UNIVERSIDAD DE CANTABRIA

Facultad de Ciencias

Proyecto Fin de Carrera

Desarrollo de una Aplicación para la Gestión de Expedientes de los Alumnos de Intercambio de la Escuela Universitaria de Enfermería

(Development of an Application for the Management of Exchange Students' Records in the School of Nursing)

Para acceder al Titulo de

INGENIERO EN INFORMÁTICA

Autor: Antonio Cuenca González

Marzo - 2013

Facultad de Ciencias

INGENIERÍA EN INFORMÁTICA

CALIFICACIÓN DEL PROYECTO FIN DE CARRERA

Realizado por: Antonio Cuenca González
Director del PFC: María Pérez Madrazo
Ponente: Pablo Sánchez Barreiro
Título: "Desarrollo de una aplicación para la Gestión de Expedientes de los Alumnos de Intercambio de la Escuela Universitaria de Enfermería"
Title: "Development of an Application for the Management of Exchange Students' Records in the School of Nursing"

Presentado a examen el día:

para acceder al Título de

INGENIERO EN INFORMÁTICA

Composición del Tribunal:
Presidente (Apellidos, Nombre): González Harbour, Michael
Secretario (Apellidos, Nombre): Martínez Fernández, María del Carmen
Vocal (Apellidos, Nombre): Menéndez de Llano Rozas, Rafael
Vocal (Apellidos, Nombre): Sánchez Barreiro, Pablo
Vocal (Apellidos, Nombre): Sanz Gil, Roberto

Este Tribunal ha resuelto otorgar la calificación de: .....

Fdo.: El Presidente	Fdo.: El Secretario
Fdo.: Vocal	Fdo.: Vocal
Fdo.: Vocal	Fdo.: El Director del PFC

II

Resumen

El objetivo de este proyecto es desarrollar un Sistema de Información que permita la gestión de los expedientes de los alumnos de programas de intercambio dentro de la Escuela Universitaria de Enfermería, a requerimiento del personal de administración y servicios de la secretaría de dicha Escuela, el cual gestionaba hasta ahora dichos expedientes de forma manual. Con objeto de automatizar y facilitar dicha gestión, se desea construir un sistema software que permita la administración de todos los datos necesarios para llevar a cabo esta gestión. Por ejemplo, deberá permitir dar de alta universidades extranjeras, centros asociados a dichas universidades así como asignaturas impartidas en dichos centros a ser consideradas dentro de los procesos de convalidación entre asignaturas.

La función principal de la aplicación debe ser la de permitir elaborar, de la forma más cómoda posible para el personal administrativo, planes de convalidación detallados por cada alumno. Dichos planes de convalidación deben especificar las asignaturas cursadas por el alumno en la universidad de destino durante el programa de intercambio y, por cada una de estas asignaturas, qué asignatura de la universidad origen del alumno se desea convalidar. Además, la aplicación deberá permitir exportar los expedientes de los alumnos a diversos formatos, tales como PDF.

La aplicación, debido a ciertas restricciones técnicas, se desarrollará como una aplicación de escritorio distribuida, utilizando MySQL como sistema gestor de base de datos y siguiendo una arquitectura en tres capas que favorezca la evolución y adaptación de la misma.

Palabras Clave: Sistemas de Información, Arquitecturas Software en Tres Capas, Gestión de Expedientes, MySQL

IV

Preface

The aim of this project is to develop an Information System that allows the management of students'records who are enrolled in exchange programmes within the School of Nursing, at the request of the management staff and the secretarial services of the school, who managed these records manually until now. In order to automate and facilitate the management, the system must enable the administration of all the data necessary to carry out this management. For example, the system must allow to register foreign universities, centers associated with these universities as well as subjects taught in these centers to be considered within the processes of recognition among subjects.

The main goal of the application must be to allow the creation of detailed recognition plans for each student in the most suitable way for administrative staff. Such recognition plans must specify the subjects studied by students in the University destination during the exchange program, and for each of these subjects, which subject of the student's origin university wants to be recognised. In addition, the application must allow to export records from students to various formats, such as PDF.

The application, due to technical restrictions, will be developed as a distributed desktop application that communicates remotely using MySQL as a database management system. The system will be develop using a three-layer architecture which favours its evolution and adaptation.

Keywords: Information Systems, Three-Layer Software Architecture, Record Management, MySQL

VI

Índice general

Índice de figuras

Capítulo 1

Introducción

Este capítulo sirve de introducción al presente documento. Describe, a grandes rasgos, los objetivos generales que el proyecto debía satisfacer, así como la estructura del presente documento.

Índice

1.1. Introducción

El objetivo de este proyecto es desarrollar un sistema de información para la gestión de los alumnos de intercambio dentro de la Escuela Universitaria de Enfermería de la Universidad de Cantabria. Los administrativos de dicha escuela realizaban la gestión de toda la información generada por estos alumnos, tales como matrículas o expedientes, de forma completamente manual.

La gestión manual y en papel de cantidades más o menos voluminosas de información tiene asociada los consabidos problemas que el desarrollo de sistemas de información pretende solucionar:

- Realizar búsquedas de forma manual en volúmenes de información de cierto tamaño puede requerir de un tiempo elevado. Si estas mismas búsquedas se realizan de forma computerizada, los tiempos de búsqueda disminuyen drásticamente, a la par que el sistema se vuelve mucho más escalable.

- Los archivos digitales ocupan un volumen de varias órdenes de magnitud menor que el de los archivos físicos.

- Los archivos digitales son más fácilmente compartibles por diferentes personas.
- Los archivos digitales son más fácilmente accesibles desde diferentes localizaciones geográficas.
- Los archivos digitales son más fácilmente duplicables que los archivos físicos, por lo que es más sencillo obtener copias de respaldo de dichos archivos, de forma que se pueda recuperar la información almacenada en caso de que, por motivo de algún accidente, como por ejemplo, una inundación, los archivos originales resultasen dañados y no pudieran recuperarse.

Dado el creciente número de alumnos de intercambio, con la consiguiente escalada de la información asociada a ser gestionada, la Secretaría de la Escuela Universitaria de Enfermería de la Universidad de Cantabria decidió llevar a cabo la informatización de ciertos procesos administrativos relacionados con la gestión de los mismos. Concretamente, los procesos que debían ser informatizados eran:

- Gestión de la información asociada a universidades origen y destino de los estudiantes de intercambio.
- Gestión de los datos personales de los alumnos de intercambio.
- Gestión de los datos personales de los profesores de intercambio.
- Gestión de los planes de convalidación que realizarán los alumnos de intercambio.

La siguiente sección describe la planificación que se siguió para desarrollar un proyecto que satisfaciese los objetivos generales antes mencionados.

1.2. Planificación del Proyecto

Esta sección describe la metodología de desarrollo que se utilizó para la realización del proyecto cuyos objetivos generales se describieron en la sección anterior. Como metodología de desarrollo se utilizó una variación del modelo incremental [26]. Dicha metodología se ilustra en la Figura 1.1, y se describe a continuación.

De acuerdo con la Figura 1.1, la primera etapa a realizar fue determinar las funciones que debía cumplir el sistema de acuerdo con las necesidades del cliente. Para ello se realizó un análisis y especificación de requisitos. El objetivo de esta etapa fue producir una especificación de requisitos detallada que describiera de forma precisa todas las funcionalidades que el sistema debía proporcionar.

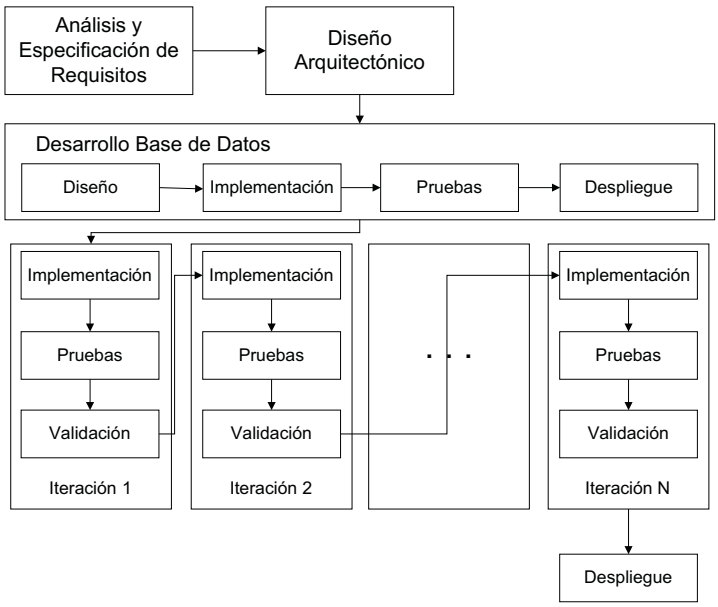

Figura 1.1: Metodología de Desarrollo Incremental

Utilizando esta especificación de requisitos se procedió a definir una arquitectura de alto nivel del sistema. El objetivo era dividir el sistema en varios componentes que pudieran ser desarrollados y testados de forma independiente. Además, dicha arquitectura debía permitir una fácil evolución del sistema, por lo que debía satisfacer unos requisitos mínimos y bien definidos de modularidad, de forma que ciertas partes de dicha arquitectura fueran fácilmente modificables sin tener que alterar otras partes de la misma.

Partiendo de dicho diseño arquitectónico, obtenido en la etapa anterior, se pasó al diseño e implementación de cada una de sus partes constituyentes. Al tratarse de un sistema de información, la parte dedicada al almacenamiento persistente de los datos es una pieza clave y fundamental en torno a la cual giran y se organizan el resto de los componentes del sistema. Por tanto, dimos prioridad al desarrollo de esta parte del sistema.

Dado que parecía claro, de acuerdo a las necesidades del cliente, que dicho almacenamiento persistente debía ser una base de datos relacional, diseñamos esta parte del sistema de acuerdo a un enfoque de desarrollo de bases de datos relacionales [5] [8]. Para ello, en primer lugar, diseñamos la base de datos mediante un modelo entidad-relación [25] [27]. A continuación, transformamos dicho modelo entidad-relación en una implementación del mismo en un sistema gestor de bases de datos relacional y realizamos una serie de pruebas sistemáticas para comprobar la consistencia de dicha base de datos. Seguidamente, creamos las clases DAO (*Data Access Objects*) [9] que permitían gestionar los datos almacenados en dicha base de datos des-

de código y realizamos las pruebas pertinentes para asegurar el correcto funcionamiento de dichas clases. Por último, desplegamos tanto las clases DAO como la base de datos en un servidor bien definido con el que poder comunicarnos durante el resto del proceso de desarrollo.

