. El administrador introduce los datos del usuario: nombre, login, password, tipo (estudiante, profesor). 3. El sistema da de alta el nuevo usuario. 4. El sistema envía notificación de alta.
Flujos alternativos
<ol style="list-style-type: none"> 2. Mensaje de error si alguno de los datos es incorrecto. 3. Mensaje de error si ya existía un usuario con el mismo login.

Caso de uso: “Enviar notificación”
<p>Descripción: El sistema envía un email de notificación a un usuario.</p> <p>Papel dentro del trabajo del usuario: Caso de uso secundario.</p> <p>Actores: sistema</p> <p>Casos de uso relacionados: <u>Alta usuario</u>, <u>Alta usuarios múltiple</u>.</p> <p>Precondición: Un usuario se ha dado de alta o de baja en el sistema.</p> <p>Postcondición: Se ha enviado una notificación del alta o baja al usuario.</p>
Flujo Principal
<ol style="list-style-type: none"> 1. El sistema envía un email al usuario notificando el alta o baja en el sistema.
Flujos alternativos

Caso de uso: “Alta usuarios múltiple”
<p>Descripción: El administrador da de alta un grupo de usuarios.</p>

<p>Papel dentro del trabajo del usuario: Caso de uso principal.</p> <p>Actores: administrador</p> <p>Casos de uso relacionados: <u>Enviar notificación</u>, <u>Alta usuarios</u>.</p> <p>Precondición: El administrador se ha identificado ante el sistema. Existe un archivo para el alta múltiple con el nombre, login, password y tipo para los nuevos usuarios.</p> <p>Postcondición: Un grupo de usuarios ha sido dado de alta.</p>
Flujo Principal
<ol style="list-style-type: none"> 1. El administrador solicita un alta múltiple. 2. El administrador introduce el nombre del archivo para el alta múltiple. 3. Para cada usuario: <ol style="list-style-type: none"> a. El sistema da de alta el nuevo usuario. b. El sistema envía notificación de alta.
Flujos alternativos
<ol style="list-style-type: none"> 2. Mensaje de error si no se encuentra el archivo. 3. <ol style="list-style-type: none"> a. Mensaje de error si ya existía un usuario con el mismo login.

Caso de uso: “Consultar usuarios”
<p>Descripción: El administrador obtiene un listado de usuarios.</p> <p>Papel dentro del trabajo del usuario: Caso de uso secundario.</p> <p>Actores: administrador</p> <p>Casos de uso relacionados: <u>Baja usuario</u>, <u>Modificar usuario</u>.</p> <p>Precondición: El administrador se ha identificado ante el sistema.</p> <p>Postcondición: Se ha obtenido un listado de usuarios del sistema.</p>
Flujo Principal
<ol style="list-style-type: none"> 1. El sistema presenta una lista de usuarios.
Flujos alternativos

Caso de uso: “Baja usuario”
<p>Descripción: El administrador da de baja un usuario.</p> <p>Papel dentro del trabajo del usuario: Caso de uso principal.</p> <p>Actores: administrador</p> <p>Casos de uso relacionados: <u>Consultar usuarios</u>, <u>Alta usuario</u>.</p> <p>Precondición: El administrador se ha identificado ante el sistema y ha seleccionado un usuario.</p> <p>Postcondición: Un nuevo usuario ha sido dado de baja.</p>
Flujo Principal
<ol style="list-style-type: none"> 1. El administrador solicita dar de baja el usuario seleccionado. 2. El sistema envía notificación de baja. 3. El sistema da de baja el usuario
Flujos alternativos
<ol style="list-style-type: none"> 1. Mensaje de error si no se ha seleccionado un usuario.

Caso de uso: “Modificar usuario”
<p>Descripción: El administrador modifica los datos de un usuario.</p> <p>Papel dentro del trabajo del usuario: Caso de uso principal.</p> <p>Actores: administrador</p> <p>Casos de uso relacionados: <u>Consultar usuarios</u>.</p> <p>Precondición: El administrador se ha identificado ante el sistema y ha seleccionado un usuario.</p> <p>Postcondición: Los datos del usuario han sido modificados.</p>
Flujo Principal
<ol style="list-style-type: none"> 1. El administrador solicita modificar los datos de un usuario. 2. El administrador introduce los nuevos datos del usuario: nombre, login, password, tipo (estudiante, profesor). 3. El sistema modifica los datos del usuario.
Flujos alternativos
<ol style="list-style-type: none"> 1. Mensaje de error si no se ha seleccionado un usuario. 2. Mensaje de error si alguno de los datos es incorrecto.

Caso de uso: “Buscar usuario”
Descripción: El administrador busca un usuario por su login. Papel dentro del trabajo del usuario: Caso de uso principal. Actores: administrador Casos de uso relacionados: <u>Consultar usuarios</u> . Precondición: El administrador se ha identificado ante el sistema. Postcondición: Se ha buscado un usuario en la lista de usuarios.
Flujo Principal
1. El administrador solicita buscar un usuario. 2. El administrador introduce el login del usuario. 3. El sistema selecciona al usuario en la lista de usuarios.
Flujos alternativos
3. Mensaje de error si no se encuentra el usuario.

1.2 Especificación de las clases de análisis.

1.2.1 Identificación de las clases de entidad.

A partir de los casos de uso identificaremos las clases de entidad:

Caso de uso Login: **usuario, estudiante, profesor, administrador**.

Caso de uso Verificar modelo: **modeloUML, herramienta UMLtoCLP (validador), repositorio**

Caso de uso Cambiar password: **usuario*, estudiante*, profesor*, administrador***.

Caso de uso Consultar ejercicios: **ejercicio, colección, usuario*, estudiante*, profesor***.

Caso de uso Ver enunciado: **ejercicio*, colección*, usuario*, estudiante*, profesor***.

Caso de uso Resolver ejercicio: **ejercicio*, colección*, usuario*, estudiante*, profesor*, modeloUML***.

Caso de uso Obtener solución: **ejercicio*, colección*, usuario*, estudiante*, profesor*, modeloUML***.

Caso de uso Crear colección: **ejercicio*, colección*, usuario*, profesor***.

Caso de uso Consultar colecciones: **colección*, usuario*, estudiante*, profesor***.

Caso de uso Eliminar colección: **ejercicio*, colección*, usuario*, profesor***.

Caso de uso Modificar colección: **ejercicio*, colección*, usuario*, profesor***.

Caso de uso Crear ejercicio: **ejercicio*, colección*, usuario*, profesor*, archivo**.

Caso de uso Eliminar ejercicio: **ejercicio*, colección*, usuario*, profesor***.

Caso de uso Modificar ejercicio: **ejercicio*, colección*, usuario*, profesor***.

Caso de uso Alta usuario: **usuario*, administrador*, notificación?**.

Caso de uso Enviar notificación: **usuario*, notificación?**.

Caso de uso Alta usuarios múltiple: **administrador*, usuario*, archivo***.

Caso de uso Consultar usuarios: **administrador*, usuario***.

Caso de uso Baja usuario: **administrador*, usuario*, notificación?**.

Caso de uso Modificar usuario: **administrador*, usuario***.

Caso de uso Buscar usuario: **administrador*, usuario***.

Las clases identificadas son: usuario, estudiante, profesor, administrador, modeloUML, ejercicio, colección, archivo, repositorio y validador. La entidad notificación se descarta por no tener ni atributos ni operaciones propios.

1.2.2 Especificación de los atributos de las clases de entidad.

Clase **usuario**: nombre(string), login(string), password(string).

Clase **estudiante**, **profesor**, **administrador**: ninguno.

Clase **modeloUML**: fileUML(archivo), fileOCL(archivo).

Clase **ejercicio**: descriptivo(string), num_orden(integer), enunciado(text),
solucion(modeloUML).

Clase **colección**: descriptivo(string).

Clase **archivo**: nombre(string), data(binario), size(int)

Clase **repositorio**: autor(string), fecha(date), valido(boolean), propiedades(archivo)

Clase **validador**: modelo(archivo), propiedades(archivo), clases(lista),
restricciones(lista), resultado(archivo), imgResultado(archivo)

1.2.3 Diagrama de clases de entidad

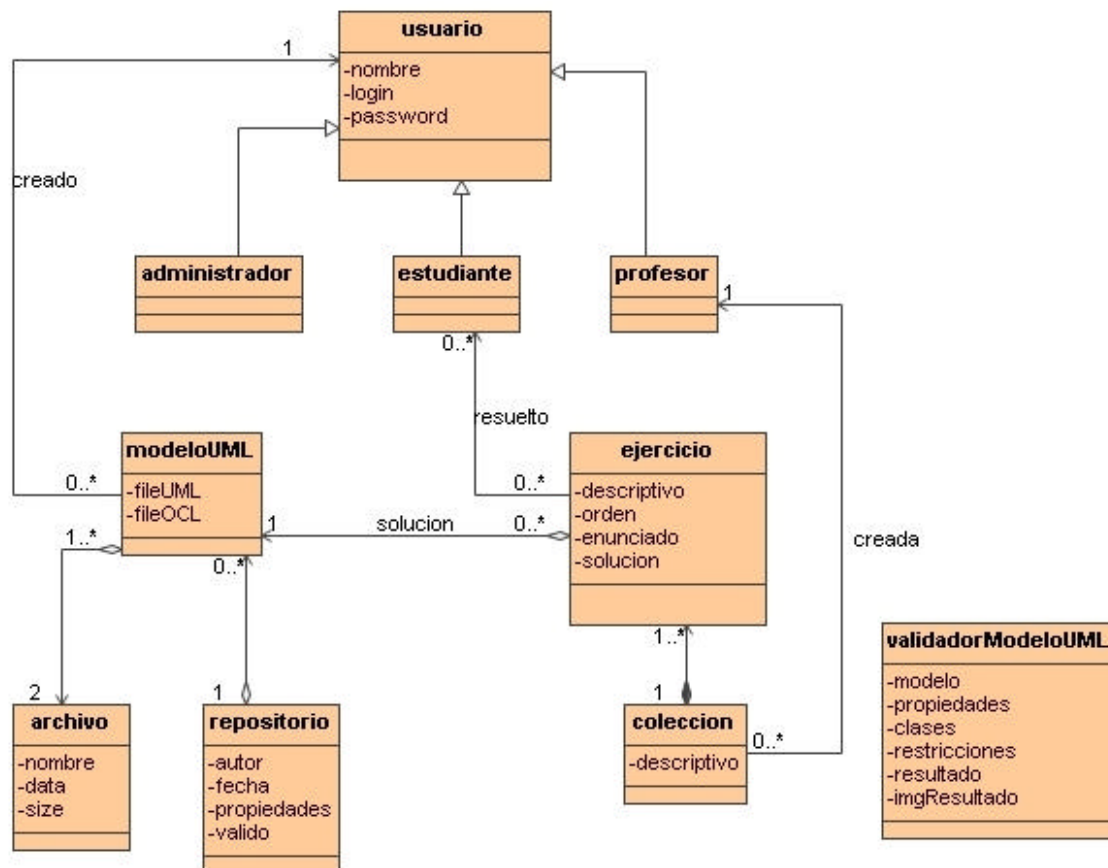


Fig. 3 Diagrama de clases de entidad

Tenemos una relación de herencia simple por especialización. La superclase es usuario y las subclases disjuntas estudiante, profesor, administrador.

También existen varias agregaciones. Entre ejercicio y colección tenemos una composición puesto que cada ejercicio pertenece a una única colección.

ValidadorModeloUML lo consideraremos una clase de entidad, su función es la de representar un modelo tratado por la herramienta UMLtoCSP.

1.2.4 Identificación de las clases de frontera y control y de las operaciones.

Realizamos un diagrama de colaboración simplificado para cada caso de uso de los que se obtendrán las clases de frontera y control necesarias así como las principales operaciones que deberán implementar.

- **Login**

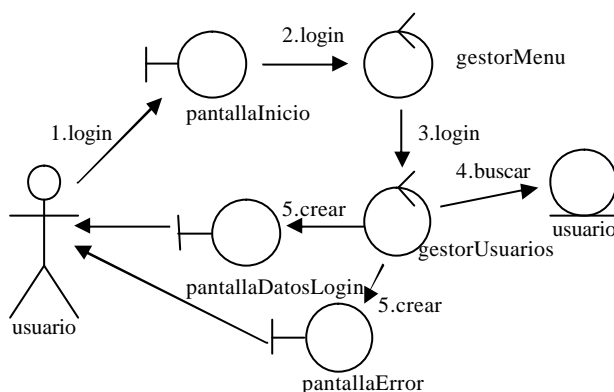


Fig. 4 Caso de uso *login*

- **Verificar modelo.**

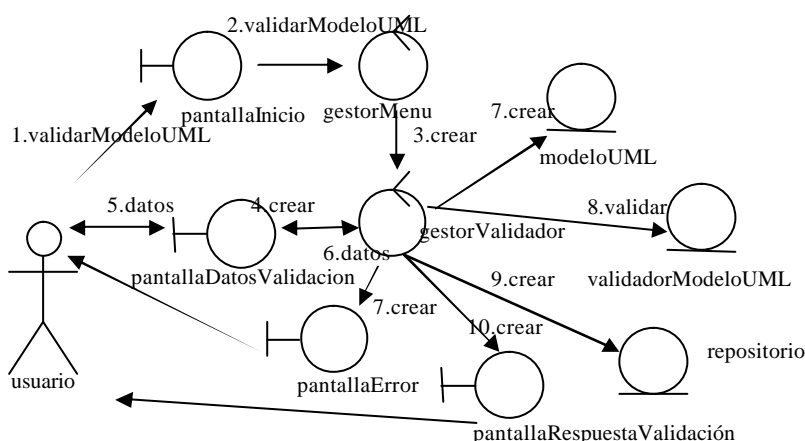


Fig. 5 Caso de uso *verificar modelo*

- **Cambiar password.**

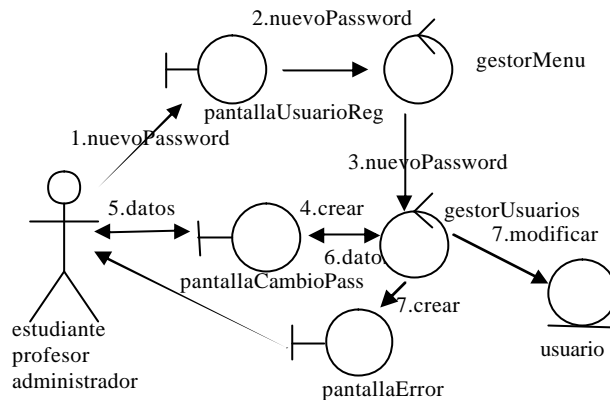


Fig. 6 Caso de uso *cambiar password*

- **Consultar ejercicios.**

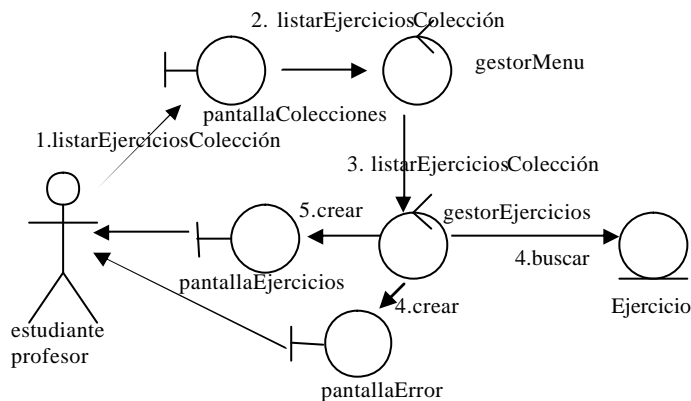


Fig. 7 Caso de uso *consultar ejercicios*

- **Ver enunciado.**

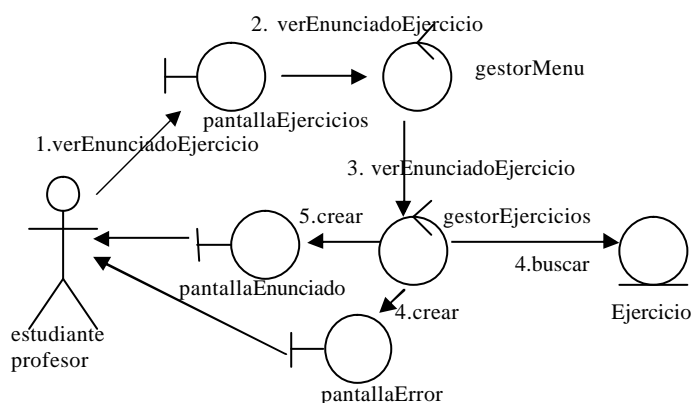


Fig. 8 Caso de uso *ver enunciado*

- Resolver ejercicio.

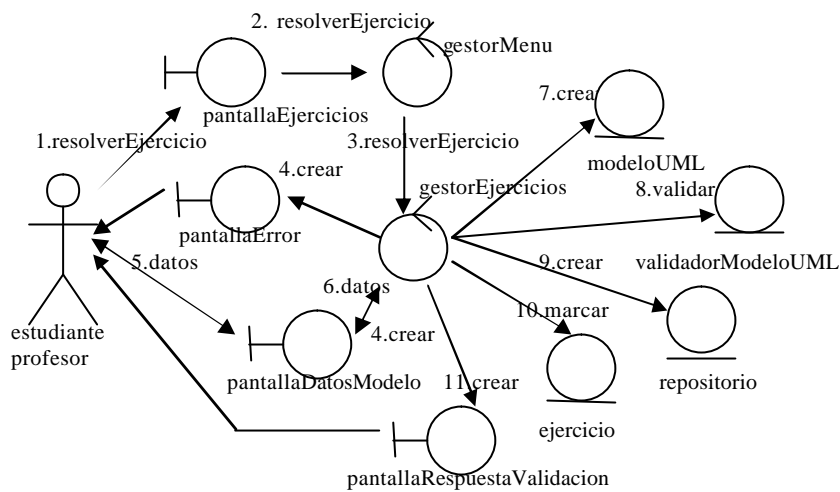


Fig. 9 Caso de uso *resolver ejercicio*

- Obtener solución.

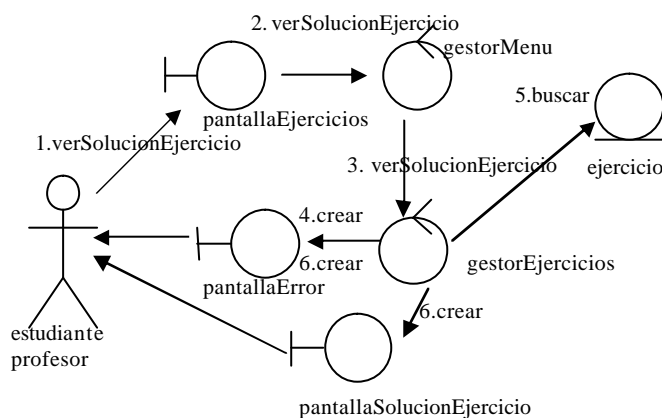


Fig. 10 Caso de uso *obtener solucion*

- Crear colección

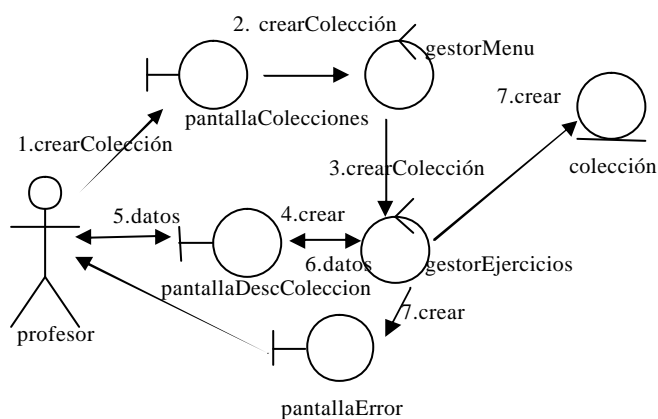


Fig. 11 Caso de uso *crear colección*


```

graph TD
    Actor[estudiante profesor] -- "1. listarColecciones" --> PantallaReg((pantallaUsuarioReg))
    PantallaReg -- "2. listarColecciones" --> GestorMenu((gestorMenu))
    GestorMenu -- "3. listarColecciones" --> GestorEjercicios((gestorEjercicios))
    GestorEjercicios -- "5. crear" --> PantallaColec((pantallaColecciones))
    PantallaColec -- "4. buscar" --> Coleccion((coleccion))
    Coleccion --> Actor
  
```

- **Eliminar colección.**



```

sequenceDiagram
    actor profesor
    participant pantallaColecciones
    participant pantallaDescriptivo
    participant pantallaError
    participant gestorMenu
    participant gestorEjericicios

    profesor->>pantallaColecciones: 1.actualizarColección
    activate pantallaColecciones
    pantallaColecciones->>gestorMenu: 2. actualizarColección
    deactivate pantallaColecciones
    activate gestorMenu
    gestorMenu->>pantallaDescriptivo: 3. actualizarColección
    deactivate gestorMenu
    activate pantallaDescriptivo
    pantallaDescriptivo->>gestorEjericicios: 4.crear
    deactivate pantallaDescriptivo
    activate gestorEjericicios
    gestorEjericicios->>pantallaDescriptivo: 6.datos
    deactivate gestorEjericicios
    activate pantallaDescriptivo
    pantallaDescriptivo->>profesor: 5.datos
    deactivate pantallaDescriptivo
    activate profesor
    profesor->>pantallaError: 
    deactivate profesor
    activate pantallaError
    pantallaError->>gestorEjericicios: 4.crear
    deactivate pantallaError
    activate gestorEjericicios
    gestorEjericicios->>gestorMenu: 7.modificar
    deactivate gestorEjericicios
    activate gestorMenu
    gestorMenu->>gestorEjericicios: 
    deactivate gestorMenu
    deactivate gestorEjericicios
  
```

Página 25 de 61

- **Crear ejercicio**

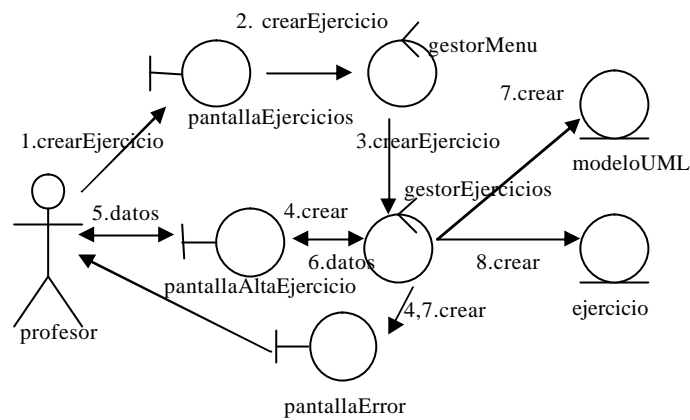


Fig. 15 Caso de uso *crear ejercicio*

- **Eliminar ejercicio.**

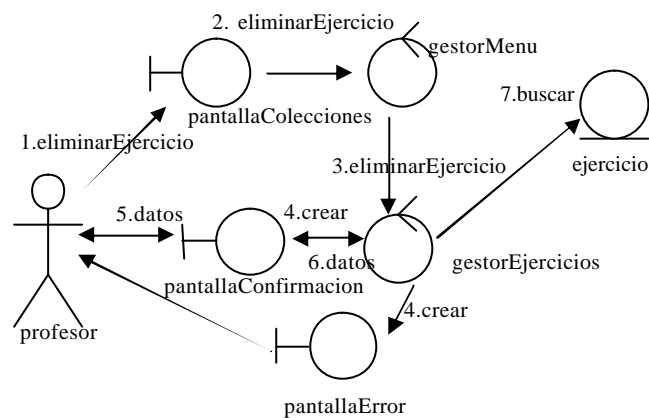


Fig. 16 Caso de uso *eliminar ejercicio*

- **Modificar ejercicio.**

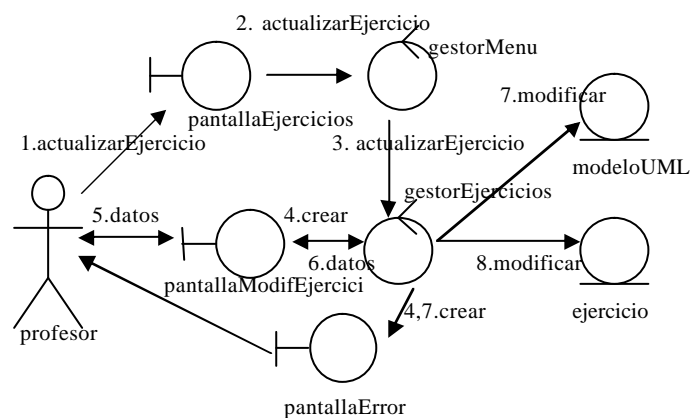


Fig. 17 Caso de uso *modificar ejercicio*

- **Alta usuario.**

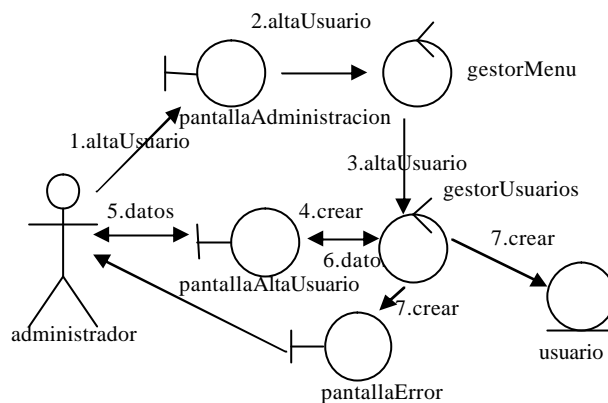


Fig. 18 Caso de uso *alta usuario*

- **Alta usuarios múltiple.**

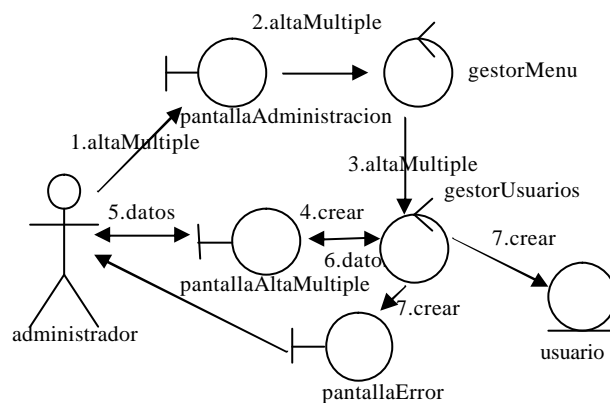


Fig. 19 Caso de uso *alta usuarios múltiple*

- **Baja usuario.**

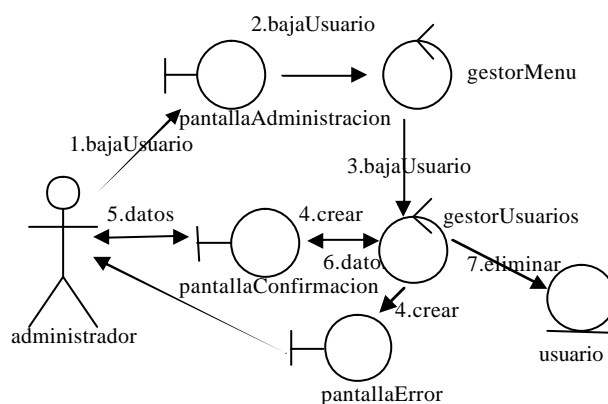


Fig. 20 Caso de uso *baja usuario*

- **Modificar usuario.**

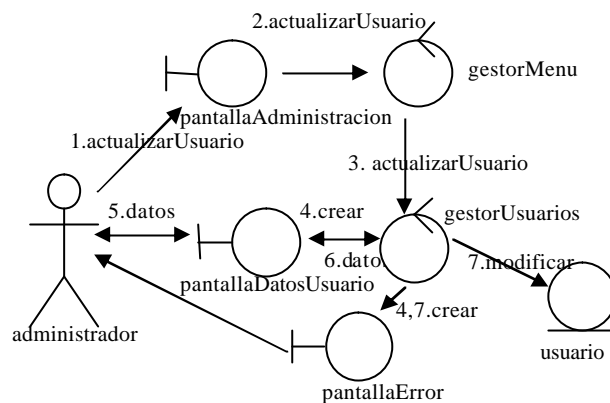


Fig. 21 Caso de uso *modificar usuario*

- **Buscar usuario.**

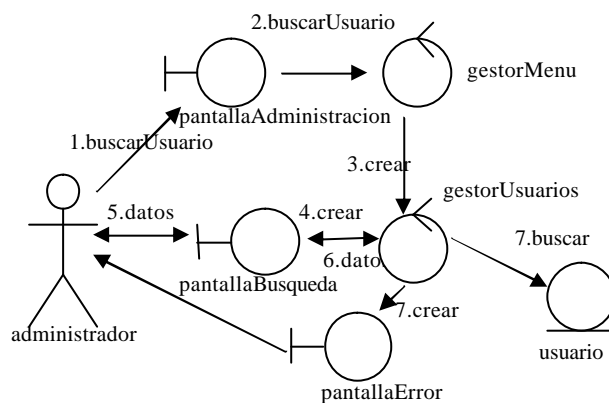


Fig. 22 Caso de uso *buscar usuario*

1.2.5 Especificación formal de los casos de uso.

Realizamos diagramas de secuencia de los casos de uso más representativos.

- Login.

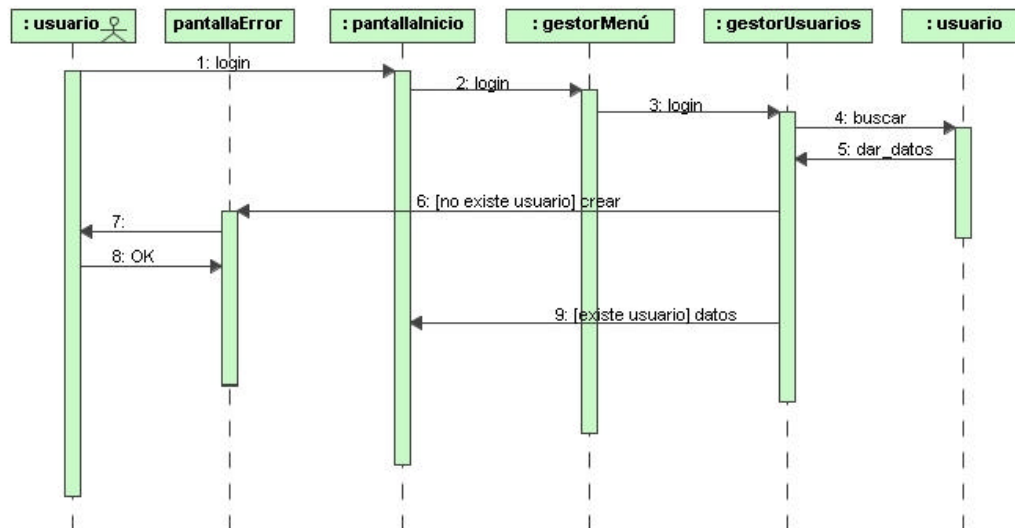


Fig. 23 Diagrama de secuencia caso de uso *login*

- Resolver ejercicio.