Con la base de datos desplegada comenzó un ciclo de iteraciones en las que se fueron implementando y probando las distintas funcionalidades del sistema. Al final de cada iteración se realizaba una validación con el cliente para comprobar que todo funcionaba correctamente. Al tratarse de un sistema centrado principalmente en la la mera manipulación de datos (altas, bajas, listados y modificaciones), no hizo falta aplicar grandes estrategias de diseño a nivel funcional, por lo que esta etapa en muchas ocasiones simplemente se omitió.

Finalmente, una vez completadas todas las iteraciones, se procedió al despliegue y la puesta en funcionamiento operativo del sistema.

Se optó por utilizar esta metodología frente a un modelo incremental en espiral puro, donde todas las fases del ciclo de vida desde la especificación de requisitos al despliegue se repetirían en cada iteración, por las siguientes razones:

1. El número de requisitos de la aplicación no es excesivo.

2. Dichos requisitos son sobradamente conocidos de antemano.

3. Dichos requisitos además son estables, no existiendo el peligro de que cambien a medida que se desarrolla el proyecto.

4. Los sistemas de información de este tipo se desarrollan, salvo contadas excepciones, siguiendo un estilo arquitectónico en tres capas [10], por lo que el diseño arquitectónico también es bastante estable.

Por tanto, tanto requisitos como arquitectura se pudieron definir sin demasiados problemas al principio del proyecto, evitando los problemas de gestión de la configuración asociados a los desarrollos en espiral puro [26]. Además, al ser la base de datos un elemento pivotal en torno al cual giran el resto de los elementos de una sistema de información, la definimos antes de diseñar e implementar el resto de componentes de la arquitectura.

La siguiente sección explica cómo se ha estructurado el resto de la presente memoria de forma que se pueda detallar el proceso descrito en esta sección para que su lectura resulte lo más amena y gratificante posible al lector.

1.3. Estructura del Documento

Tras esta introducción, el presente documento se estructura como sigue:

El siguiente capítulo estará dedicado a la ingeniería de requisitos y al diseño arquitectónico. Este capítulo explicará el proceso de captura y especificación de requisitos, tanto funcionales como no funcionales para, a continuación, detallar su modelado y especificación. Por último se describirá el diseño arquitectónico utilizado y las justificaciones de esta decisión.

Seguidamente se describirán detalladamente los procedimientos, tanto de diseño como de implematación, seguidos para desarrollar la base de datos del sistema. Tras esto se describirán las pruebas que se realizaron a la base de datos y que nos permitieron comprobar su correcto funcionamiento. Seguidamente se detallarán los procedimientos de desarollo y pruebas de las clases de accesos a datos. Finalmente se describirirá el proceso seguido para el despliegue de nuestra base de datos que permitiría que quedara disponible para la siguiente fase.

A continuación se explicará como se llevó a cabo el proceso de desarrollo de las capas de presentación y de negocio. Se explicará el proceso iterativo que se siguió mediante el cual se iban añadiendo funcionalidades con cada iteración a través de una explicación más detallada de dos de las iteraciones realizadas. Seguidamente se mostrará como se desplegó la aplicación una vez que todas las iteraciones fueron completadas. En última instancia se mostrará un ejemplo de como es posible realizar un cambio de la interfaz sin que el resto del sistema se vea afectado.

Finalmente, en el capítulo que sirve de sumario a este documento, se detallará todo el proceso realizado a modo de resumen. Seguidamente se detallarán las conclusiones obtenidas tras la realización de este trabajo y por último se hablará sobre posible trabajos futuros y como estos podrán ser llevados a cabo.

Capítulo 2

Ingeniería de Requisitos y Diseño Arquitectónico

En este capítulo se describe cómo se realizó el proceso de captura y especificación de requisitos, tanto funcionales como no funcionales. Se muestra, para algunos de los requisitos capturados, cómo han sido modelados y cómo han sido especificados. Además, se describe el diseño arquitectónico realizado, justificando el por qué de dicho diseño.

Índice

2.1. Introducción

El presente capítulo describe con detalle las dos primeras fases de nuestro proyecto de acuerdo a la metodología descrita en el capítulo anterior. Estas fases eran generales para todo el proyecto y no se realizaron de forma incremental, dado que las características del proyecto aconsejaban hacerlas de una sola vez, y no por incrementos, de cara a facilitar la gestión de la configuración (ver Sección 1.2). Dichas fases eran el *Análisis y Especificación de Requisitos* y el *Diseño Arquitectónico*.

En el caso de la primera fase, el primer problema a resolver es que técnica se iba a utilizar para realizar la captura de requisitos. Se trataba de extraer de las fuentes de información disponibles información precisa y fiable acerca de lo que tenía que hacer nuestro sistema. El siguiente paso era estructurar y

especificar de acuerdo a alguna técnica suficientemente conocida dicha información de forma que se obtuviese al final del proceso una *Especificación de Requisitos* que determinase de forma clara, precisa y no ambigua, qué debía hacer nuestro sistema. Para ello, se especificaron los requisitos en forma de casos de uso [4]. Esta fase de análisis de requisitos se complementó con un análisis acerca de cómo diversos requisitos no funcionales afectaban a nuestro sistema.

Partiendo de esta primera fase, se procedió a diseñar la arquitectura del sistema. Tal como se comentó en el capítulo anterior, al tratarse de un sistema de información básico, no había demasiado lugar para la creatividad del arquitecto software. La aplicación se diseñaría en base a un modelo arquitectónico en tres capas. Este modelo arquitectónico es el utilizado por la mayoría de los sistemas de información y su objetivo es facilitar la evolución del sistema, tratando de minimizar el impacto de ciertos cambios sobre la arquitectura. Por ejemplo, siguiendo este modelo arquitectónico, migrar el sistema gestor de bases de datos utilizado sólo debería afectar a una parte muy concreta de la arquitectura, permaneciendo el resto de la arquitectura estable.

Tras esta introducción, este capítulo se estructura como sigue: La sección 2.2 describe el proceso de captura de requisitos. La sección 2.3 muestra como los requisitos capturados se especificaron mediante la técnica de casos de uso. A continuación, la Sección 2.4 analiza la influencia de una serie de requisitos no funcionales bien conocidos sobre nuestro sistema. Por último, la Sección 2.5 describe el proceso de diseño de la arquitectura, sirviendo la Sección 2.6 de sumario y cierre al capítulo.

2.2. Captura de Requisitos

Esta sección describe el proceso llevado a cabo para identificar los requisitos que debía satisfacer nuestra aplicación. Dicho proceso de captura de requisitos se llevó a cabo mediante la técnica de *análisis de documentos* y *análisis de procesos de negocio* [22]. El cliente proporcionó una serie de documentos que representaban los documentos a informatizar por nuestra aplicación. Dichos documentos constituían la fuente de información para especificar los requisitos que debía satisfacer nuestra aplicación con respecto al almacenamiento de información.

Además, diversos administrativos escenificaron en diversas ocasiones el proceso de creación y aprobación de un plan de estudios, de forma que pudimos comprender y especificar el proceso que debían seguir los documentos a informatizar, desde que se crean hasta que acaba su vida útil.

A partir de los documentos y de las especificaciones, se identificaron una serie de requisitos que debía satisfacer nuestra aplicación. La Figura 2.1 muestra el conjunto de requisitos completo que debía satisfacer la aplicación.

Referencia	Requisito
R01	Se tiene que almacenar, consultar y modificar los datos de las universidades con las que hay o ha habido un convenio.
R02	Se tiene que almacenar, consultar y modificar los datos de los centros asociados a una universidad.
R03	Se tiene que almacenar, consultar y modificar los datos de las asignaturas impartidas dentro de un centro.
R04	Se tiene que almacenar, consultar y modificar los datos personales de los alumnos.
R05	Se tiene que almacenar, consultar y modificar lo s datos personales de los profesores que realicen un intercambio.
R06	Para cada alumno, se deberá poder definir un *plan de estudios*. Un plan de estudios detalla a nivel individual las convalidaciones de asignaturas entre la universidad de origen y la de destino.
R07	Adicionalmente algunos alumnos podrán convalidar prácticas por lo que la aplicación debe ser capaz de incluirlas en el plan de estudios.
R08	Las prácticas se gestionan mediante una descripción de las tareas realizadas en el centro de origen y en el centro de destino, el número de créditos ECTS que le corresponden y la nota que se obtenga.
R09	Por cada universidad, se deben gestionar los siguientes datos: Nombre y si existe convenio actualmente o no.
R10	Por cada centro, se deben gestionar los siguientes datos: Nombre y la universidad a la que pertenece.
R11	Por cada asignatura, se deben gestionar los siguientes datos: Nombre, semestre en el que se imparte, curso y número de créditos ECTS.
R12	Por cada alumno, se deberán gestionar los siguientes datos personales: Nombre completo, documento de identificación, año académico, email y email auxiliar, teléfono y teléfono auxiliar, centro al que pertenece, centro en el que realizará el intercambio, el programa de intercambio que está utilizando y un campo de observaciones.
R13	Por cada profesor, se deberán gestionar los siguientes datos personales: Nombre completo, documento de identificación, año académico, el semestre en el que realiza la visita , email y email auxiliar, teléfono y teléfono auxiliar, centro al que pertenece, centro en el que realizará el intercambio y un campo de observaciones.
R14	Para crear un *plan de estudios*, la aplicación deberá visualizar las asignaturas en las que se encuentra matriculado el alumno de intercambio y que puede convalidar.
R15	Para crear un *plan de estudios*, por cada asignatura del plan origen, se debe poder elegir la asignatura del plan destino por la cual se desea convalidar la asignatura del plan origen.
R16	Por motivos administrativos, la consistencia de los pares de convalidación asignatura origen-destino se comprobará manualmente. Es decir, la aplicaci ón debe permitir establecer pares de convalidación tales como *Introducción a la Filosofía – Álgebra*, aunque en principio pudiesen carecer de sentido.
R17	El sistema no debe permitir la eliminación de datos almacenados.
R18	La aplicación debe poder ser ejecutada en cualquiera de los ordenadores de la Escuela.
R19	Debe establecerse una política de copias de seguridad de la base de datos
R20	Las personas autorizadas deberán acceder a la aplicación mediante su nombre de usuario y su contraseña.
R21	La aplicación debe generar un fichero en formato pdf con el plan de estudios detallado de cada alumno.