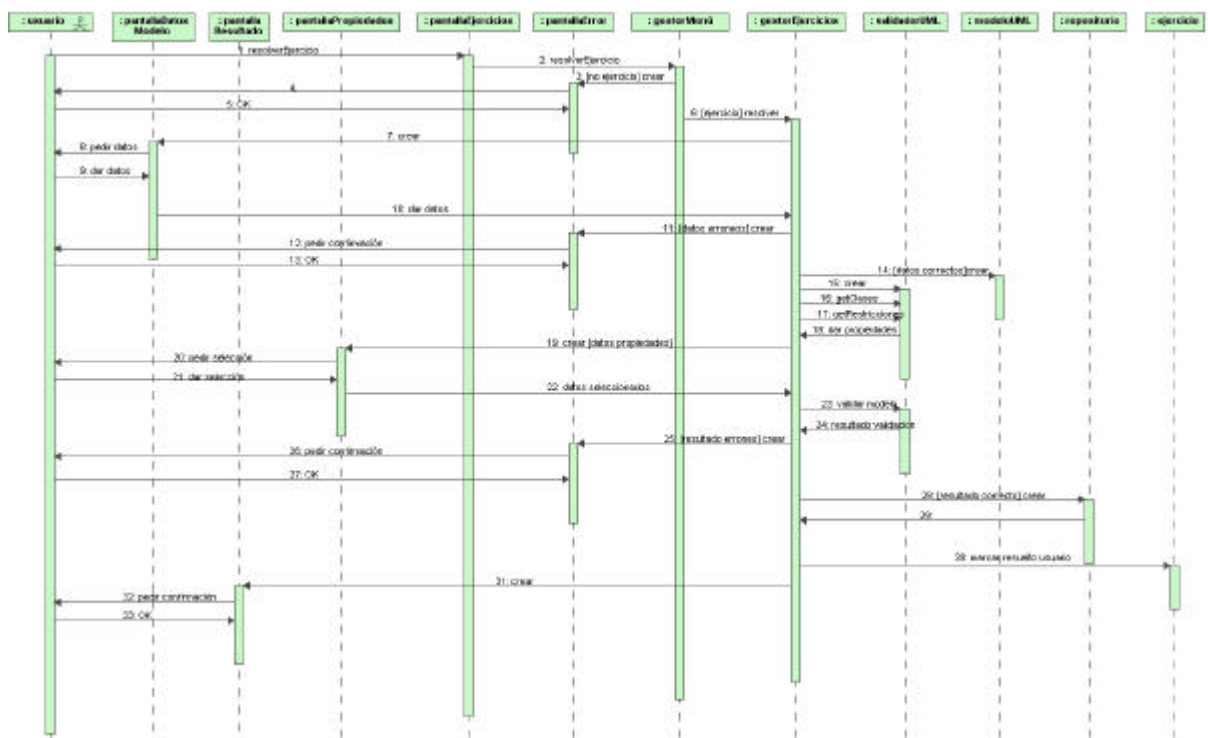


Fig. 24 Diagrama de secuencia caso de uso *resolver ejercicio*

El caso de uso *verificar modelo* es muy parecido a éste y no se representa.

- Consultar ejercicios.

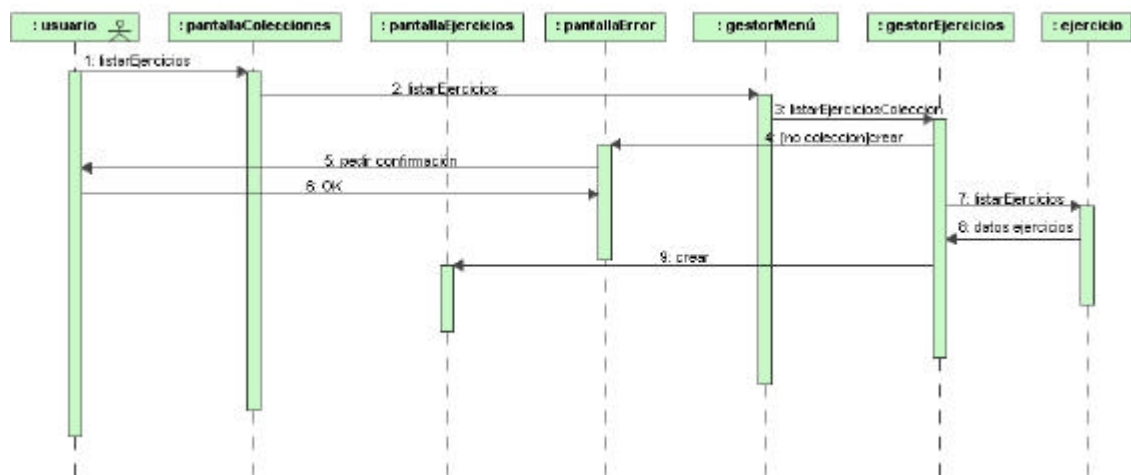


Fig. 25 Diagrama de secuencia caso de uso *consultar ejercicios*

Consultar colecciones es un caso de uso muy similar a éste.

- Obtener solución.

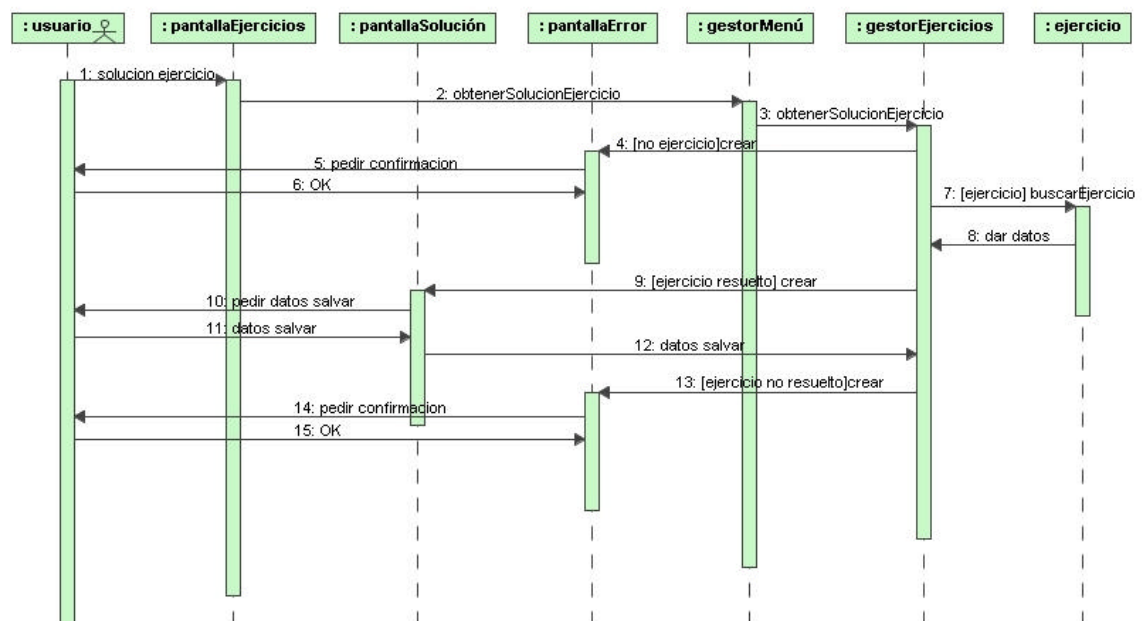


Fig. 26 Diagrama de secuencia caso de uso *obtener solución*

- Crear ejercicio.

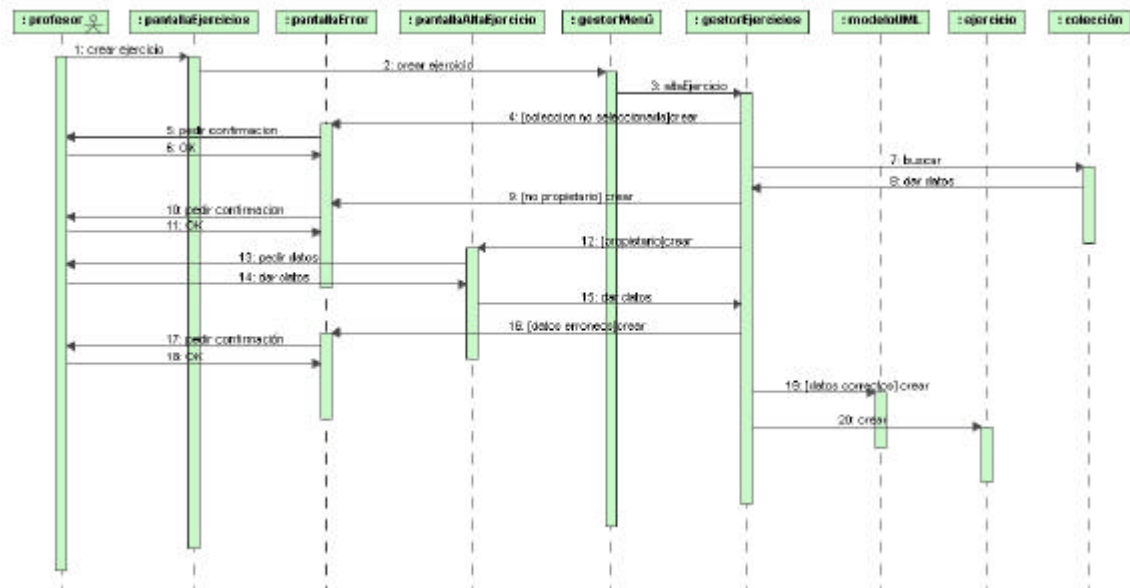


Fig. 27 Diagrama de secuencia caso de uso *crear ejercicio*

Los casos de uso *modificar ejercicio* y *eliminar ejercicio* son muy parecidos a éste y no se representan.

- Ver enunciado.

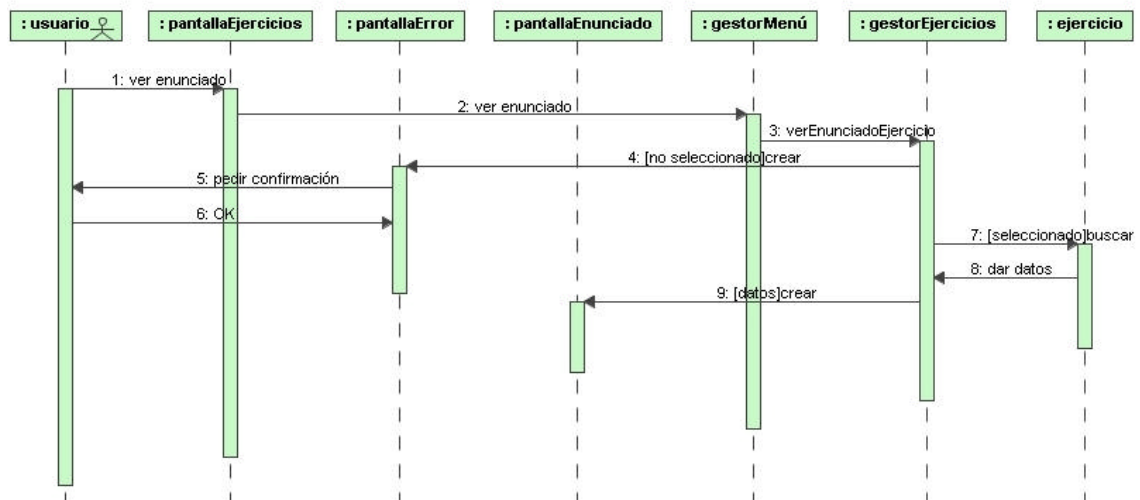


Fig. 28 Diagrama de secuencia caso de uso *ver enunciado*

- **Cambiar password.**

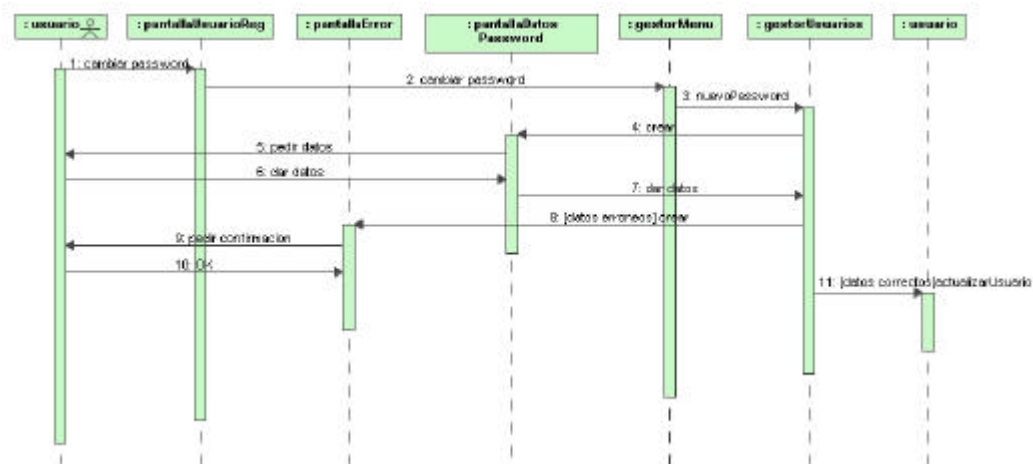


Fig. 29 Diagrama de secuencia caso de uso *cambiar password*

- **Eliminar colección.**

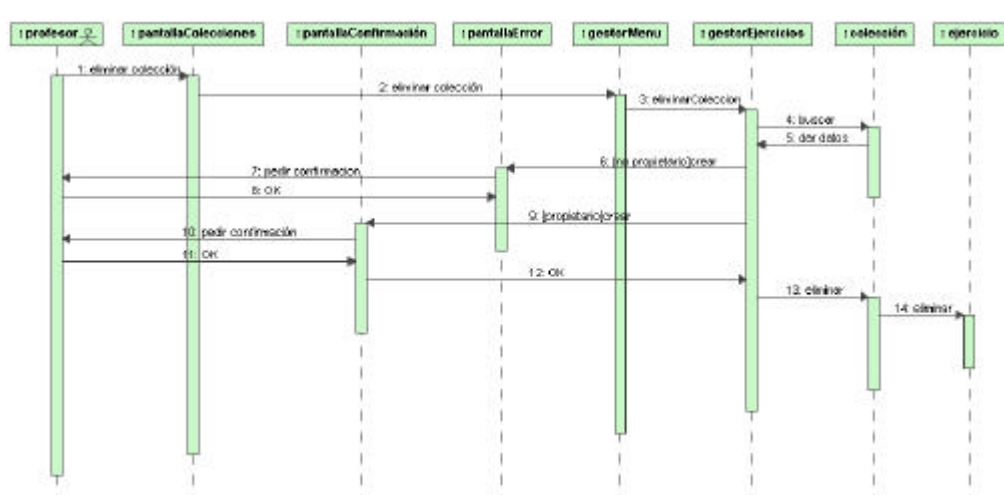


Fig. 30 Diagrama de secuencia caso de uso *eliminar colección*

- **Alta usuario.**

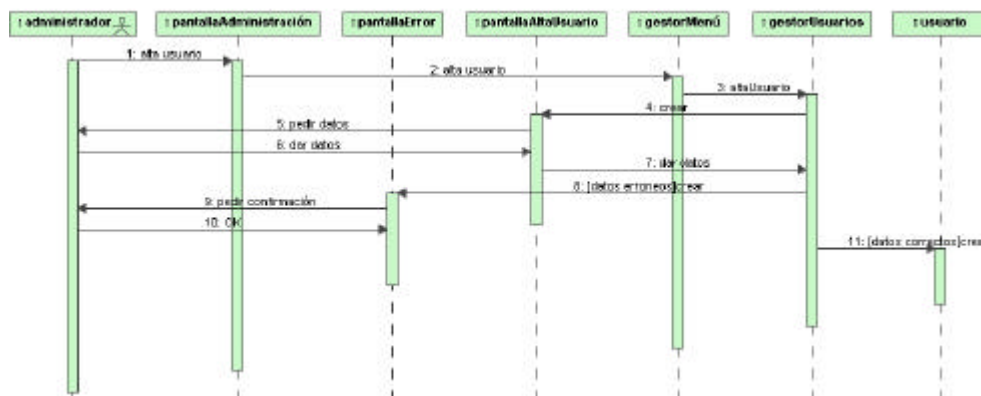


Fig. 31 Diagrama de secuencia caso de uso *alta usuario*

Similar a *baja usuario*, *modificar usuario* y *buscar usuario* que no se representan.