Figura 2.1: Requisitos identificados para nuestra aplicación

Una vez definidos de forma sistemática los requisitos a satisfacer por parte de nuestra aplicación, pasamos a especificarlos de forma más detallada y a elaborar los correspondientes modelos de requisitos que permitiesen la definición de la arquitectura del sistema, de acuerdo con la metodología expuesta en la Sección 1.2.

2.3. Especificación y Modelado de Requisitos

Esta sección describe el proceso por el cual se especificaron y modelaron los requisitos tras su captura, la cual fue descrita en la sección anterior. Este procedimiento de especificación y modelado se llevo a cabo mediante la técnica de *casos de uso* [16].

Los casos de uso se organizaron en diferentes niveles de granularidad, de acuerdo con la jerarquía propuesta por Cockburn [4]. Dicha jerarquía asocia a cada caso de uso un color dependiendo de si describen aspectos generales o específicos del sistema a desarrollar. Dado que el lector podría no estar familiarizado con dicha jerarquía, la describimos a continuación, de mayor a menor nivel de abstracción.

Casos de uso blancos En el primer nivel, el más abstracto, se encuentran los objetivos estratégicos del sistema. Estos objetivos son los casos de uso más generales de la aplicación. Por ejemplo, en nuestro caso un caso de uso blanco sería *Gestionar Universidades*.

Casos de uso azules Tras este primer nivel se encuentra el nivel donde se describen los objetivos concretos de usuario. Un caso de uso azul estará asociado con un caso de uso blanco. Por ejemplo, un caso de uso azul sería *Alta Universidad*, el cual estaría asociado al caso de uso blanco *Gestionar Universidades*. Este caso de uso blanco podría tener asociados otros casos de uso azules, tales como *Baja Universidad* o *Listar Universidades*

Casos de uso negros Es el nivel más bajo. Se refiere a casos de uso que describen detalles técnicos sobre la implementación de los casos de uso. Por ejemplo, en una caso de uso de nivel negro se describiría el proceso por el cual al dar de alta un formulario web, este formulario web cursa una petición AJAX que sería a su vez atendida por un servicio web tipo REST. Estos tipos de casos de uso, aunque aparecen en documentos técnicos reales, no deberían ser considerados como casos de uso, ya que ofrecen un nivel excesivo de detalle para lo que se considera adecuado a un nivel de especificación de requisitos.

Tras clarificar cómo funciona la clasificación propuesta por Cockburn, exponemos los diferentes casos de uso tanto de nivel blanco como azul que

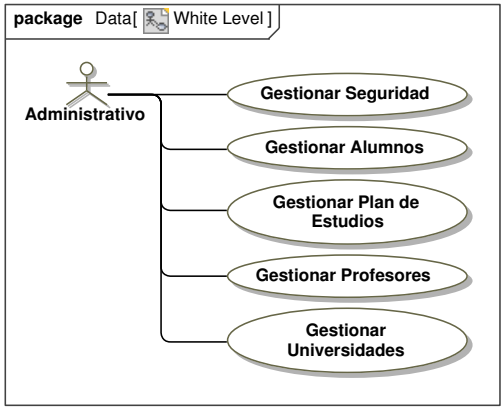

Figura 2.2: Casos de uso de blancos identificados

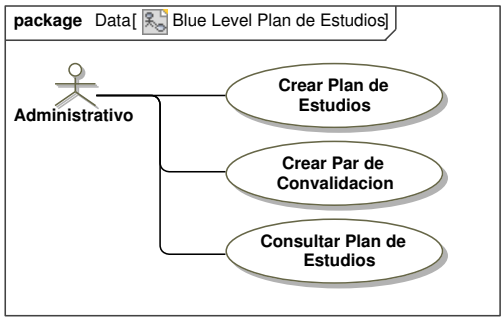

Figura 2.3: Casos de uso azules para *Gestionar Plan de Estudios*

hemos identificamos para nuestro sistema en particular. La Figura 2.2 muestra los casos de uso blancos, mientras que las Figuras 2.3 y 2.4 muestran los casos de uso azules asociados a los casos de uso blancos *Gestionar Plan de Estudios* y *Gestionar Alumnos*. Por ejemplo, para el caso de *Gestionar Alumnos*, de acuerdo con los requisitos capturados en la sección anterior (ver Figura 2.1, requisito `R04`), la aplicación debía soportar como casos de uso concretos: *Alta Alumno*, *Modificar Alumno*, *Consultar Alumno* y *Listar Alumnos*. Los datos concretos a gestionar por cada alumno se especificaron mediante un diagrama entidad-relación, el cual se esbozó durante la fase de Ingeniería de Requisitos y se terminó de especificar completamente durante la fase de desarrollo de la base de datos (Dicho diagrama se muestra en la Figura 3.1).

Una vez identificados los casos de uso que debía soportar nuestra aplicación, el siguiente paso era especificarlos a un nivel suficiente de detalle para

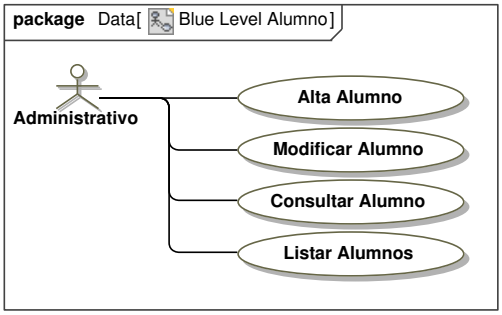

Figura 2.4: Casos de uso azules para *Gestionar Alumnos*

Caso de Uso	Crear Plan de Estudios
Actores	Administrativo
Descripción	Se crea un nuevo plan de estudios para un alumno determinado
Precondiciones	1. El administrativo debe estar identificado en el sistema. 2. El administrativo está en la interfaz gráfica para la administración de alumnos. 3. El alumno no posee plan de estudios creado.
Flujo Principal	4. El administrativo selecciona un alumno. 1. El administrativo selecciona la opción *Crear Plan de Estudios*. 2. Se crea un *Plan de Estudios* vacio para el alumno seleccionado. 3. Se abre la interfaz gráfica para la gestión de planes de de estudio y se muestra el plan de estudios (vacío) creado.
Postcondiciones	1. El nuevo plan de estudios queda almacenado en el sistema para su posterior consulta o modificación.

Figura 2.5: Plantilla para el caso de uso *Crear Plan de Estudios*

que se pudiese llevar a cabo el diseño e implementación de la aplicación. Para ello especificamos los casos de uso mediante plantillas. Las Figuras 2.5 y 2.6 muestran las plantillas para los casos de uso *Crear Plan de Estudios* y *Crear Par de Convalidación*.

Caso de Uso	Crear Par de Convalidación
Actores	Administrativo
Descripción	Se crea un nuevo par de convalidación para un alumno determinado
Precondiciones	1. El administrativo debe estar identificado en el sistema. 2. El administrativo está en la interfaz gráfica para la administración de planes de estudio.
Flujo Principal	1. El administrativo selecciona la opción de *Nueva Entrada*. 2. La interfaz muestra una lista con las posibles asignaturas de origen y las posibles asignaturas de destino, de acuerdo a los centros de origen y destino del alumno. 3. El administrativo selecciona una asignatura de origen. 4. El administrativo selecciona una asignatura de destino. 5. El administrativo selecciona la opción de *Guardar*.
Flujos Alternativos	1. Si el administrativo selecciona la opción de *Cancelar*, se cierra la interfaz gráfica para la administr ación de planes de estudio, sin realizar ningún cambio en el sistema, y regresando a la interfaz gráfica de la cual procedíamos.
Postcondiciones	1. El nuevo par de convalidación queda almacenado en el sistema.
Aclaraciones	De acuerdo con lo solicitado por el cliente, el sistema *no* deberá chequear la consistencia de los pares de convalidaci ón, ya que esta consistencia debe haber sido comprobado con anterioridad en otros procesos administrativos.

Figura 2.6: Plantilla para el caso de uso *Crear Par de Convalidación*

2.4. Análisis de Requisitos no Funcionales

Tras identificar, modelar y especificar los requisitos funcionales que debía satisfacer nuestra aplicación, la siguiente tarea era realizar un análisis de la influencia de los diversos requisitos no funcionales comúnmente existentes para nuestra aplicación concreta. Esta sección contiene dicho análisis.

Para analizar la influencia de los requisitos no funcionales, nos basamos en la taxonomía de requisitos no funcionales proporcionados por la norma ISO 25010 [15]. Esta norma define un modelo de calidad que incluye características sobre aspectos tanto internos como externos y de uso de un sistema software. Dicho modelo de calidad se refleja en una serie de requisitos no funcionales que afectan en mayor o menor medida a la mayoría de los siste-

mas software. Analizamos a continuación la influencia de tales requisitos no funcionales en nuestro sistema:

Corrección Funcional Cada operación debe asegurar que a su finalización se mantiene la integridad de los datos almacenados en el sistema, incluso cuando las operaciones terminan debido a situaciones excepcionales.

Rendimiento No existen restricciones especiales en relación con el rendimiento y el uso de recursos del sistema. No se prevén necesidades de cómputo complejas ni un tamaño de la base de datos excesivo. De hecho, la cantidad de datos a almacenar se considera bastante reducida en comparación con las capacidades del hardware actuales. No obstante, las operaciones relacionadas con la interfaz de usuario deberán tener el tiempo de respuesta propio de los sistemas interactivos. Es decir, el sistema debe ofrecer algún tipo de respuesta al usuario antes de que éste pueda empezar a considerar que el sistema se encuentra bloqueado.

Usabilidad El sistema debe permitir al usuario realizar las tareas especificadas de la manera más eficiente posible, reduciendo lo máximo posible la navegación a través de la interfaz para realizar las tareas más comunes. De igual forma, cada interfaz deberá ofrecer un número acotado de elementos de forma que la cantidad de información a procesar por el usuario final sea asumible para éste.

Usabilidad La curva de aprendizaje debe ser lo más pequeña pequeña posible.

Fiabilidad La fiabilidad del sistema debe ser la normal de cualquier sistema software. No se trata de un sistema software de alta disponibilidad que tenga que estar funcionando 24 horas al día los 7 días de la semana. El sistema puede permitirse pequeñas caídas, siempre y cuando el tiempo de recuperación entre caídas sea pequeño, nunca superior al tiempo que se tarda en reiniciar un computador o un servidor. En caso de un fallo grave, como pueda ser que el servidor de bases de datos sufra un fallo hardware, tiene que ser posible restaurar el sistema de forma completa.