2 DISEÑO

2.1 Diseño de la arquitectura de la aplicación.

2.1.1 Diseño de alto nivel.

A alto nivel podemos descomponer la funcionalidad del aplicativo en cuatro componentes principales:

Usuarios: encargado de presentar y gestionar la información de los usuarios y el proceso de autenticación.

Ejercicios: encargado de la gestión y presentación de todo lo referente a las colecciones de ejercicios, tanto para su creación como para su utilización.

Validador: se ocupa del proceso de validación de los modelos UML, incluyendo la necesaria adaptación con la herramienta UMLtoCSP.

Repositorio: Gestiona el almacenamiento de los modelos UML presentados a validación.

Cada uno de estos componentes implementa una funcionalidad básica con una serie de operaciones que han sido identificadas durante el proceso de análisis.

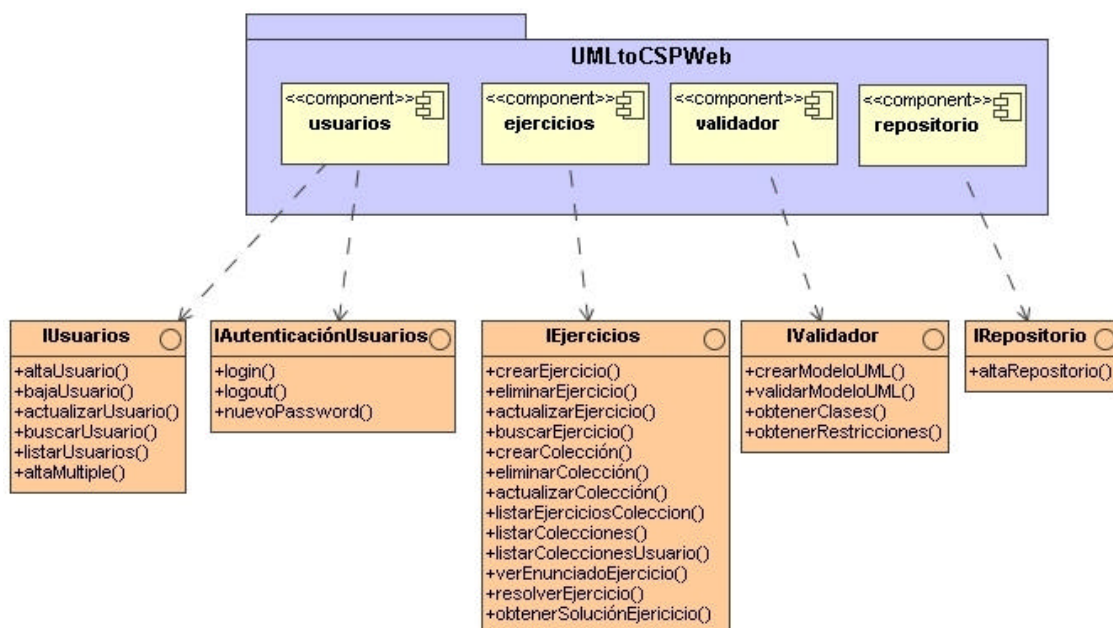


Fig. 32 Diagrama de diseño de alto nivel

2.1.2 Diseño arquitectónico detallado.

Estamos desarrollando un aplicativo web para el que uno de los modelos arquitectónicos más extendidos y que será el que usaremos, es el de cliente/servidor estructurado en tres capas:

- Capa de presentación: que permite la interacción del usuario con la aplicación.
- Capa de negocio: que implementa la funcionalidad básica de la aplicación.
- Capa de integración: que interactúa con las fuentes de datos que almacenan permanentemente la información.

Así el siguiente paso en el diseño ha consistido en descomponer cada uno de los componentes anteriores en las tres capas definiendo más exactamente las responsabilidades de cada uno y sus interrelaciones. El diagrama siguiente muestra esa descomposición representando las tres capas como paquetes UML. El componente Validador no tiene representación en la capa de integración puesto que no es necesario almacenar ningún dato relativo a él de forma permanente.

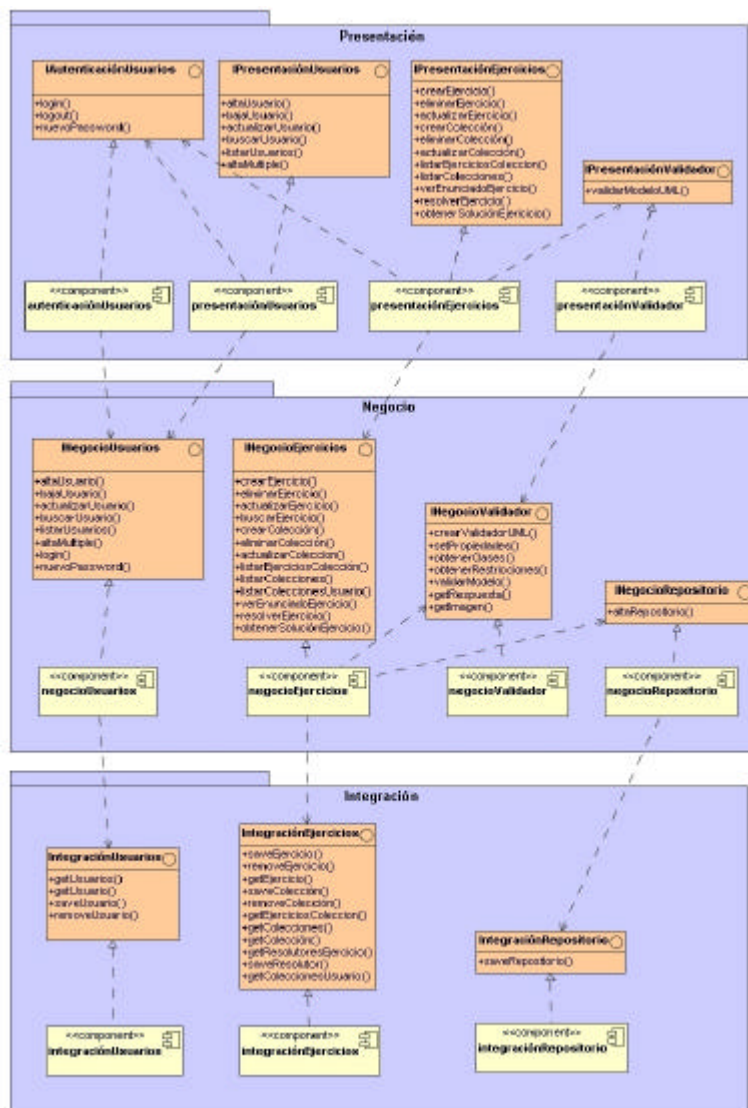


Fig. 33 Diagrama de diseño arquitectónico detallado

Seguiremos profundizando el proceso tomando decisiones de diseño de la arquitectura interna de algunos de los componentes, para ello nos apoyaremos en el uso de patrones de arquitectónicos y de diseño.

2.1.2.1 Capa de presentación.

Para la capa de presentación usaremos el patrón arquitectónico MVC-2 (Modelo Vista Controlador). Su principal objetivo es aislar tanto los datos de la aplicación como el estado de la misma (Modelo), del mecanismo utilizado para representar (Vista) dicho estado, así como para modularizar esta vista y modelar la transición entre estados del modelo (Controlador). Así la capa se divide en tres áreas funcionales:

- **Vista:** la presentación de los datos.
- **Controlador:** el que atenderá las peticiones y componentes para toma de decisiones de la aplicación.
- **Modelo:** la lógica del negocio o servicio y los datos asociados con la aplicación.

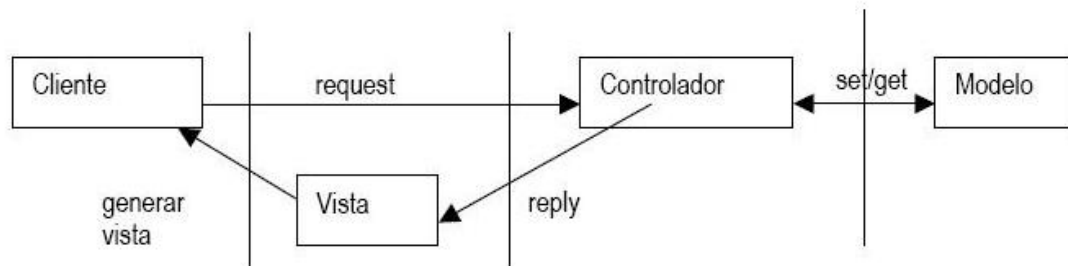


Fig. 34 Modelo MVC-2

MVC proporciona desacoplo entre las capas. Desacopla datos y lógica de negocio de la lógica de presentación permitiendo la actualización y desarrollo independiente de cada uno de los citados componentes, proporciona por tanto reusabilidad, adaptabilidad y mantenibilidad.

Si consideramos que el modelo se hallará implementado en la capa de negocio, cada uno de los componentes de la capa de presentación (autenticaciónUsuarios, presentaciónUsuarios, presentaciónEjercicios y presentaciónValidador) se dividirá en un componente que hará de controlador y un componente que hará de vista.

- **AutenticaciónUsuarios.**

Este componente incluirá dos vistas, las pantallas de login de los usuarios y administrador y la de cambio de password.

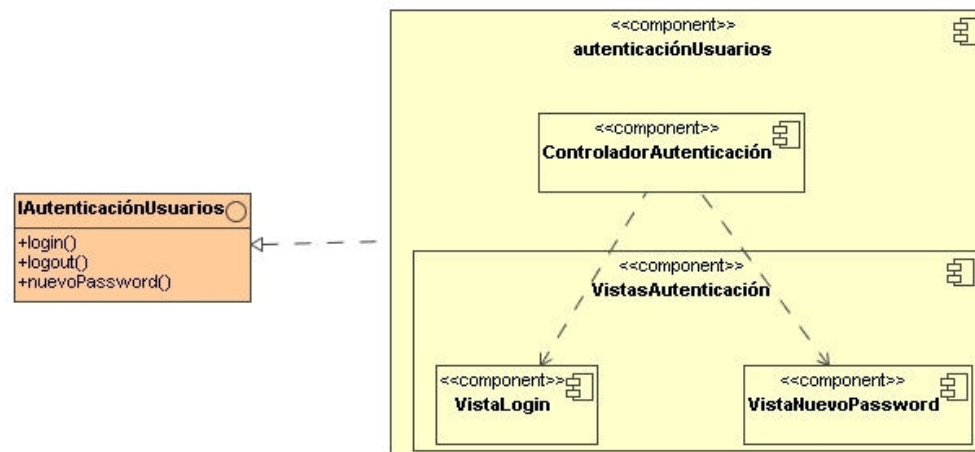


Fig. 35 Diagrama del componente *autenticaciónUsuarios*

- **presentaciónUsuarios.**

Componente accesible únicamente en el modo de administración y que incluirá las vistas de la lista de usuarios del sistema, el alta de un nuevo usuario, y el alta múltiple de usuarios.

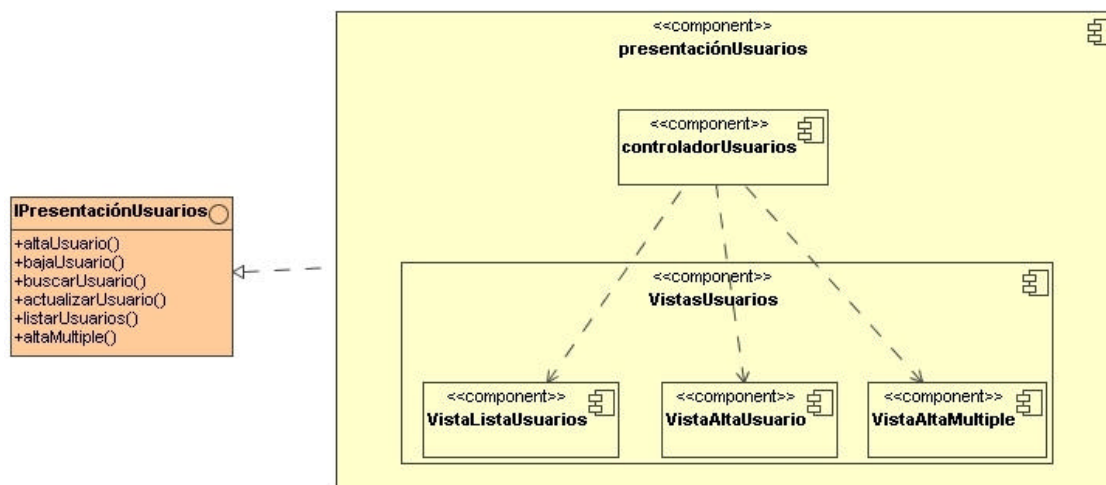


Fig. 36 Diagrama del componente *presentaciónUsuarios*

- **presentaciónEjercicios.**

Este componente será accesible únicamente a los usuarios registrados y permitirá distintas funciones según se trate del tipo de usuario: estudiante o profesor.

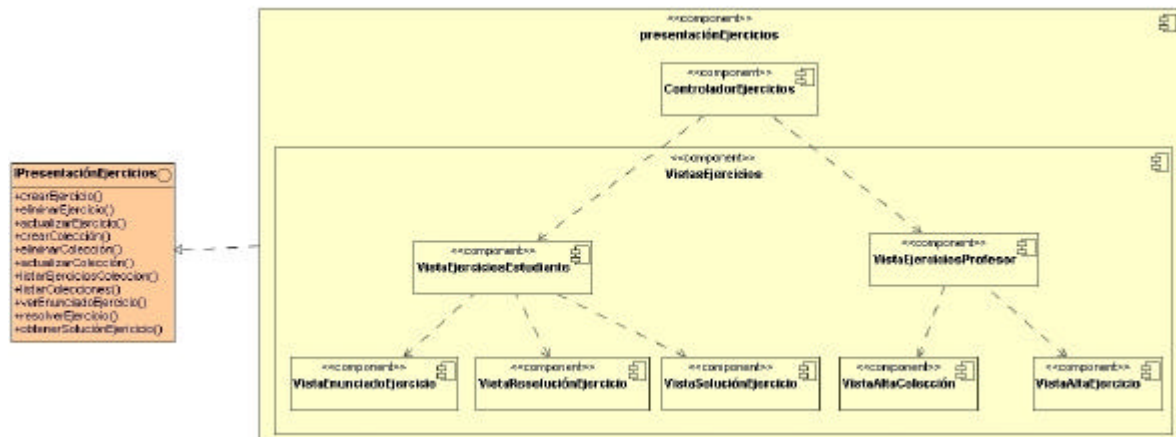


Fig. 37 Diagrama del componente *presentaciónEjercicios*

- **presentaciónValidador.**

Componente accesible para usuarios registrados y no registrados, que representa el proceso de validación de un modelo UML.

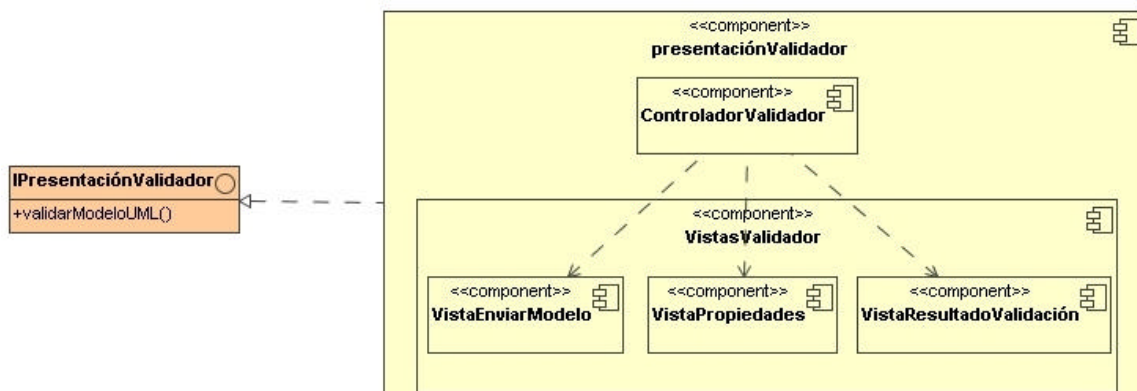


Fig. 38 Diagrama del componente *presentaciónValidador*

2.1.2.2 Capa de negocio.

En la capa de negocio se han identificado los cuatro componentes mostrados en la figura 33, que son los encargados de proporcionar la lógica de negocio independientemente de cómo sea la presentación de los datos y su almacenamiento en la base de datos.

Cada uno de los componentes implementa una interfaz y puede hacer uso de otras interfaces tanto de su misma capa como de otras capas. Las operaciones sobre estos componentes se realizarán de forma síncrona (el cliente espera la respuesta del servidor).

Para esta capa haremos uso del patrón Facade o Fachada que permite que solo se ofrezcan unos pocos puntos de entrada al sistema que nos oculten el conjunto de clases que lo forman. Como resultado se simplifica el acceso a la capa de negocio desde la

capa de presentación al mismo tiempo que aísla los posibles cambios que puedan producirse en una de las partes.

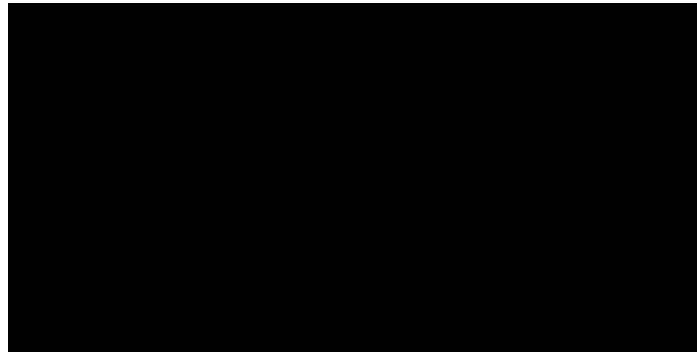


Fig. 39 Patrón Facade

- **NegocioUsuarios.**

Se encarga de la gestión de los usuarios.

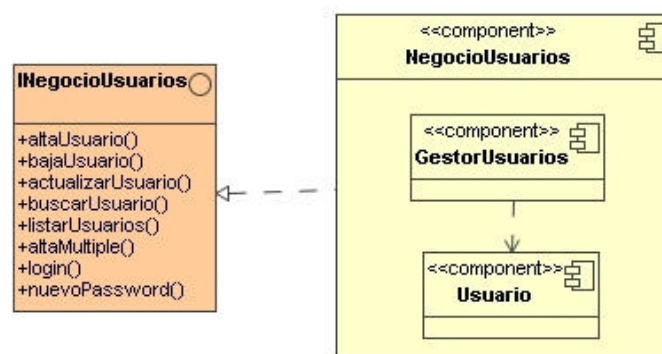


Fig. 40 Diagrama del componente *NegocioUsuarios*

- **NegocioEjercicios.**

Se encarga de la gestión de las colecciones y los ejercicios de que se componen.

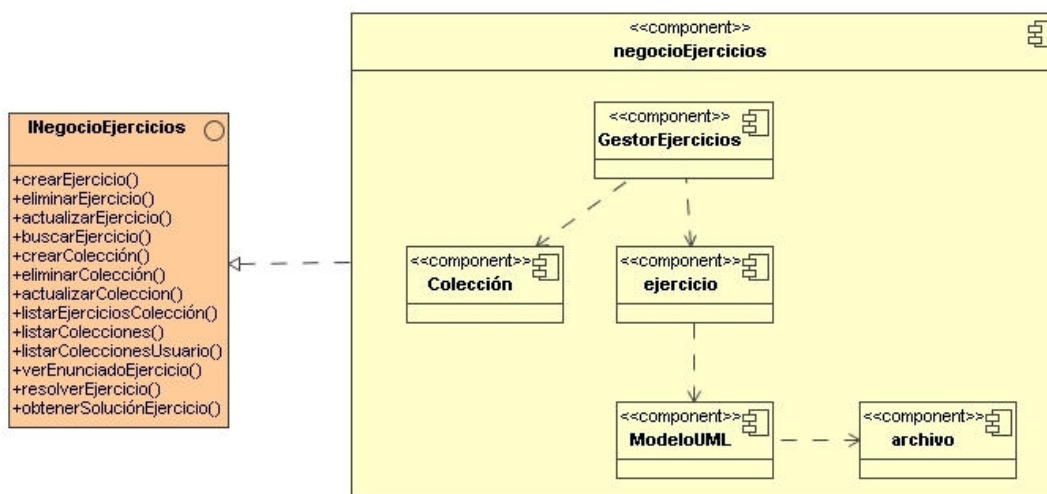


Fig. 41 Diagrama del componente *NegocioEjercicios*

- **NegocioValidador.**

Este componente encapsula la funcionalidad de la herramienta UMLtoCSP y permite realizar la validación de un modelo UML. Para su desarrollo nos apoyaremos en un patrón Proxy.

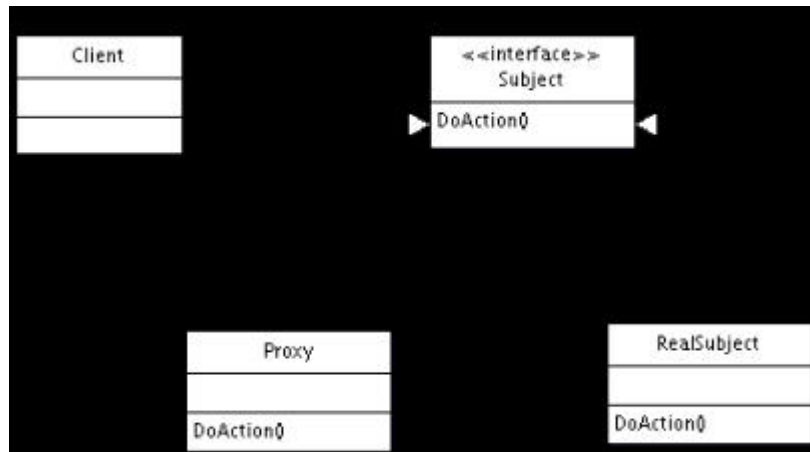


Fig. 42 Patrón proxy

Un Proxy es en términos generales una clase que funciona como un interfaz a otro objeto que como es el caso de la herramienta UMLtoCSP puede residir en otra máquina.

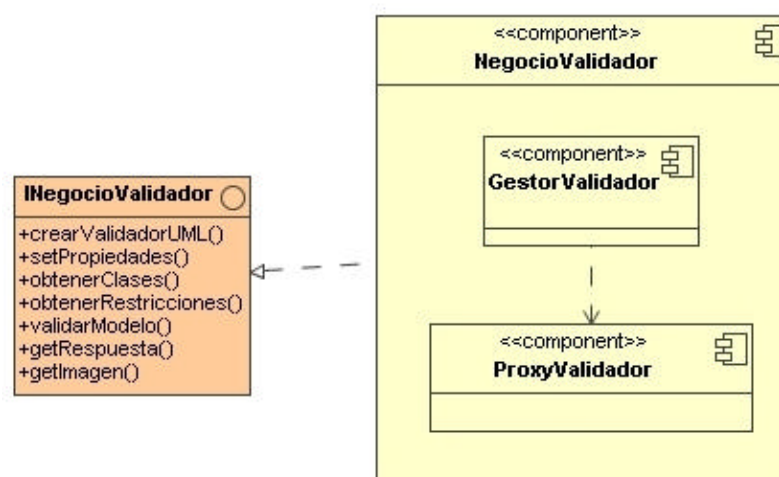


Fig. 43 Diagrama del componente *NegocioValidador*

- **NegocioRepositorio.**

Se encarga de almacenar en el repositorio los modelos presentados a validación.

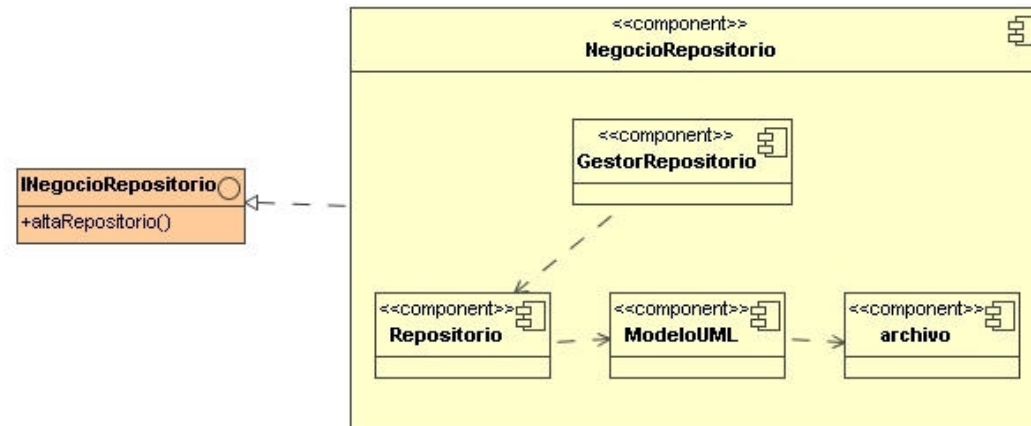


Fig. 44 Diagrama del componente *NegocioRepositorio*

2.1.2.3 Capa de integración.

Para la capa de integración utilizaremos el patrón DAO (Data Access Object) para encapsular los accesos a las fuentes de datos. Los componentes de negocio se apoyan en el DAO a través de la interfaz que éste expone y el DAO oculta completamente los detalles de la implementación concreta de las fuentes de datos.

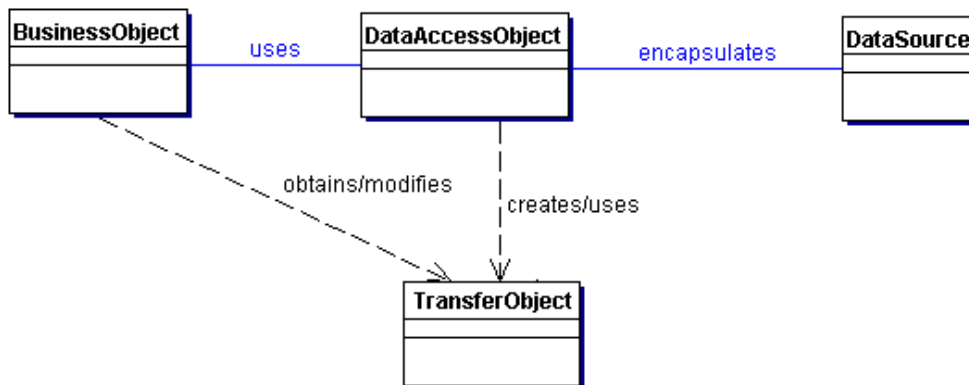


Fig. 45 Patrón DAO

Con el patrón DAO obtenemos dos beneficios fundamentales:

- Aislar las conexiones a la fuente de datos en una capa fácilmente identificable y mantenible.
- Se disminuye el nivel de acoplamiento entre clases, reduciendo la complejidad de realizar cambios.

- IntegraciónUsuarios.

Nos permitirá almacenar los datos de los usuarios.

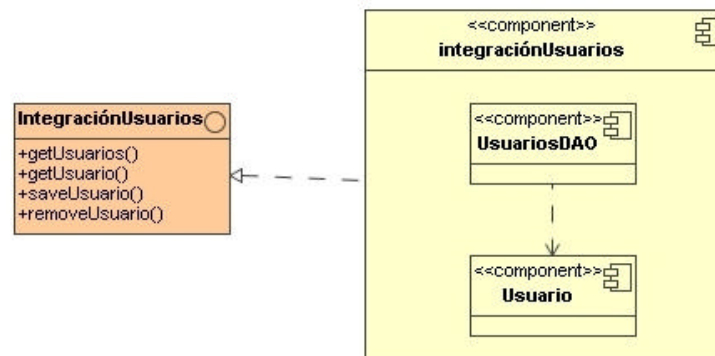


Fig. 46 Diagrama del componente *integraciónUsuarios*

- IntegraciónEjercicios.

Mediante éste módulo se almacenarán los datos relativos a las colecciones y sus ejercicios.

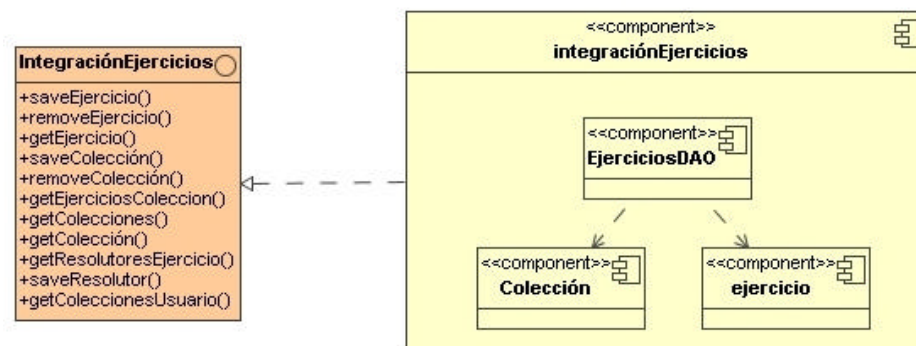


Fig. 47 Diagrama del componente *integraciónEjercicios*

- IntegraciónRepositorio.

Almacenará los datos relativos a los modelos UML presentados a validación. La gestión del repositorio solo prevé inicialmente la operación de alta.

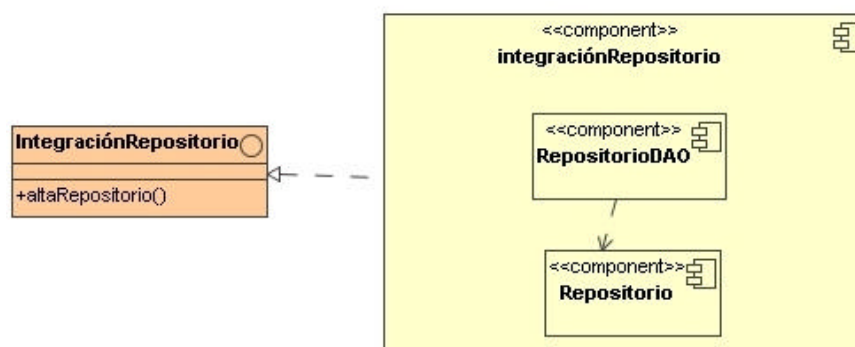


Fig. 48 Diagrama del componente *integraciónRepositorio*

2.1.3 Integración con la herramienta UMLtoCSP.

La integración con la herramienta UMLtoCSP se realizará creando un **web service** que correrá sobre la máquina donde reside la herramienta.

La World Wide Web Consortium (W3C) define un servicio web como un sistema de software diseñado para soportar interacciones máquina a máquina con intercambio de información sobre una red. Un web service es esencialmente una API que puede ser accedida a través de una red y ejecutada en un sistema remoto. Los estilos más comunes de uso de servicios web son tres:

- **RPC.** Basados en las llamadas a función. Típicamente la unidad básica de los web services RPC es la operación WSDL.
- **SOA.** Implementan una arquitectura de acuerdo a los conceptos de una arquitectura orientada al servicio, donde la unidad básica de comunicación es el mensaje y no la operación.
- **REST.** Intentan emular el http y protocolos similares limitando la interfaz a una serie de operaciones estándar como GET, PUT, DELETE etc.