Seguridad Sólo podrán acceder al sistema y a los datos almacenados los usuarios que estén autorizados para ellos.

Privacidad Los datos almacenados deberá cumplir lo dispuesto en la Ley Oficial de Protección de Datos y el Esquema Nacional de Seguridad, de aplicación para todas las administraciones públicas.

Evolución El sistema debe ser lo más modular posible de forma que pueda adaptarse a posibles evoluciones tanto en los procesos de convalidación

como en el entorno donde se despliega la aplicación. En especial, es de esperar que la aplicación acabe evolucionando hasta convertirse en una aplicación web cuya interfaz puede integrarse con las interfaces proporcionados por otros sistemas software para la gestión universitaria.

Una vez conocidos y analizados los diversos requisitos, tanto funcionales como no funcionales que debe satisfacer nuestro sistema, podemos proceder al diseño de su arquitectura, de acuerdo con el proceso de desarrollo mostrado en la Figura 1.1.

2.5. Diseño Arquitectónico

De acuerdo con nuestra metodología (ver Figura 1.1), tras identificar y especificar los requisitos que debía satisfacer nuestra aplicación, el siguiente paso era definir la arquitectura de nuestro sistema software. Esta sección explica en primer lugar el estilo arquitectónico elegido, justificándolo convenientemente. A continuación, muestra el diseño arquitectónico realizado.

2.5.1. Estilo Arquitectónico

El estilo arquitectónico elegido fue, como no podía ser de otra manera, el de una arquitectura en tres capas [10]. La justificación básica es que dicho esquema es el utilizado por una amplísima mayoría de sistemas de información como el que pretendemos desarrollar. Ello se debe a que este estilo proporciona una organización modular que ha demostrado ser muy resistente a los cambios típicos que acontecen dentro de un sistema de información. A continuación detallamos dicha organización modular.

El objetivo principal de una organización arquitectural en tres capas es separar en módulos bien definidos las responsabilidades o lógica asociada a tres aspectos fundamentales de todo sistema de información: (1) presentación, (2) lógica de procesamiento o negocio; y (3) persistencia.

Estas capas se organizarían de acuerdo a la disposición expuesta en la Figura 2.7. Cada capa sólo se comunica con la capa adyacente a través de interfaces bien definidas. Este estilo arquitectónico funcionaría tal como se describe a continuación.

En primer lugar, la capa de presentación tiene como responsabilidad principal mostrar y gestionar la interfaz al usuario. Dependiendo de cada sistema, esta interfaz puede ser una interfaz web, una interfaz gráfica de escritorio o una interfaz gráfica de aplicación móvil, tales como Android [20] o iOS [21]. Dicha interfaz se puede utilizar tanto para introducir datos en el sistema como para visualizar datos de salida. En el caso de que se vayan a introducir datos, es responsabilidad de la capa de presentación comprobar que dichos datos sean correctos antes de enviarlos a la capa de negocio para

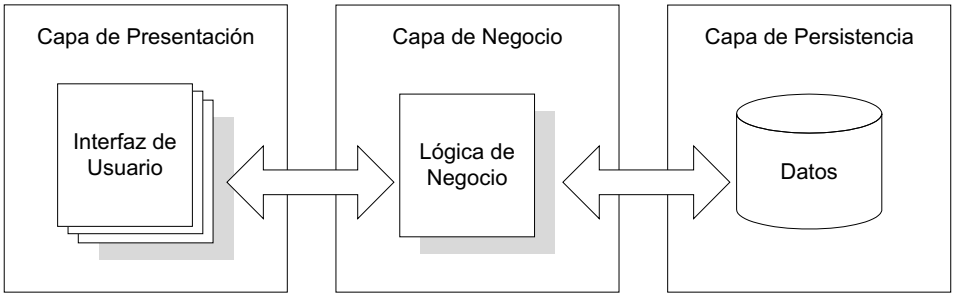

Figura 2.7: Arquitectura de tres capas

su procesamiento. Por ejemplo, en nuestro caso, la fecha de nacimiento de un alumno debería ser, al menos, una fecha perteneciente al pasado (se entiende que el alumno no puede haber nacido dentro de 2 años, por ejemplo). Una vez comprobada la validez de los datos, se pasarían a la capa de negocio para que proceda a su procesamiento. En el caso de la visualización de datos, la capa de presentación tiene como responsabilidad mostrar la información devuelta por el sistema de la forma más amigable posible al usuario. Además, es la encargada de manipular y reordenar los datos de acuerdo a las necesidades de cada usuario. Por ejemplo, la capa de presentación podría inicialmente mostrar una lista de alumnos ordenados alfabéticamente. No obstante, la interfaz podría ofrecer al usuario la posibilidad de visualizar dichos datos ordenados por fecha de nacimiento de cada alumno. En este caso, la reordenación de dicha lista se llevaría a cabo dentro de la capa de presentación, sin necesidad de invocar a la capa de negocio.

La capa de negocio es la que contiene la lógica principal de la aplicación. Es la que implementa los procesos de negocio que gobiernan cómo se deben manipular los diversos datos que almacena el sistema. Por ejemplo, la lógica de negocio es la que debe comprobar que los pares de convalidación creados son correctos, o cuando es posible añadir un nuevo par de convalidación, así como determinar cuando un plan de convalidación puede ser cerrado. La capa de negocio recibe peticiones de la capa de presentación y se encarga de su procesamiento. Para procesar dicha petición, puede necesitar recuperar datos de la base de datos, o alterar el estado de la misma. Por ejemplo, si desde la capa de presentación realizamos una petición para añadir un nuevo par de convalidación, deberemos, tras comprobar en las reglas de negocio que dicha acción es factible, dar de alta este nuevo par de convalidación. La capa de negocio puede también necesitar consultar los datos almacenados por la aplicación, ya sea para devolverlos a petición del usuario, o porque necesite conocer ciertos datos con objeto de decidir si una petición de la capa de presentación puede realizarse o no, de acuerdo con las reglas de negocio definidas.

Existen multitud de formas de almacenar los datos de una aplicación. La solución más natural es usar una base de datos relacional [8]. No obstante, existen otras opciones, como XML [3], bases de datos orientadas a objetos [14] o los modernos sistemas NoSQL [24]. Incluso en el caso de optar por la solución clásica de usar un sistema gestor de bases de datos relacional, existen diferentes opciones, ya que hay múltiples sistemas gestores de bases de datos relacionales disponibles en el mercado, cada uno con sus pequeñas diferencias. El objetivo principal de la capa de persistencia es aislar al resto de la aplicación de la forma concreta en la cual almacenamos los datos de una aplicación. Para ello se crean una serie de interfaces bien definidas, denominadas *DAO (Data Access Objects)* que se encargan de definir las operaciones *CRUD (Create, Read, Update, Delete)* que podemos realizar sobre los datos de nuestro sistema.

Normalmente se define una interfaz DAO por cada clase o elemento que forme parte del dominio de nuestra aplicación. Por ejemplo, si tenemos una clase *Alumno* en el dominio de nuestra aplicación, crearemos una interfaz denominada *AlumnoDao* que contenga las operaciones necesarias para crear, borrar, modificar, listar, buscar (con diversos criterios) y borrar alumnos. Por cada tipo de almacenamiento concreto, se implementa dicha interfaz de acuerdo a las particularidades de dicho sistema de almacenamiento. De esta forma, los cambios en el sistema de almacenamiento no afectarían a la aplicación, ya que la capa de negocio seguiría accediendo a los datos a través de la misma interfaz, aunque cambie la implementación de la misma.

De igual forma, cualquier cambio en la apariencia externa de la aplicación sólo afectaría a la capa de presentación. Resulta relativamente cómodo, por ejemplo, evolucionar de una interfaz de escritorio a una interfaz web gracias al uso de una arquitectura de tres capas. Cómo se puede realizar este cambio se mostrará en la Sección 4.5. De la misma forma, si nuestra base de datos se tuviese que integrar en un futuro con una base de datos mayor ya existente dentro de los servicios de informática de la universidad, posiblemente distribuida, dicho cambio sólo afectaría a la capa de persistencia. Y en esta misma línea, cualquier modificación de las políticas de convalidación de asignaturas sólo tendría repercusión en la capa de negocio, no afectando ni a la presentación de los datos ni a su forma de almacenamiento.

Resumiendo, las razones por las cuales se ha optado por este estilo arquitectónico son:

1. Es un modelo muy estable y ampliamente utilizado en el desarrollo de sistemas de información.

2. Permite una fácil evolución de la capa de presentación, por lo que nuestra aplicación podría integrarse fácilmente en el conjunto de aplicaciones web ofrecidas por los servicios de informática de la Universidad de Cantabria.

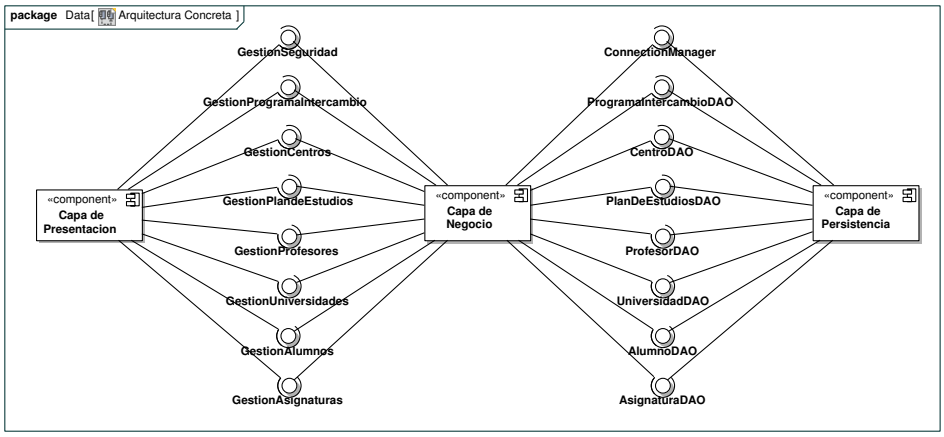

Figura 2.8: Arquitectura concreta de nuestro sistema

Listado 2.1: Clase *GestionAlumnos*

```
1 public interface GestionAlumnos {
2
3   public int nuevoAlumno(Alumno a);
4
5   public List<Alumno> getAlumnos ();
6
7   public Alumno getAlumno(String documentoId);
8
9   public int modificaAlumno(String documentoIdAntiguo, Alumno nuevo);
10 }
```

3. Permite una fácil evolución de la capa de persistencia, permitiendo que la base de datos actual sea reemplazada en el futuro por cualquier otra fuente de datos proporcionada por los servicios informáticos de la Universidad de Cantabria. Igualmente, si se decidiese que la utilización de un sistema gestor de bases de datos relacional resultase muy pesado para la cantidad de datos almacenar, podríamos sustituir dicho gestor por soluciones más ligeras, como las modernas NoSQL.