Mientras que no existe una especificación única para los web services si existen algunas especificaciones que se encuentran comúnmente en todos ellos.

SOAP. Es un protocolo para el intercambio de mensajes basados en XML sobre una red de computadores, normalmente usando http. SOAP detalla con precisión como debe codificarse un encabezado http y un archivo xml para que un programa funcionando en una máquina determinada pueda llamar a un programa funcionando en otra máquina y pasarle información y también como el programa llamado puede devolver una respuesta.

WSDL. Es un lenguaje basado en XML que proporciona un modelo para describir web services. Es esencialmente una descripción de cómo comunicarse utilizando web services. Un programa cliente que se conecta a un web service puede leer el documento WSDL para determinar que funciones están disponibles en el servidor. Cualquier tipo de datos utilizados son incluidos en el documento WSDL en forma de un esquema XML. El cliente puede entonces usar SOAP para llamar una de las funciones listadas en el WSDL.

Utilizaremos el estilo SOA y haremos uso de las especificaciones SOAP y WSDL. Aunque dado que las dos aplicaciones a comunicar están escritas en el mismo lenguaje, Java, y podríamos utilizar un método de comunicación más eficiente como RMI se ha optado por el uso de un web service por varios motivos:

- Fomenta el uso de protocolos y estándares basados en texto que hacen más fácil acceder a su contenido y entender su funcionamiento.
- Son muy prácticos ya que pueden aportar gran independencia entre la aplicación que usa el servicio Web y el propio servicio. Además al estar basados en http sobre TCP permiten que servicios y software ubicados en diferentes lugares geográficos puedan ser combinados fácilmente.
- Es una tecnología en auge que merece la pena ser estudiada.

El web service ofrecerá las siguientes funcionalidades:

- ValidarModelo. Un método para validar un modelo UML/OCL. Se le entregará:
 - o El contenido del archivo UML a validar, en forma de array de bytes.
 - o El contenido del archivo OCL, en forma de array de bytes.
 - o Una cadena opcional con las propiedades que se desean validar, con el siguiente formato:
[-strong-sat] [-weak-sat] [-liveliness clase] [-no-subsumption restriccion1 restriccion2] [-no-redundancy restriccion1 restriccion2]
 - o Una cadena con un identificador para el modelo. Es importante que este identificador sea único para evitar posibles interferencias con otras validaciones simultáneas.

El método devuelve una cadena con la respuesta de la herramienta UMLtoCSP.

- getResultado. Un método para obtener la respuesta del canal de errores de la herramienta UMLtoCSP. Se le entrega el identificador del modelo validado del que se desea obtener el canal de errores y devuelve una cadena correspondiente a dicho canal.
- getImagen. Un método para obtener la imagen que puede devolver la herramienta UMLtoCSP en caso de validación positiva. Se le entrega el identificador del modelo validado del que se desea obtener la imagen y devuelve un array de bytes correspondiente a dicha imagen que puede ser vacío si no existiera.

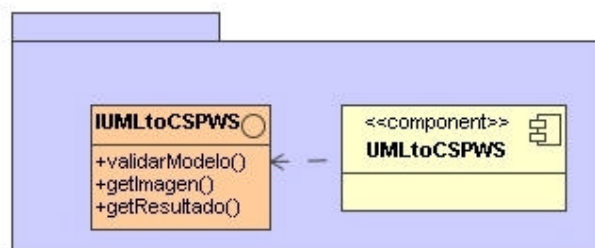


Fig. 49 Diagrama del Web Service

2.1.4 Diagrama de despliegue.

Relativas al despliegue de la aplicación las decisiones que se han tomado son las siguientes:

- Se dividirá el acceso en una parte pública que tendrá funciones para usuarios anónimos y usuarios registrados y una parte privada usada únicamente para la administración de los usuarios.
- Todos los componentes (con una única excepción) se ejecutarán en una misma máquina, con lo que los accesos serán locales. Dadas las características de la aplicación no resulta necesario implementarla de forma distribuida, con lo que con un acceso local se consigue mayor eficiencia.

- El adaptador con la herramienta UMLtoCSP estará preparado para ejecutarse en una máquina independiente de la del resto de componentes de la aplicación.

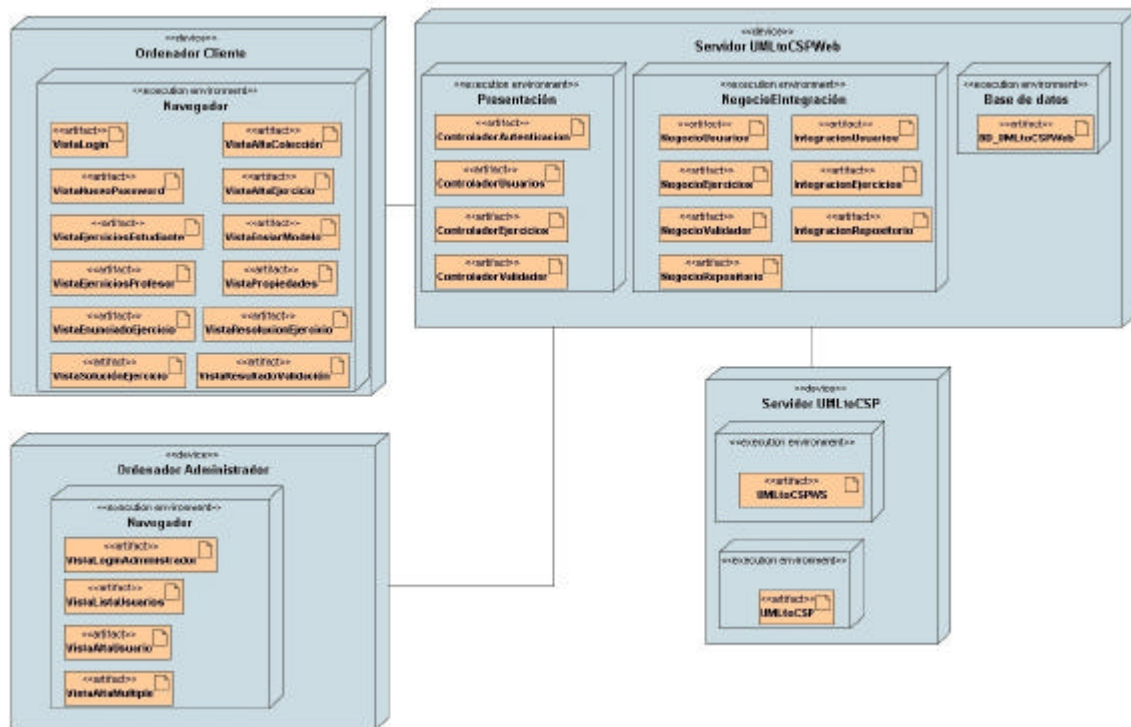


Fig. 50 Diagrama de despliegue

2.1.5 Clases estáticas de diseño.

2.1.5.1 Descripción detallada de las clases de entidad.

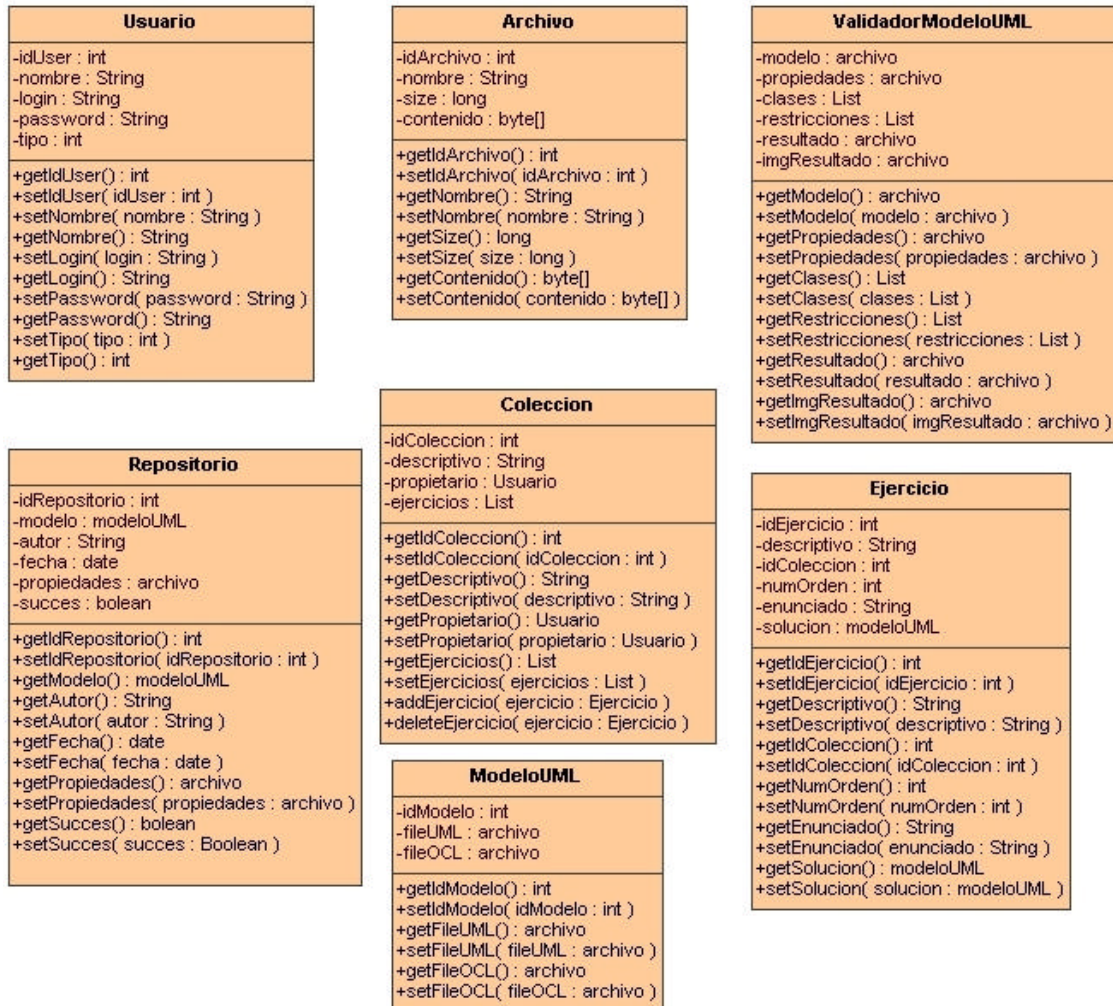


Fig. 51 Diagrama de clases estáticas de entidad

2.1.5.2 Descripción detallada de las clases de gestión e integración.

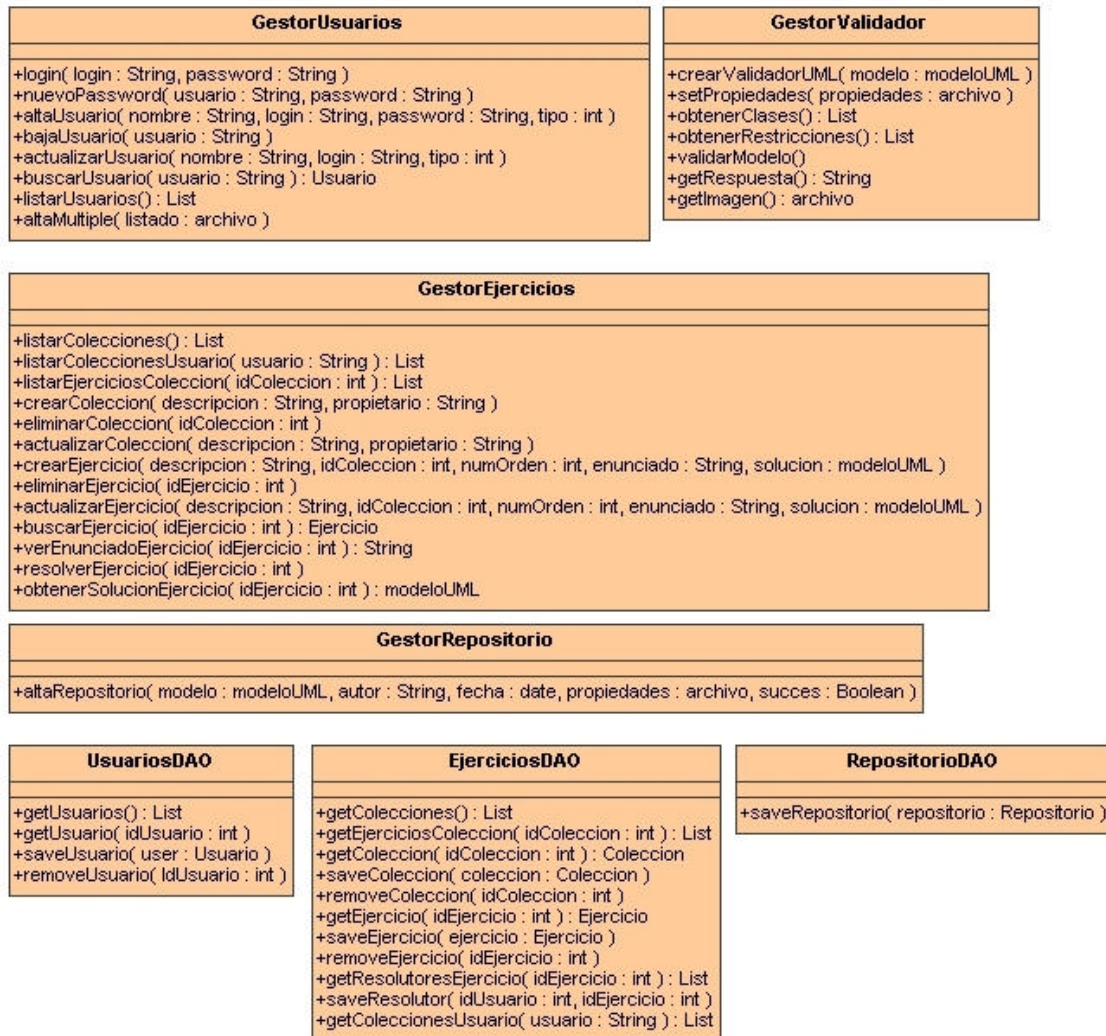


Fig. 52 Diagrama de clases estáticas de gestión e integración

2.2 Diseño de la persistencia.

A continuación se presenta el diagrama de entidad-relación de los datos de la aplicación.