Una vez seleccionado el estilo arquitectónico, el siguiente paso fue diseñar la arquitectura de nuestro sistema, lo cual se explica en la siguiente subsección.

2.5.2. Arquitectura Concreta del Sistema

La Figura 2.8 muestra el diseño arquitectónico concreto para nuestra aplicación. Es una arquitectura simple, con un componente por cada capa. Entre la capa de presentación y negocio se ha definido una interfaz por cada

Listado 2.2: Clase *AlumnoDAOInterfaz*

```
1  public interface AlumnoDAOInterfaz {
2
3    public int nuevoAlumno(Alumno a);
4
5    public int modificaAlumno(String documentoIdAntiguo, Alumno nuevo);
6
7    public List<Alumno> getAlumnos();
8
9    public Alumno getAlumno(String documentoId);
10
11 }
```

caso de uso de alto nivel identificado en la fase de Ingeniería de Requisitos (ver Sección 2.3). Entre la capa de negocio y presentación se ha definido una interfaz DAO por cada elemento persistente perteneciente al dominio de nuestra aplicación. La definición de estas interfaces se ha realizado de forma iterativa junto con el diseño de la base de datos, ya que es en dicho proceso de diseño donde se terminaron de definir qué elementos persistentes debía manejar nuestra aplicación.

El Listado 2.1 muestra como ejemplo la interfaz *GestionAlumnos* proporcionada por la capa de negocio. Esta interfaz ofrece todas las operaciones necesarias para la manipulación de los datos de los alumnos que se corresponden con los casos de uso azules identificados anteriormente y que pueden verse en la Figura 2.4. De esta manera tenemos el método *nuevoAlumno* que nos permite dar de alta un alumno en el sistema, disponemos también del método *getAlumnos* que nos proporciona una lista con todos los alumnos del sistema. Si queremos consultar un alumno concreto lo haremos con el método *getAlumno* y finalmente si queremos modificar los datos de un alumno ya existente disponemos del método *modificarAlumno*.

Por otra parte, el Listado 2.2 proporciona un ejemplo de interfaz DAO para la manipulación de *Alumno*. Esta interfaz nos ofrece las mismas operaciones que en el caso anterior. Esto es debido a que la lógica de la capa de negocio es sencilla y actúa como pasarela entre las capas de presentación y persistencia y por lo tanto no requiere de operaciones especiales con la base de datos.

2.6. Sumario

En este capítulo se ha descrito cómo se realizó el proceso de captura y especificación de requisitos, tanto funcionales como no funcionales. Además, se ha mostrado, para algunos de los requisitos capturados, cómo han sido modelados y cómo han sido especificados. Además, se ha descrito el diseño arquitectónico realizado, justificando el por qué de dicho diseño. El siguiente

capítulo describe el proceso de desarrollo de la base de datos de nuestra aplicación, el cual constituye la siguiente fase de nuestro proyecto, de acuerdo a la metodología expuesta en la Sección 1.2.

Capítulo 3

Desarrollo de la Base de Datos

Este capítulo describe cómo se realizaron las distintas fases del desarrollo de la base de datos del sistema. Más concretamente, se explica como se llevó a cabo su diseño y su implementación. También se describen las pruebas realizadas para comprobar el correcto funcionamiento de la misma. A continuación, se mostrarán los procesos de desarrollo y pruebas de las clases de acceso a datos para, finalmente, mostrar el proceso realizado para su despliegue.

Índice

3.1. Introducción

Un sistema de información es un sistema que proporciona a ciertos usuarios la información precisa en el momento que la necesitan. Por tanto, el elemento fundamental o pieza fundamental de todo sistema de información son sus datos, más concretamente, el mecanismo a través del cual el sistema almacena ciertos datos de forma persistente. Este capítulo describe con detalle cómo se desarrolló dicho mecanismo de almacenamiento persistente. En nuestro caso, elegimos como mecanismo de almacenamiento persistente una base de datos relacional.

La razón fue que las bases de datos relacionales son la tecnología más adecuada y madura para la mayoría de los sistemas. Sólo se descartaría en casos especiales donde la recuperación de la información obedece a parámetros de búsqueda muy concretos, por ejemplo, en el almacenamiento de datos geoespaciales, cuando el volumen de datos a almacenar es muy reducido o cuando los recursos de los que disponemos son limitados, como en el caso de ciertos sistemas empotrados. En nuestro caso, el volumen de datos a manejar no parece inicialmente muy elevado, pero dado que nuestro sistema tiene un cierto carácter archivístico, es decir, nunca se borran datos, es de esperar que el volumen de datos comience a ser considerable con el paso de los años. En este caso, parece conveniente aprovechar el rendimiento que ofrecen los sistemas gestores de bases de datos para manipular grandes volúmenes de información.

Esta sección describe el desarrollo de dicha base de datos conforme a la metodología ya expuesta en la Figura 1.1. Actualmente los procesos de desarrollo de bases de datos son suficientemente maduros y conocidos. La primera fase de este desarrollo fue la realización de un modelo conceptual de alto nivel, independiente del modelo relacional, que especifique qué datos debemos almacenar, las características de dichos datos, y cómo se relacionan entre ellos. Dicho modelo conceptual se construye utilizando un modelo entidad-relación extendido [25] [27]. Tras realizar este modelo conceptual, se aplica un conocido algoritmo de transformación de entidad relación a relacional y se traduce el modelo relacional a un conjunto de sentencias SQL que permiten crear el esquema de bases de datos deseado.

Una vez creada la base de datos la sometimos a varias pruebas sistemáticas para comprobar su correcto funcionamiento. Las pruebas estaban orientadas a comprobar que se podían realizar las operaciones CRUD sobre los diferentes elementos que habían sido identificados a nivel conceptual, así como que la base de datos era capaz de mantener la integridad de sus datos. Por ejemplo, que si un centro estaba asociado a una universidad, dicha universidad existía.

A continuación, creamos las clases que implementaban las interfaces DAO para completar la capa de persistencia y procedimos a definir y realizar de forma sistemática las pruebas necesarias para comprobar el correcto funcionamiento de estas clases de acceso a datos. Con esto quedaba completada la capa de persistencia de nuestra arquitectura.

Finalmente se procedió al despliegue de la base de datos y las clases de acceso a datos para su utilización durante el desarrollo del sistema. Esta fase de despliegue incluía tanto la configuración del sistema gestor de bases de datos como del sistema de copias de seguridad destinado a satisfacer el requisito no funcional de disponibilidad. En caso de que el servidor de bases de datos resultase irremediablemente dañado, por ejemplo, por un incendio, habría siempre una copia de respaldo que permitiese restaurar el sistema.

En adelante, este capítulo se estructura de la siguiente manera: La Sec-

ción 3.2 describe el proceso de creación del modelo entidad-relación extendido y su posterior transformación e implementación. A continuación, la Sección 3.3 describe cómo se realizó el proceso de prueba de la base de datos. La Sección 3.4 describe el proceso de desarrollo de las clases de acceso a datos, así como su proceso de prueba. En la Sección 3.5 se explica el proceso seguido para el despliegue de la base de datos, sirviendo la Sección 3.6 de sumario y cierre a este capítulo.

3.2. Diseño e implementación de la base de datos

Esta sección describe el proceso del diseño e implementación de la base de datos relacional que utilizará nuestra aplicación. El diseño de la base de datos se realizó, como ya se ha comentado, mediante un diagrama entidad-relación. Para la implementación de la base de datos se decidió utilizar *MySql* [6] [7], debido a que se trata de un sistema gestor de bases de datos relacionales [5] [8] muy utilizado actualmente, maduro, estable, gratuito, con amplia documentación disponible y con el cual teníamos experiencia previa. Para realizar el modelo de entidad-relación, se decidió utilizar *DBDesigner*, una herramienta de modelado integrada con *MySql* y que posee además capacidades de Ingeniería Directa para generar automáticamente el código SQL correspondiente a un modelo entidad-relación.

Para realizar el modelo entidad-relación extendido se partió del documento de especificación de requisitos creado en la fase anterior del proceso de desarrollo. No obstante, cuando surgieron dudas o la información del documento de especificación de requisitos resultaba insuficiente, se procedió a revisar las fuentes de información que se utilizaron para capturar los requisitos con objeto de adoptar las decisiones adecuadas. La Figura 3.1 muestra el modelo entidad-relación que se produjo como resultado de esta de diseño. Se han suprimido los atributos de las entidades, dejando sólo sus claves primarias y externas, para facilitar la visualización de dicho diagrama.