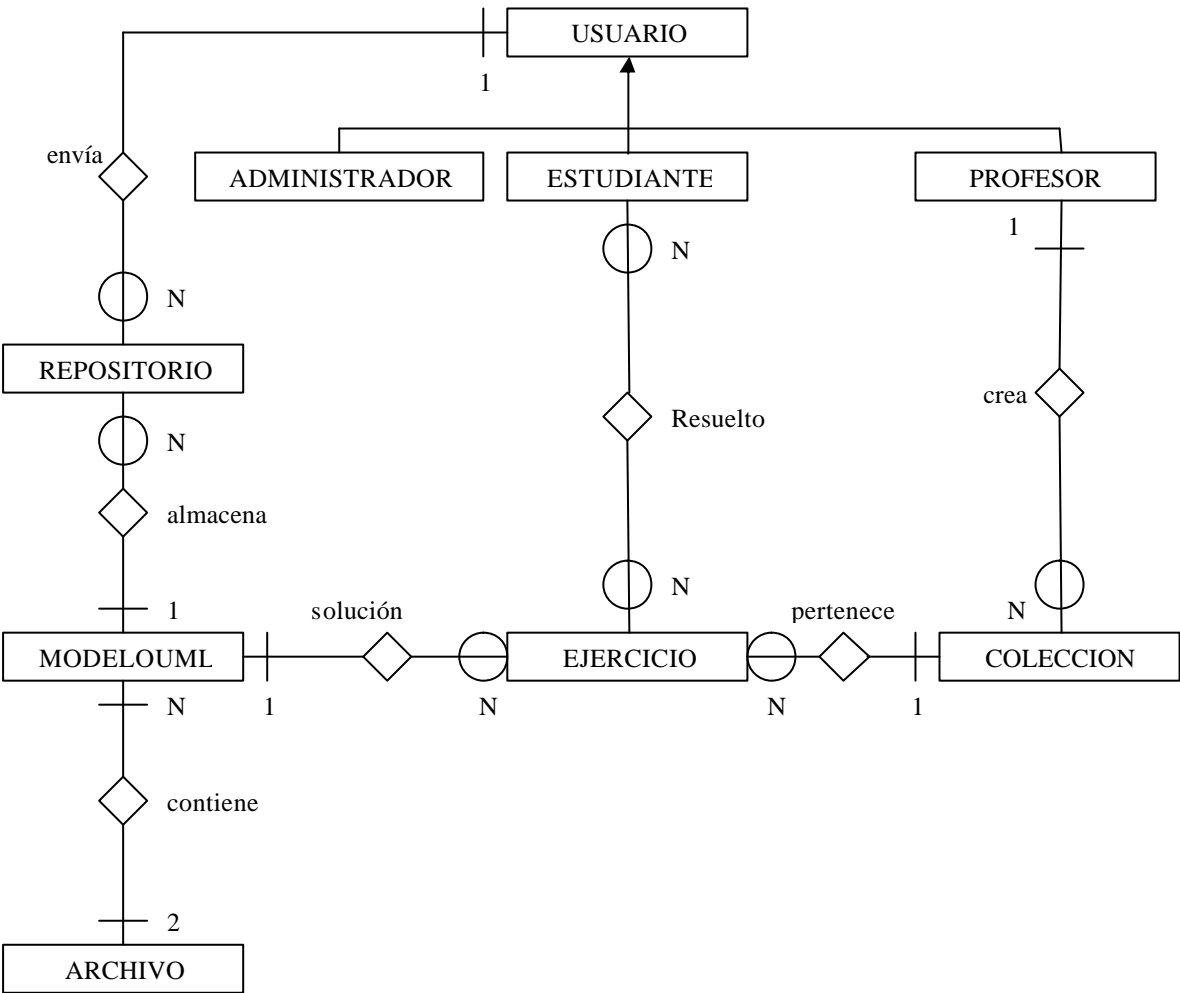


Fig. 53 Diagrama entidad-relación

Y las tablas que utilizaremos:

Tabla usuarios		
login	String	Clave primaria
name	String	
password	String	

Tabla roles		
login	String	Clave primaria
roles	String	

Tabla archivo		
idArchivo	int	Clave primaria
nom	String	
size	int	
contenido	binary	

Tabla modeloUML		
idModelo	int	Clave primaria
<i>idUML</i>	int	Clave foránea a archivo
<i>idOCL</i>	int	Clave foránea a archivo

Tabla coleccion		
idColeccion	int	Clave primaria
descripcion	String	
<i>idPropietario</i>	int	Clave foránea a usuario

Tabla ejercicio		
idEjercicio	int	Clave primaria
descripcion	String	
enunciado	texto	
<i>idColeccion</i>	int	Clave foránea a coleccion
numOrden	int	
<i>idSolucion</i>	int	Clave foránea a modeloUML

Tabla repositorio		
idRepositorio	int	Clave primaria
loginAutor	String	
<i>idModelo</i>	int	Clave foránea a modeloUML
fechaCreacion	date	
<i>idFilePropiedades</i>	int	Clave foránea a archivo
suces	boolean	

Tabla resuelto		
<i>idUsuario</i>	int	Clave foránea a usuario
<i>idEjercicio</i>	int	Clave foránea a ejercicio
idUsuario, idEjercicio		Clave primaria

Usaremos una tabla para almacenar los roles considerando que un usuario pueda tener varios.

2.3 Diseño de la interfaz de usuario.

A continuación se presenta el diseño de la interfaz de usuario. Las pantallas muestran la funcionalidad de la aplicación y una aproximación a su aspecto definitivo.

- **Interfaz pública de la aplicación**

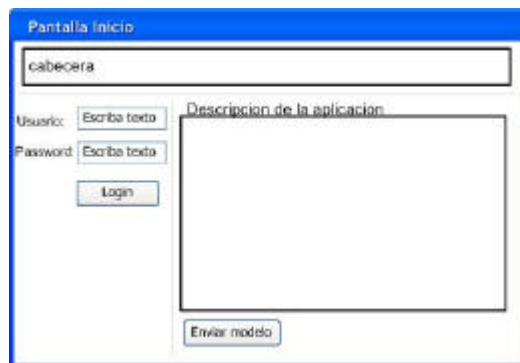
The 'Pantalla Inicio' (Start Screen) features a blue header bar with the title 'Pantalla Inicio'. Below the header is a white box labeled 'cabecera'. To the left of a large 'Descripción de la aplicación' text area, there are login fields: 'Usuario:' and 'Password:' each followed by an 'Escribe texto' input field, and a 'Login' button. At the bottom right of the main content area is an 'Enviar modelo' button.

Fig. 54 Pantalla Inicio

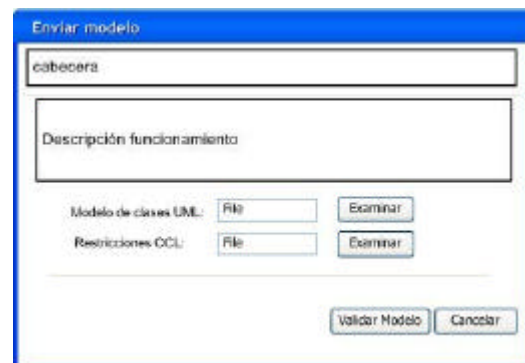
The 'Enviar modelo' (Send Model) screen has a blue header bar with the title 'Enviar modelo'. It includes a 'cabecera' box. Below it is a 'Descripción funcionamiento' text area. Further down are two rows of controls: 'Modelo de clases UML:' with a 'File' button and an 'Eliminar' button, and 'Restricciones OCL:' with a 'File' button and an 'Eliminar' button. At the bottom right are 'Validar Modelo' and 'Cancelar' buttons.

Fig. 57 Pantalla Enviar Modelo

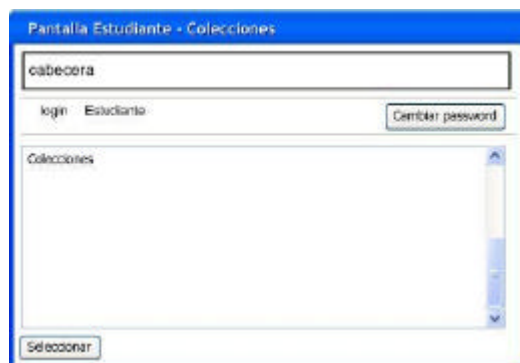
The 'Pantalla Estudiante - Colecciones' (Student Collections) screen has a blue header bar with the title 'Pantalla Estudiante - Colecciones'. It features a 'cabecera' box, a 'login: Estudiante' label, and a 'Cambiar password' button. Below these is a large 'Colecciones' list area with a vertical scrollbar. At the bottom left is a 'Seleccionar' button.

Fig. 55 Pantalla Colecciones estudiante

The 'Pantalla Profesor' (Professor) screen has a blue header bar with the title 'Pantalla Profesor'. It includes a 'cabecera' box, a 'login: Profesor' label, and a 'Cambiar password' button. Below is a large 'Colecciones' list area with a vertical scrollbar. At the bottom left is a 'Seleccionar' button, and at the bottom right are 'Crear', 'Actualizar', and 'Eliminar' buttons.

Fig. 58 Pantalla Colecciones profesor

The 'Cambiar password' (Change Password) screen has a blue header bar with the title 'Cambiar password'. It features a 'cabecera' box and a 'login: Estudiante/Profesor/Administrador' label. Below are three password fields: 'Password anterior:' with an 'Escribe texto' input, 'Password nuevo:' with an 'Escribe texto' input, and 'Confirmar password:' with an 'Escribe texto' input. At the bottom are 'Aceptar' and 'Cancelar' buttons.

Fig. 56 Pantalla cambio password

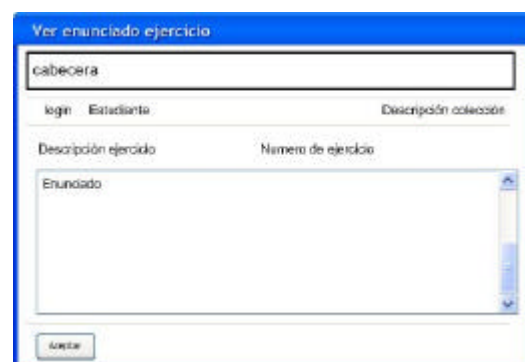
The 'Ver enunciado ejercicio' (View Exercise Statement) screen has a blue header bar with the title 'Ver enunciado ejercicio'. It includes a 'cabecera' box, a 'login: Estudiante' label, and a 'Descripción colección' label. Below are two labels: 'Descripción ejercicio' and 'Número de ejercicio'. A large 'Enunciado' text area with a vertical scrollbar is positioned below these. At the bottom left is an 'Aceptar' button.

Fig. 59 Pantalla ver enunciado

Fig. 60 Pantalla resolver ejercicio

Fig. 64 Pantalla ver solución ejercicio

Fig. 61 Pantalla propiedades modelo

Fig. 65 Pantalla resultado validación

Fig. 62 Pantalla crear colección

Fig. 66 Pantalla crear ejercicio

Fig. 63 Pantalla ejercicios estudiante

Fig. 67 Pantalla ejercicios profesor

- Interfaz privada de la aplicación

Fig. 68 Pantalla Login administrador

Fig. 70 Pantalla inicial administrador

Fig. 69 Pantalla alta múltiple

Fig. 71 Pantalla alta usuario

3 IMPLEMENTACIÓN

3.1 Decisiones de implementación.

El presente proyecto se enmarca dentro del área temática de J2EE.

J2EE es un conjunto de especificaciones creada por Sun para el desarrollo de aplicaciones empresariales distribuidas multicapa basadas en componentes y en el lenguaje Java. Está formado por un conjunto de componentes modulares y estandarizados que ofrecen servicios automatizados para la construcción de aplicaciones distribuidas, la estructura base de las cuales está formada por tres capas:

- Capa cliente: que soporta una gran variedad de tipos de cliente.
- Capa intermedia o de negocio: que contiene dos capas:
 - Capa Web: que atiende las peticiones de los clientes y les retorna un resultado.
 - Capa EJB: que soporta componentes EJB (Enterprise Java Beans).
- Capa de información empresarial (EIS): es la que da soporte a la información existente en las bases de datos.

La tecnología de componentes divide el diseño de una aplicación en capas, que han de ser el máximo de independientes las unas de las otras (desacoplamiento) para facilitar el mantenimiento y disminuir la complejidad. Las mejores prácticas de esta tecnología han sido recogidas en los patrones arquitectónicos. Por ello nos apoyaremos en ellos y en el uso de frameworks para el desarrollo del presente proyecto.

Un framework es una estructura conceptual básica usada para resolver un problema complejo. Así un framework software es un diseño reusable para un sistema de software, creado con la intención de facilitar el desarrollo de ese sistema software y que permite a los diseñadores y programadores pasar más tiempo identificando requerimientos de software que tratando con los tediosos detalles de bajo nivel para proveer un sistema funcional.

Para el desarrollo del proyecto he elegido el **framework Spring**. Se trata de un framework de código abierto creado para reducir la complejidad del desarrollo de aplicaciones empresariales. Hace posible el uso de simples JavaBeans para conseguir cosas que previamente solo eran posibles con EJB's. Spring está formado por varios módulos contruidos sobre un núcleo que ejerce de container y que pueden ser utilizados independientemente según las necesidades de la aplicación a desarrollar.

Este framework se adapta perfectamente a las características de la aplicación a desarrollar ya que es muy ligero, nos permite prescindir de EJB (excesivamente complejo para el caso de que no se necesite acceso remoto) y proporciona una implementación del patrón MVC que resulta muy adecuada para la capa de presentación. Así usaremos Spring para las capas de presentación y negocio.

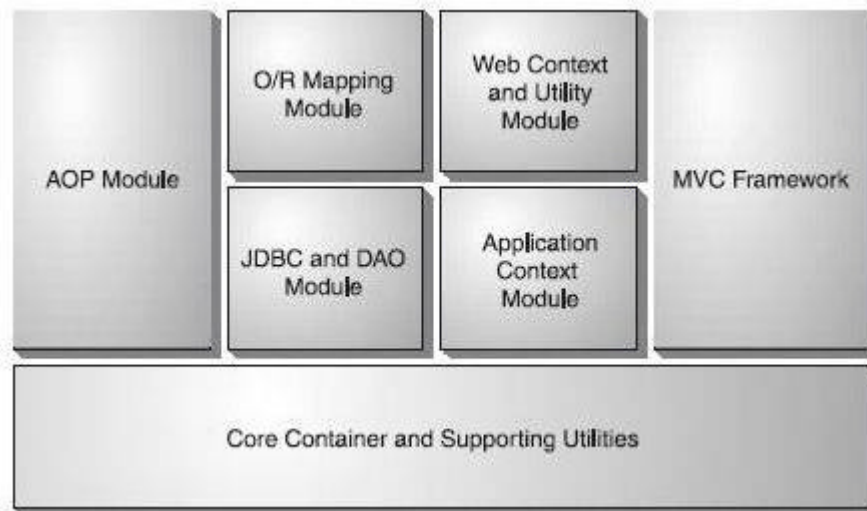


Fig. 72 Framework Spring

Entre las características básicas de Spring y que utilizamos en el presente proyecto cabe mencionar la Inversión de Control (IoC) que funciona como un JNDI inverso en el que el container da las dependencias a un objeto en tiempo de instanciación y la programación orientada a aspectos (AOP) que permite separar la lógica de la aplicación de los servicios del sistema por ejemplo para el manejo de transacciones.

De la seguridad del sistema se ocupará el framework **Acegi Security** que proporciona seguridad declarativa para aplicaciones Spring. Lo usaremos para el proceso de autenticación. Acegi utiliza una serie de filtros que interceptan los request al servlet y fuerzan la autenticación cuando se quiere acceder a un recurso protegido. También se utilizará la capacidad de encriptación de passwords que ofrece el framework.

Para la capa de integración usaremos un framework de mapeo objeto-relacional (ORM): **Hibernate**, que se integra perfectamente con Spring y permite almacenar POJO's en una base de datos de forma sencilla. Hibernate permite especificar el mapeo objeto relacional de dos formas diferentes:

- Mediante archivos descriptores, en formato xml.
- Directamente mediante anotaciones en las clases de entidad.

Usaremos este último enfoque que proporciona mayor claridad a la hora de visualizar el código y su mapeo correspondiente. Además el uso de anotaciones aporta otro beneficio si como será el caso hacemos uso únicamente del estándar JPA (Java Persistence Api) y es que las anotaciones no poseerán ninguna dependencia con Hibernate por lo que resultará fácil en el futuro cambiar Hibernate por cualquier otro ORM que funcione bajo JPA modificando únicamente los archivos de configuración de Spring.

Para la implementación del web service necesario nos apoyaremos en **Apache CXF**, que es un framework para web services de código abierto, fácil de utilizar y que se integra perfectamente con Spring. CXF implementa el API JAX-WS que permite

desarrollar web services simples fácilmente mediante el uso de anotaciones y soporta los dos enfoques de desarrollo: contrato primero (WSDL), o código primero. Se utilizará este último enfoque por su sencillez: a partir del código Java se genera el documento WSDL.