Como puede verse en el diagrama, cada alumno tiene asignado un programa de intercambio. Además, el alumno tiene asignados dos centros, el de origen y el de destino. Cada centro pertenece a una universidad concreta. Obviamente, uno de los centros, ya sea el de origen o el de destino, ha de ser la Escuela Universitaria de Enfermería de la Universidad de Cantabria. En cada centro, a su vez, se imparten una serie de asignaturas. Los alumnos pueden convalidar asignaturas o prácticas. Ambos elementos poseen mecanismos de convalidación diferenciados. Por tanto, cada alumno tiene asociadas las prácticas que desee convalidar, si hay alguna, y un plan de estudios, que define las asignaturas que quiere convalidar. Un plan de estudios contiene un conjunto de pares de asignaturas. En cada par de asignaturas, una asignatura representa la asignatura cursada mientras que la otra es la asignatura por la que se desea convalidar. Por último los profesores que realicen

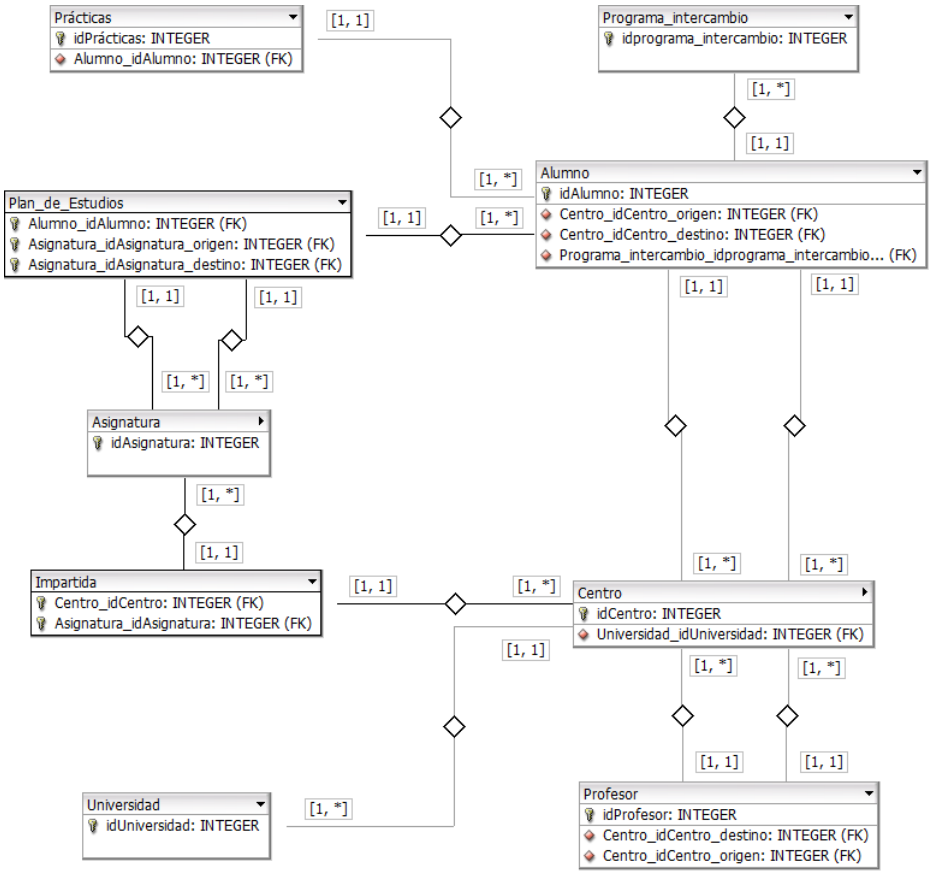

Figura 3.1: Modelo Entidad-Relación

intercambios tienen asignados, de igual manera que los alumnos, un centro de origen y otro de destino. La Figura 3.2 muestra las entidades *Alumno* y *Asignatura* de forma detallada, especificando los atributos que conforman cada entidad.

Como comentamos, el proceso de transformación de dicho modelo entidad-relación en un modelo relacional y en sus correspondientes sentencias SQL, puede realizar de forma automática utilizando las facilidades proporcionadas por la herramienta de diseño integrada en *MySQL DBDesigner*. Utilizando estas facilidades, generamos el script SQL para la creación del esquema de base de datos que correspondía a nuestro modelo. Parte de dicho script se muestra en el Listado 3.1.

Con dicho script, estábamos listos para crear un esquema de base de datos sobre el cual realizar las pruebas necesarias para comprobar que era capaz de dar soporte a las operaciones requeridas sobre los datos a almacenar, así como mantener la integridad de los datos del sistema.

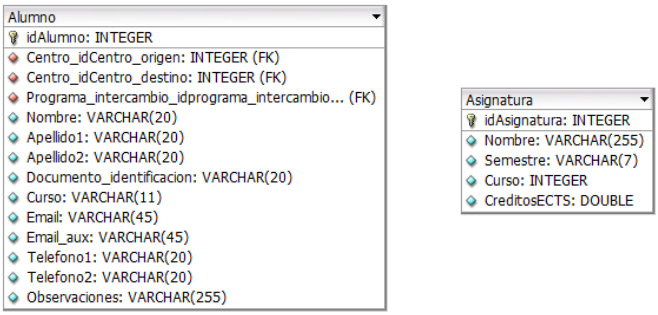

Figura 3.2: Tablas Alumno y Asignatura

Listado 3.1: Ejemplo del script de creación de la Base de Datos

```
1  CREATE DATABASE IF NOT EXISTS 'intercambio';
2
3  USE 'intercambio';
4
5  CREATE TABLE 'alumno' (
6    'idAlumno' int(10) unsigned NOT NULL auto_increment,
7    'Centro_idCentro_origen' int(10) unsigned NOT NULL,
8    'Centro_idCentro_destino' int(10) unsigned NOT NULL,
9    'programa_intercambio_idprograma_intercambio' int(10) unsigned NOT
         NULL,
10   'Nombre' varchar(20) NOT NULL,
11   'Apellido1' varchar(20) NOT NULL,
12   'Apellido2' varchar(20) default NULL,
13   'Documento_identificacion' varchar(20) NOT NULL,
14   'Curso' varchar(11) NOT NULL,
15   'email' varchar(45) NOT NULL,
16   'email_aux' varchar(45) default NULL,
17   'telefono1' varchar(20) NOT NULL,
18   'telefono2' varchar(20) default NULL,
19   'observaciones' varchar(255) default NULL,
20   PRIMARY KEY  ('idAlumno'),
21   UNIQUE KEY 'Documento_identificacion' ('Documento_identificacion'),
22   KEY 'Alumno_FKIndex1' ('programa_intercambio_idprograma_intercambio')
         ,
23   KEY 'Alumno_FKIndex2' ('Centro_idCentro_origen'),
24   KEY 'Alumno_FKIndex3' ('Centro_idCentro_destino')
25 );
```

3.3. Pruebas de la base de datos

Una vez creado el esquema de base de datos, procedimos a testearlo. Esta sección describe dicho proceso. Como hemos comentado, la base de datos es la pieza fundamental de un sistema de información, ya que va a ser la encargada de almacenar todos los datos del sistema. Por este motivo, es necesario prestar especial atención a estas pruebas.

Las pruebas se realizaron mediante un procedimiento de caja blanca, desarrollando una serie de pruebas unitarias implementadas como sentencias SQL, las cuales se almacenaron en un script de prueba. Este script comprueba que para cada tabla, usando datos válidos, puedan crearse, modificarse y leerse diferentes elementos. Dado que en nuestro sistema nunca va a ser necesario borrar elementos, no se realizaron pruebas que permitiesen comprobar que es posible eliminar los datos almacenados. Además, se comprobó que para cada tipo de dato inválido, el propio sistema gestor de bases de datos impedía la ejecución de tales comandos.

Por tanto, el proceso de pruebas se divide en dos partes diferenciadas: la prueba del funcionamiento con datos correctos, y la prueba del comportamiento ante datos incorrectos. El funcionamiento del proceso de pruebas para datos correctos es el siguiente:

1. Inicialmente se introducen una serie de datos válidos en la base de datos. Con esto aseguramos que se pueden dar de alta datos en el sistema, siempre y cuando se den ciertas condiciones. Por ejemplo, para dar de alta un nuevo centro, precisamos de que exista al menos una universidad a la cual asociar dicho centro.

2. A continuación, listamos cada tabla para comprobar que: (1) se pueden listar los elementos; y (2) que los elementos están correctamente almacenados.

3. Tras comprobar que los datos han sido correctamente almacenados, procedimos a modificar cada campo que de cada tabla para comprobar que era posible modificar los datos. Posteriormente, volvimos a listar cada tabla para comprobar que los datos se habían modificado correctamente.

Por otra lado, para comprobar el comportamiento ante datos incorrectos, se llevó a cabo el siguiente proceso.

1. Por cada campo de cada tabla, se intentó introducir un dato incorrecto. Por ejemplo, allí donde debía aparecer un entero, se introducía una cadena de caracteres.

2. Por cada tabla, se comprobaron las restricciones de unicidad intentando introducir datos duplicados.

3. Por cada tabla, se comprobaron las restricciones de integridad referencial intentando introducir como clave externa valores que no correspondían a elementos dados de alta en el sistema.

El Listado 3.2 muestra, a modo de ejemplo, un trozo del script de pruebas utilizado.

Listado 3.2: Ejemplo del script de pruebas de la Base de Datos

```
1 /*inserciones correctas comprobando campos nulos y únicos*/
2 insert into ‘alumno‘ values (’1’, ’3’, ’2’, ’1’, ’Nombre1’, ’Apellido1’
      , ’Apellido21’, ’11111111A’, ’2012 / 2013’, ’nombre1@alumnos.unican
      .es’, ’’, ’666112233’, ’’, ’’);
3 insert into ‘alumno‘ values (’2’, ’1’, ’2’, ’1’, ’Nombre2’, ’Apellido2’
      , ’’, ’11111111B’, ’2012 / 2013’, ’nombre2@alumnos.unican.es’, ’
      apellido2@unican.es’, ’666111222’, ’942111222’, ’observaciones’);
4
5 /*volvemos a insertarlos para comprobar que no lo permite*/
6 insert into ‘alumno‘ values (’1’, ’3’, ’2’, ’1’, ’Nombre1’, ’Apellido1’
      , ’Apellido21’, ’11111111A’, ’2012 / 2013’, ’nombre1@alumnos.unican
      .es’, ’’, ’666112233’, ’’, ’’);
7 insert into ‘alumno‘ values (’2’, ’1’, ’2’, ’1’, ’Nombre2’, ’Apellido2’
      , ’’, ’11111111B’, ’2012 / 2013’, ’nombre2@alumnos.unican.es’, ’
      apellido2@unican.es’, ’666111222’, ’942111222’, ’observaciones’);
8
9 /*insertamos repitiendo un valor único ya introducido*/
10 insert into ‘alumno‘ values (’3’, ’1’, ’2’, ’1’, ’Nombre3’, ’Apellido3’
      , ’’, ’11111111B’, ’2012 / 2013’, ’nombre2@alumnos.unican.es’, ’’,
      ’666111222’, ’’, ’’);
11
12 /*Comprobamos que están insertados*/
13 Select * from alumno
14
15 /*Modificamos algún campo*/
16 UPDATE alumno SET nombre = ’Modificado’ WHERE nombre = ’Nombre1’;
17
18 /*Comprobamos los cambios*/
19 Select * from alumno
```

Tras la ejecución y superación de estas pruebas, se daba por validado el esquema de base de datos, por lo que se procedió a desarrollar las clases de acceso a datos que implementaban las interface DAO. Este proceso se describe en la siguiente sección.