3.2 Software utilizado.

El software utilizado para desarrollar el presente proyecto ha sido el siguiente:

- **Java JDK 6.** Necesario como mínimo el JDK 5 para el soporte de anotaciones.
<http://java.sun.com/javase/downloads/index.jsp>.
- **Apache Tomcat 5.5.26.** Servidor web con soporte de servlets y JSP.
<http://www.coreservlets.com/Apache-Tomcat-Tutorial/tomcat-5.5.html>.
- **Mysql 5.0.51.** Sistema de gestión de base de datos relacional, multihilo y multiusuario bajo licencia GNU GPL.
<http://dev.mysql.com/downloads/mysql/5.0.html>
- **Mysql Administration tools.** Herramientas administrativas para Mysql.
<http://dev.mysql.com/downloads/gui-tools/5.0.html>.
- **Eclipse 3.3.2.** Entorno de desarrollo integrado de código abierto.
<http://www.eclipse.org/downloads/index.php>
- **Spring framework 2.5.3.** Framework de código abierto de desarrollo de aplicaciones para la plataforma Java. <http://www.springframework.org/>
- **Hibernate 3.2.6.** Herramienta de Mapeo objeto-relacional para la plataforma Java. Es un software libre bajo licencia GNU LGPL. <http://www.hibernate.org/6.html>.
- **Acegi Security 1.0.6.** Framework de seguridad altamente integrable con Spring.
<http://www.acegisecurity.org/>.
- **Apache CXF 2.0.4.** framework para web services de código abierto.
<http://cxf.apache.org/download.html>.
- **Displaytag library 1.1.1.** Software libre que proporciona patrones de presentación web. <http://displaytag.sourceforge.net/11/>.
- **Free SMTP Server.** Servidor de SMTP libre para Windows.
<http://www.softstack.com>.
- **UMLtoCSP.** Herramienta para la corrección automática de modelos UML/OCL.
<http://gres.uoc.edu/UMLtoCSP/>.

3.3 Despliegue y configuración.

Los productos obtenidos son dos archivos war: PFC_prevertem.war y UMLtoCSPWS.war que deberán ser desplegados en un servidor web como es el Tomcat.

PFC_prevertem.war contiene el aplicativo web UMLtoCSPWeb con las funcionalidades descritas en apartados anteriores. Será desplegado en la máquina en la que se desee tenerlo operativo.

- **UMLtoCSPWS.war** es el web service creado para la herramienta UMLtoCSP y deberá ser desplegado en la máquina en la que resida dicha herramienta.

3.3.1 Configuración del aplicativo UMLtoCSPWeb.

Las configuraciones necesarias para el funcionamiento del aplicativo se realizan a través del archivo PFC_prevertem\WEB-INF\applicationContext.xml y son las siguientes:

- **Configuración de la base de datos Mysql.** Debe configurarse la dirección y el puerto donde se halla la BBDD así como un usuario y password con privilegios sobre ella. Por defecto viene configurado a localhost en el puerto 3306 con un usuario root sin password.

```
<bean id="dataSource"
    class="org.springframework.jdbc.datasource.DriverManagerDataSource">
    <property name="driverClassName">
        <value>com.mysql.jdbc.Driver</value>
    </property>
    <property name="url">
        <value>jdbc:mysql://localhost:3306/umltoenspweb</value>
    </property>
    <property name="username">
        <value>root</value>
    </property>
    <property name="password">
        <value></value>
    </property>
</bean>
```

- **Configuración del Web Service.** Debe configurarse la dirección y el puerto de la máquina donde reside el Web Service. Por defecto viene configurado a localhost en el puerto 8080.

```
<bean id="umltoenspFactory" class="org.apache.cxf.jaxws.JaxWsProxyFactoryBean">
    <property name="serviceClass" value="uoc.edu.ws.service.UmltoenspService" />
    <property name="address"
        value="http://localhost:8080/UMLtoCSPWS/uoc/edu/ws/service" />
    <property name="serviceFactory" ref="jaxwsAndAegisServiceFactory"/>
</bean>
```

- **Configuración del servidor de SMTP.** Debe configurarse la dirección y el puerto de dicho servidor. Por defecto viene configurado a localhost sobre el puerto 456.

```
<bean id="mailSender" class="org.springframework.mail.javamail.JavaMailSenderImpl">
    <property name="host"><value>localhost</value></property>
    <property name="port"><value>456</value></property>
</bean>
```

También puede configurarse el remitente de los emails automáticos de alta de usuario.

```
<bean id="mailMessage" class="org.springframework.mail.SimpleMailMessage">
    <property name="from"><value><![CDATA[ UMLtoCSPWeb
UMLtoCSPWeb@uoc.edu]]></value></property>
    <property name="subject"><value>Alta usuario UMLtoCSPWeb</value></property>
</bean>
```


- **Configuración del sistema de logs.** Por último debe configurarse en el archivo PFC_prevertem\WEB-INF\classes\log4j.properties el directorio donde se guardarán los logs de la aplicación.

```
log4j.appender.logfile.File=C:/Archivos de programa/apache-tomcat-5.5.26/logs/umltocspweb.log
```

3.3.2 Configuración del Web Service UMLtoCSPWS.

Las configuraciones necesarias para el funcionamiento del web service se realizan a través del archivo PFC_prevertem\WEB-INF\appContext.xml y son las siguientes:

- **Configuración de la herramienta UMLtoCSP.** Debe configurarse el directorio en el que reside el jar de la herramienta.

```
<bean id="umltocspServiceImpl" class="uoc.edu.ws.service.UmltocspServiceImpl">
  <property
name="umltocspDir" ><value>d:/uoc/2008_01/proyecto/umltocsp/umltocsp/umltocsp/lib</value>
</property>
```

- **Configuración del directorio de temporales.** Debe configurarse un directorio para la creación de archivos temporales que utiliza el web service.

```
<bean id="umltocspServiceImpl" class="uoc.edu.ws.service.UmltocspServiceImpl">
  <property
name="umltocspDir" ><value>d:/uoc/2008_01/proyecto/umltocsp/umltocsp/umltocsp/lib</value>
</property>
  <property name="dirTemp" ><value>c:/temp</value> </property>
</bean>
```

3.3.3 Configuración de un entorno para pruebas.

Se puede disponer de un entorno de pruebas, con diversos datos disponibles de la siguiente forma:

En ApplicationContext.xml desactivar el flag de envío de email:

```
<bean id="usuarioManager" class="uoc.edu.umltocspweb.service.UsuarioManagerImpl" >
  <property name="usuarioDao" ref="usuarioDao"/>
  <property name="mailSender"><ref bean="mailSender" /></property>
  <property name="message"><ref bean="mailMessage" /></property>
  <property name="encoder"><ref bean="passwordEncoder" /></property>
  <property name="salt"><ref bean="mySalt" /></property>
  <property name="flagMail"><value>false</value></property>
</bean>
```

Desde Mysql ejecutar el script umltocspweb_creacion.sql, para crear la BBDD y un usuario administrador con login **admin** y password **admin**.

Ejecutar la aplicación y con el usuario admin., seleccionar alta múltiple de usuarios y utilizar el archivo usuarios_pruebas.csv para añadir varios usuarios:

Usuario: profesor1	password: profesor1
Usuario: profesor2	password: profesor2

Usuario: estudiante1 password: estudiante1
Usuario: estudiante2 password: estudiante2

Desde Mysql ejecutar el script umltocspweb_pruebas.sql para añadir diversos datos a la BBDD.

Ahora se dispone de un entorno con 5 usuarios y diversos datos para realizar pruebas.

VALORACION FINAL

Conclusiones

Considero alcanzados los dos objetivos propuestos para el PFC. En primer lugar la ampliación de conocimientos relativos a la temática del área en que se enmarca el trabajo, la tecnología Java y la arquitectura J2EE, temática tan amplia como fascinante y en cuyo conocimiento he dado un paso adelante desde el nivel introductorio que se adquiere con las asignaturas de la UOC hasta un nivel de iniciado capaz de abordar el desarrollo de una aplicación bajo esta plataforma. Y que mejor forma de adquirir estos conocimientos que la de enfrentarse a las dificultades que plantea el segundo objetivo, el diseño y la implementación de un aplicativo web, objetivo que considero alcanzado al disponer de una aplicación plenamente funcional y que puede servir de base para un desarrollo que tenga una explotación real.

Las dificultades no han sido pocas, especialmente debido a la amplitud de la materia, que hace que la elección inicial de un camino adecuado a seguir sea complicada y obliga a dedicar un periodo de tiempo amplio a examinar las diferentes tecnologías existentes en busca de aquellas que resulten aptas a los propósitos del trabajo. No obstante, una vez superados los escollos iniciales, he disfrutado mucho de la realización del proyecto especialmente durante la fase de implementación.

En definitiva, considero que el PFC es una herramienta de aprendizaje útil por la experiencia que se adquiere al enfrentarse a las diferentes fases de desarrollo de un proyecto de envergadura reducida pero real.

Mejoras para futuras versiones

Como con casi todas las aplicaciones quedan muchos aspectos por añadir o mejorar, a continuación cito algunos de los más obvios que tenía previstos pero no he llegado a realizar.

- Utilizar un canal seguro para la transmisión de datos confidenciales como los passwords.
- Añadir un parser para obtener los nombres de las clases de un archivo xmi, como ayuda a la hora de seleccionar las propiedades de un modelo a validar, y mejorar el existente para las restricciones que es muy rudimentario.
- Mejorar la navegabilidad por la lista de usuarios, añadiendo funciones como la búsqueda de usuarios.

GLOSARIO

- **Administrador:** persona que tiene acceso exclusivo a la aplicación encargada de la gestión de usuarios.
- **Archivo xmi:** archivo en formato XML de intercambio de metadatos. Usado para compartir modelos UML entre diferentes herramientas de modelado.
- **Arquitectura J2EE:** modelo de aplicaciones distribuidas en diversas capas o niveles.
- **Base de datos (BBDD):** conjunto de datos que pertenecen al mismo contexto almacenados sistemáticamente para su uso posterior.
- **Capa de presentación:** es la que ve el usuario, presenta la información al usuario. Se comunica únicamente con la capa de negocio.
- **Capa de datos:** donde residen los datos. Recibe solicitudes de almacenamiento o recuperación de información desde la capa de negocio.
- **Capa de negocio:** donde reside la lógica de la aplicación. Recibe peticiones del usuario y envía las respuestas después de su procesamiento.
- **DAO:** acrónimo de Data Access Object, componente de software que suministra una interfaz común entre la aplicación y uno o más dispositivos de almacenamiento de datos, tales como una Base de datos o un archivo.
- **Diagrama de colaboración:** diagrama que muestra las interacciones entre objetos.
- **Diagrama estático:** muestra el conjunto de clases y objetos importantes que forman parte de un sistema y las relaciones existentes entre ellos.
- **Diagrama E-R:** herramienta para el modelado de datos de un sistema de información. Expresan entidades relevantes para un sistema de información, sus interrelaciones y sus propiedades.
- **Diagrama de Gantt:** diagrama que permite seguir el curso de cada actividad, con las fechas de inicio y final y el tiempo total requerido para la ejecución de la actividad.
- **Diagrama de secuencia:** muestra las interacciones entre objetos ordenadas en secuencia temporal.
- **Diseño arquitectónico:** Diseño de la arquitectura de un sistema.
- **EJB:** acrónimo de Enterprise Java Beans, forma parte del estándar de construcción de aplicaciones empresariales J2EE.
- **Encriptar:** técnica por la cual la información se hace ilegible para terceras personas. Para poder acceder a la información es necesaria una clave.
- **Estudiante:** Role de la aplicación UMLtoCSPWeb que identifica a un usuario que puede acceder a las colecciones de ejercicios.
- **Framework:** estructura de soporte definida donde otro proyecto de software puede ser organizado y desarrollado.
- **Herencia:** mecanismo que permite derivar una clase de otra, de forma que extiende su funcionalidad.
- **Java:** lenguaje para el desarrollo de aplicaciones exportables a la red y capaz de operar sobre cualquier plataforma.

- **JPA**: acrónimo de Java Persistence API, API de persistencia desarrollada para la plataforma Java que busca unificar la manera en que funcionan las utilidades que proveen un mapeo objeto-relacional.
- **JSPs**: extensión de los Servlets que permite de forma fácil la fusión de código con paginas HTML estandar.
- **Login**: Autenticación para acceder a un servicio.
- **Modelo UML/OCL**: Modelo de clases representadas por un diagrama UML y anotadas con un archivo de restricciones OCL.
- **MVC**: patrón de arquitectura que define la organización independiente del Modelo (objetos de negocio), la Vista (interfaz de usuario) y el Controlador (control del flujo de trabajo de la aplicación).
- **MySQL**: sistema de gestión de base de datos, multihilo i multiusuario.
- **OCL**: acrónimo de Object Constraint Language, es un lenguaje declarativo para describir reglas que aplican a modelos UML.
- **ORM**: acrónimo de Object-Relational Mapping. Técnica de programación para convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y el utilizado en una base de datos.
- **Patrón de diseño**: son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y en otros ámbitos referentes al diseño de interacción o interfaces.
- **Profesor**: Role de la aplicación UMLtoCSPWeb que identifica a un usuario que puede crear colecciones de ejercicios.
- **Proxy**: un programa o dispositivo que realiza una acción en representación de otro.
- **Repositorio**: sitio centralizado donde se almacena y mantiene información digital. En el contexto de UMLtoCSPWeb utilizado para almacenar los modelos UML presentados a validación.
- **Servlets**: es una clase Java que puede cargarse dinámicamente para extender la funcionalidad de un servidor web.
- **SGBDs relacionales**: implementación de base de datos relacionales que soporta todos los aspectos del modelo relacional incluidos los dominios y las dos reglas generales de integridad.
- **SMTP**: acrónimo de Simple Mail Transfer Protocol, Protocolo de red basado en texto utilizado para el intercambio de mensajes de correo electrónico entre computadoras o distintos dispositivos.
- **TCP**: acrónimo de Transmission Control Protocol, protocolo de comunicación orientado a conexión y fiable del nivel de transporte usado en Internet.
- **UML**: siglas de lenguaje unificado de modelo, lenguaje gráfico para visualizar, especificar, construir i documentar un sistema de software.
- **Usuario**: cualquier persona que se conecta a través de Internet al aplicativo UMLtoCSPWeb.
- **Web Service**: un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones.
- **WSDL**: acrónimo de Web Services Description Language, un formato XML que se utiliza para describir servicios Web. describe la interfaz pública a los servicios Web.

BIBLIOGRAFIA

Libros.

- **Spring in Action**
Craig Walls, Ryan Breidenbach.
Manning Publications Co
- **Agile Java Development with Spring, Hibernate and Eclipse.**
Anil Hemrajani
Sams Publishing
- **Professional Java Development with the Spring Framework.**
Rod Jonson, Juergen Hoeller, Alef Arendsen, Thomas Risberg
Colin Sampaleanu
Wiley Publishing, Inc
- **Beginning Spring 2: From Novice to Professional.**
Dave Minter
Apress
- **Pro Spring.**
Rob Harrop, Jan Machacek
Apress
- **Beginning Hibernate: From Novice to Professional.**
Dave Minter, Jeff Linwood
Apress
- **Hibernate in Action**
CHRISTIAN BAUER, GAVIN KING
Manning Publications Co

Artículos.

[Developing a Spring Framework MVC application step-by-step](#)
[The Spring Framework - Reference Documentation](#)
[Acegi Security in one hour](#)
[Spring Acegi Tutorial](#)
[Developing J2EE Applications Using Hibernate Annotations and Spring](#)
[Web Services with Spring 2.5 and Apache CXF 2.0](#)
[Send E-mail Using Spring and JavaMail](#)
[Creating a Java, Eclipse, Tomcat, MySQL Environment](#)