3.4. Desarrollo de las clases de acceso a datos

Una vez finalizado el desarrollo y las posteriores pruebas del esquema de la base de datos, se implementaron las clases de acceso a datos. El objetivo de esta clases es implementar una serie de interfaces bien definidas que permiten aislar al resto de la aplicación de la tecnología concreta que se ha utilizado para gestionar la persistencia de los datos. Dichas interfaces, que fueron definidas en la etapa de diseño arquitectónico (ver Sección 2.5), permiten acceder a los datos almacenados por la aplicación utilizando conceptos relativos únicamente al paradigma orientado a objetos, tales como clases y colecciones. Conceptos específicos de tecnologías concretas, tales como los cursores para bases de datos relacionales [8] o los árboles DOM (*Document Object Model*) [19] para documentos XML [3].

La implementación realizada es específica para MySql. La conexión a la base de datos se realiza a través de JDBC (*Java Database Connectivity)* [23]. JDBC nos proporciona una serie de interfaces y métodos necesarios para la conexión y manipulación de base de datos relacionales alojadas en sistemas gestores de bases de datos pertenecientes a diversos fabricantes, tales como *MySQL* [6] [7], *Oracle* [17] [13] o *Sybase* [28] [11].

Por tanto, lo que cada clase de acceso a datos contiene es el código necesario para implementar cada interfaz DAO a través de una serie de invocaciones a métodos JDBC. El listado 3.3 muestra parte del código de la clase de acceso a datos *AlumnoDAOMySql*, la cual implementa la interfaz *AlumnoDAOInterfaz* (ver Listado 2.2), encargada de la gestión de los datos relativos a la clase o entidad *Alumno*.

Listado 3.3: Clase *AlumnoDAOMySql*

```
1  public class AlumnoDAOMySql implements AlumnoDAOInterfaz{
2
3    private Statement stm;
4
5    private void AlumnoDAOgetConn(){
6      try {
7        stm = ConnectionManagerMySql.conn.createStatement();
8      } catch (SQLException e) {
9        e.printStackTrace();
10     }
11   }
12
13   public int nuevoAlumno(Alumno a){
14     AlumnoDAOgetConn();
15
16     try {
17       return stm.executeUpdate("INSERT INTO intercambio.Alumno (" +
18           "`Centro_idCentro_origen`, " +
19           "`Centro_idCentro_destino`, " +
20           "`programa_intercambio_idprograma_intercambio`, " +
21           "`Nombre`, " +
22           "`Apellido1`, " +
23           "`Apellido2`, " +
24           "`Documento_identificacion`, " +
25           "`Curso`, " +
26           "`email`, " +
27           "`email_aux`, " +
28           "`telefono1`, " +
29           "`telefono2`, " +
30           "`observaciones`) VALUES ("+
31           a.getIdCentroOrigen()+","+
32           a.getIdCentroDestino()+","+
33           a.getIdProgramaIntercambio()+"," +
34           "'"+a.getNombre()+"'," +
35           "'"+a.getApellido1()+"'," +
36           "'"+a.getApellido2()+"'," +
37           "'"+a.getDocumentoIdentificacion()+"'," +
38           "'"+a.getCurso()+"'," +
39           "'"+a.getEmail1()+"'," +
40           "'"+a.getEmail2()+"'," +
41           "'"+a.getTelefono1()+"'," +
42           "'"+a.getTelefono2()+"'," +
```

```
43            "'"+a.getObservaciones()+"')");
44      } catch (MySQLIntegrityConstraintViolationException e){
45        return -1; // retornamos -1, estamos violando la restriccion de
              unicidad del identificador
46      } catch (SQLException e) {
47        e.printStackTrace();
48      }
49      return 0;
50    }
51
52    public int modificaAlumno(String documentoIdAntiguo, Alumno nuevo){
53      AlumnoDAOgetConn();
54
55      try {
56        return stm.executeUpdate("UPDATE intercambio.Alumno SET " +
57            "Centro_idCentro_origen="+nuevo.getIdCentroOrigen()+", " +
58            "Centro_idCentro_destino="+nuevo.getIdCentroDestino()+", " +
59            "programa_intercambio_idprograma_intercambio="+nuevo.
                  getIdProgramaIntercambio()+", " +
60            "Nombre='"+nuevo.getNombre()+"', " +
61            "Apellido1='"+nuevo.getApellido1()+"', " +
62            "Apellido2='"+nuevo.getApellido2()+"', " +
63            "Documento_Identificacion='"+nuevo.getDocumentoIdentificacion
                  ()+"', " +
64            "Curso='"+nuevo.getCurso()+"', " +
65            "email='"+nuevo.getEmail1()+"', " +
66            "email_aux='"+nuevo.getEmail2()+"', " +
67            "telefono1='"+nuevo.getTelefono1()+"', " +
68            "telefono2='"+nuevo.getTelefono2()+"', " +
69            "observaciones='"+nuevo.getObservaciones()+"' " +
70            "WHERE Documento_Identificacion ='"+documentoIdAntiguo+"'");
71      } catch (MySQLIntegrityConstraintViolationException e){
72        return -1; // retornamos -1, estamos violando la restriccion de
              unicidad del identificador
73      } catch (SQLException e) {
74        e.printStackTrace();
75      }
76      return 0;
77    }
78
79    public List<Alumno> getAlumnos(){
80      AlumnoDAOgetConn();
81      List<Alumno> ls = new ArrayList<Alumno>();
82
83      try {
84        ResultSet rs = (ResultSet) stm.executeQuery("SELECT * FROM
              intercambio.Alumno ORDER BY Nombre");
85        while (rs.next()){
86          Alumno a = new Alumno();
87          a.setIdAlumno(rs.getInt("idAlumno"));
88          a.setIdCentroOrigen(rs.getInt("Centro_idCentro_origen"));
89          a.setIdCentroDestino(rs.getInt("Centro_idCentro_destino"));
90          a.setIdProgramaIntercambio(rs.getInt("
                programa_intercambio_idprograma_intercambio"));
91          a.setNombre(rs.getString("Nombre"));
92          a.setApellido1(rs.getString("Apellido1"));
93          a.setApellido2(rs.getString("Apellido2"));
94          a.setDocumentoIdentificacion(rs.getString("
                Documento_identificacion"));
95          a.setCurso(rs.getString("Curso"));
96          a.setEmail1(rs.getString("email"));
97          a.setEmail2(rs.getString("email_aux"));
```

```
98          a.setTelefono1(rs.getString("telefono1"));
99          a.setTelefono2(rs.getString("telefono2"));
100         a.setObservaciones(rs.getString("observaciones"));
101
102         ls.add(a);
103       }
104     } catch (SQLException e) {
105       e.printStackTrace();
106     }
107     return ls;
108   }
109
110   public Alumno getAlumno(String documentoId){
111     AlumnoDAOgetConn();
112     Alumno a = new Alumno();
113
114     try {
115       ResultSet rs = (ResultSet) stm.executeQuery("Select * FROM
              intercambio.Alumno WHERE idAlumno="+documentoId);
116       if (rs.next()){
117         a.setIdCentroOrigen(rs.getInt("Centro_idCentro_origen"));
118         a.setIdCentroDestino(rs.getInt("Centro_idCentro_destino"));
119         a.setIdProgramaIntercambio(rs.getInt("
                programa_intercambio_idprograma_intercambio"));
120         a.setNombre(rs.getString("Nombre"));
121         a.setApellido1(rs.getString("Apellido1"));
122         a.setApellido2(rs.getString("Apellido2"));
123         a.setDocumentoIdentificacion(rs.getString("
                Documento_identificacion"));
124         a.setCurso(rs.getString("Curso"));
125         a.setEmail1(rs.getString("email"));
126         a.setEmail2(rs.getString("email_aux"));
127         a.setTelefono1(rs.getString("telefono1"));
128         a.setTelefono2(rs.getString("telefono2"));
129         a.setObservaciones(rs.getString("observaciones"));
130       }
131     } catch (SQLException e) {
132       e.printStackTrace();
133     }
134     return a;
135   }
136 }
```

Dentro de dicha clase, por ejemplo, el método *nuevoAlumno* tiene como cometido dar de alta un nuevo alumno en la aplicación, a partir de un objeto de la clase *Alumno*, que recibe como parámetro de entrada. Por tanto, la responsabilidad principal del método *nuevoAlumno* es insertar una nueva fila con los datos del alumno que se recibe como parámetro en la correspondiente tabla de nuestra base de datos relacional. Para ello, lo primero que debemos hacer es obtener una conexión con nuestro esquema concreto de base de datos (Listado 3.3, Línea 14).

A continuación se realiza la inserción con los datos del alumno recibido como parámetro. Para ello se construye y ejecuta, utilizando las facilidades proporcionadas por JDBC, una sentencia SQL tipo *INSERT*, utilizando como datos de entrado los atributos del objeto alumno recibidos como parámetro (Listado 3.3, Líneas 17-43). Si se viola alguna restricción de las declara-

das para la tabla *Alumno*, como la de unicidad para las claves primarias, se lanza una excepción del tipo *MySQLIntegrityConstraintViolationException*. En ese caso la inserción no podrá realizarse y se retornará un valor negativo para indicar del error. El error se procesaría en la capa de negocio, y se transformaría en una serie de indicaciones en la capa de presentación que proporcionen al usuario información útil acerca de como solventarlo.

El método *getAlumnos* (Listado 3.3, Líneas 79-108) es un método típico de consulta, el cual devuelve una lista, como colección de objetos, con todos los objetos de tipo *Alumno* almacenados en nuestra base de datos. Dicho método no recibe ningún parámetro de entrada. Este método obtiene, en primer lugar, una conexión con la base de datos (Listado 3.3, Línea 80). A continuación, ejecuta una consulta SQL tipo *SELECT* sobre la tabla *Alumno* para obtener todas las entradas de dicha tabla (Línea 84). Dicha consulta devuelve un *ResultSet*, que es cómo se denomina un cursor en terminología JDBC. Un cursor no es más que una matriz bidimensional donde el elemento *(i,j)* contiene el valor de atributo j de la fila i para el resultado de la consulta realizada. Dicho *ResultSet* es un concepto propio de las bases de datos relacionales, por lo que lo debemos transformar a una simple colección de objetos antes de devolver dichos datos a la capa de negocio. Para ello recorremos dicho cursor o matriz fila por fila, creando un objeto de tipo *Alumno* por cada fila y añadiéndolo a la colección que hemos de devolver (Listado 3.3, Líneas 85-103). Obviamente, se pueden producir diversas situaciones excepcionales durante este proceso, las cuales son convenientemente gestionadas (Listado 3.3, Líneas 104-106).

Este esquema, el cual separa la interfaz de acceso a los datos de sus implementaciones concretas, permite que los cambios que debamos realizar en un futuro con relación a la tecnología de persistencia que estamos usando sólo afecten a las clases de acceso a datos mostradas en esta sección. El resto de la aplicación permanecería sin alteración alguna.

3.5. Despliegue de la base de datos

Una vez que dispusimos de todo el material necesario para la capa de persistencia, podíamos proceder a su despliegue, de forma que la capa de persistencia quedase lista para ser utilizada como una caja negra durante el desarrollo de las diferentes iteraciones de desarrollo de la capa de presentación y negocio, de acuerdo a la metodología expuesta en la Figura 1.1. Esta sección describe dicha fase de despliegue.

La primera parte del despliegue consistió en seleccionar un servidor donde desplegar el sistema gestor de base de datos *MySQL*. Debido a la inexistencia de un equipo adecuado en las instalaciones del cliente que pudiese ser utilizado como servidor, se decidió adquirir un equipo nuevo para tal fin. Aparte de la necesidad generada por esta aplicación, se decidió que sería

buena idea disponer de un servidor interno para futuras aplicaciones. Como este servidor iba a ser susceptible de contener datos sensibles protegidos por la LOPD (*Ley Orgánica de Protección de Datos* [12]) se le buscó una ubicación adecuada con acceso restringido. Una vez adquirido el servidor, instalamos en él la suite *AppServ*. Esta suite proporciona tanto el sistema gestor de base de datos *MySql* como una serie de herramientas para la gestión de dicho sistema, como un servidor de datos al que se pueda acceder de forma remota. A continuación, se creó el esquema de la base de datos utilizando para ello el script generado en la Sección 3.2.

Con la base de datos instalada en el servidor, se procedió a su configuración. En primer lugar se dio de alta un usuario con los permisos necesarios de acceso, consulta y modificación de dicho esquema de base de datos. Este usuario sería el que utilizarían las clases de acceso a datos para conectarse al servidor. Por tanto, configuramos también dichas clases con los datos correctos para acceder de forma remota al servidor. Además, para poder permitir dichas conexiones remotas, fue necesario reconfigurar de forma apropiada el *firewall* de *Windows*. *MySQL* utiliza por defecto el puerto 3306 para sus conexiones remotas. Este puerto está bloqueado por defecto por el firewall de *Windows* y por lo tanto fue necesaria crear una regla de entrada para permitir las conexiones TCP a través de este puerto.

Por último se instaló y configuró un sistema de copias de respaldo para poder realizar copias de seguridad de nuestra base de datos. Para realizar estas copias de seguridad se utilizó *PhpMyBackupPro*. Esta herramienta nos permite crear y restaurar copias de seguridad de una manera sencilla mediante su acceso web. Para definir la política de copias de seguridad se tuvieron en cuenta los periodos en los que la base de datos estaría más activa. Al tratase de una aplicación para el registro de los alumnos de intercambio es evidente que ese periodo sería durante las fechas de matriculaciones, es decir, al principio de cada cuatrimestre. Teniendo esto en cuenta se programaron copias de seguridad diarias automáticas durante esas semanas. Como era posible que durante la semana posterior a las matriculaciones hubiera algún cambio excepcional, se decidió prolongar ese periodo durante una semana más. Por último se permitió la realización manual de copias de seguridad por parte de cualquier administrador de la base de datos por si fuese necesario realizar alguna en un momento particular determinado.

Con esto finalizaba el proceso de creación de la capa de persistencia, la cual quedaba lista para ser utilizada como un componente opaco durante el desarrollo de las capas de presentación y negocio.

3.6. Sumario

Este capítulo ha descrito cómo se realizaron las distintas fases del desarrollo de la capa de persistencia del sistema. Ha explicado cómo se llevó a

cabo su diseño y su implementación. También se han descrito las pruebas realizadas para comprobar el correcto funcionamiento del esquema de base de datos, con objeto de comprobar que este era capaz de mantener la integridad de los datos del sistema. A continuación, se ha descrito el proceso de desarrollo de las clases de acceso a datos. Por último, se ha descrito el proceso de despliegue de la capa de persistencia.

Capítulo 4

Desarrollo de las Capas de Presentación y Negocio

El presente capítulo describe el proceso de desarrollo de las capas de presentación y negocio, las cuales fueron realizadas mediante diversas iteraciones en las que se añadía nueva funcionalidad al sistema tal y como se vio en la metodología expuesta al principio de este documento. A continuación se detalla el proceso de despliegue de ambas capas que, junto a la capa de persistencia desplegada en la sección anterior, conforman la aplicación completa. Por último se explica el desarrollo de una iteración adicional en la que, a modo de ejemplo, se demostrará como es posible realizar una interfaz web sin que esto afecte al resto del sistema.

Índice

4.1. Introducción

Una vez que la capa de persistencia estuvo finalizada y lista para su uso se continuó con el desarrollo de las capas de presentación y negocio. Dado la simplicidad de las operaciones de la capa de negocio, se decidió prescindir de la etapa de diseño detallado, al considerarse que ésta no podía aportar nada significativo a la calidad del producto final, y su realización

alargaría los tiempos de desarrollo. Por tanto, la primera fase de cada iteración consistió en la implementación del código necesario para soportar la nueva funcionalidad a incluir. Para ello, en primer lugar, se desarrollaron las partes de la interfaz de usuario necesarias para dar soporte a los casos de uso incluidos en cada iteración. Una vez testada dicha interfaz gráfica, se implementaron los servicios de la capa de negocio que eran necesarios para dar soporte a las operaciones de la interfaz gráfica. La capa de negocio a su vez hacía uso de la capa de persistencia. Una vez implementada la capa de negocio, se probaba el correcto funcionamiento de las tres capas. Una vez comprobado el correcto funcionamiento de la nueva lógica añadida, se mostraba al cliente para que éste validase el trabajo realizado y diese su aprobación.

Por motivos de espacio, y en aras de hacer este documento no excesivamente pesado al lector, no describiremos todas las iteraciones realizadas, centrándonos exclusivamente en dos de ellas, más concretamente, en la iteración dedicada a implementar la lógica relacionada con la gestión de las universidades y la dedicada a implementar la lógica relacionada con la gestión de las asignaturas. Ambas iteraciones se describen en las dos siguientes secciones.

Una vez que todas las iteraciones estuvieron finalizadas se procedió al despliegue completo de la aplicación, que será explicado detalladamente en la Sección 4.4. La Sección 4.5 muestra un ejemplo de como podría realizarse un cambio de interfaz para la aplicación sin necesidad de modificar las capas de negocio o persistencia. Finalmente la última sección de este capítulo sirve de sumario y cierre al mismo.

4.2. Iteración 1: Gestión de Universidades

En esta primera iteración se desarrollaron los requisitos relacionados con la *Gestión de Universidades*. Dicha gestión de universidades, más concretamente, está asociada con los requisitos *R01* y *R09* (Ver Figura 2.1). Estos requisitos incluyen tanto dar de alta nuevas universidades como la modificación de las existentes o la consulta de las mismas, además de especificar los distintos datos que debemos guardar de ellas. Además, al ser la primera iteración, debíamos también diseñar el esqueleto de la interfaz de la aplicación de forma que pudiese añadirse a ésta las funcionalidades requeridas por las restantes iteraciones.

Por tanto, en primer lugar se diseño un armazón para la interfaz gráfica de la aplicación que permitiese acomodar los formularios, tanto de entrada como de salida, que se iban a ir incorporando a la aplicación tras sucesivas iteraciones. Tras el diseño de dicho armazón, se diseñaron los formularios necesarios para soportar la gestión de universidades. Una vez comprobado el correcto funcionamiento de la interfaz, se procedió a la implementación

de la capa de negocio y a su posterior fase de pruebas. Concluidas las tres capas de la aplicación para la gestión de las universidades, se realizó la integración de las mismas, se realizó un nuevo proceso de pruebas destinado a comprobar el correcto funcionamiento de las tres capas y se mostró el trabajo realizado a los usuarios finales para su validación y aceptación. Las siguientes subsecciones describen en mayor detalle el proceso de desarrollo de esta iteración.

4.2.1. Desarrollo de la Capa de Presentación

El primer paso para desarrollar esta iteración era, tal como se ha comentado con anterioridad, desarrollar el esqueleto de la interfaz gráfica para nuestra aplicación. En nuestro caso, la interfaz gráfica debía ser una interfaz de escritorio, que íbamos a desarrollar utilizando las librerías que proporciona el lenguaje *Java* para tal propósito. Como dicha interfaz debía contener diversos formularios, donde cada conjunto de ellos serviría para un propósito determinado, como la gestión de las universidades, se optó por organizar la interfaz siguiendo un modelo de *carpetas y pestañas*. Cada pestaña contendría el conjunto de formularios destinado a satisfacer un objetivo concreto, es decir, a satisfacer un mismo requisito de alto nivel o caso de uso blanco.

La Figura 4.1 muestra la estructura general de la interfaz creada y la estructura de carpetas y pestañas, donde se distinguen las siguientes pestañas:

Alumnos Se encarga de gestionar toda la información referente a los alumnos y sus planes de estudio.

Profesores Se encarga de gestionar toda la información referente a los intercambios realizados por profesores.

Universidades Se encarga de gestionar la información relativa a las distintas universidades participantes en programas de intercambio, así como los centros de los que disponen y las asignaturas que oferta cada centro.

Probar el correcto funcionamiento de este esqueleto de la interfaz gráfica no tenía mayor misterio. Bastaba con comprobar que desde cada pestaña podíamos movernos a cualquiera de las otras existentes. Una vez creado dicho esqueleto, el siguiente paso era crear los formularios que diesen soporte a los casos de uso de bajo nivel relacionados con la gestión de universidades. Más concretamente, debía soportarse tanto el alta de nuevas universidades como la modificación de los datos de las ya existentes.

Para dar soporte a dichas operaciones, se creó el pequeño formulario que se muestra en la Figura 4.1. Si queremos dar de alta una nueva universidad, deberemos pulsar el botón *Nuevo*, tras lo cual se nos mostrará el diálogo

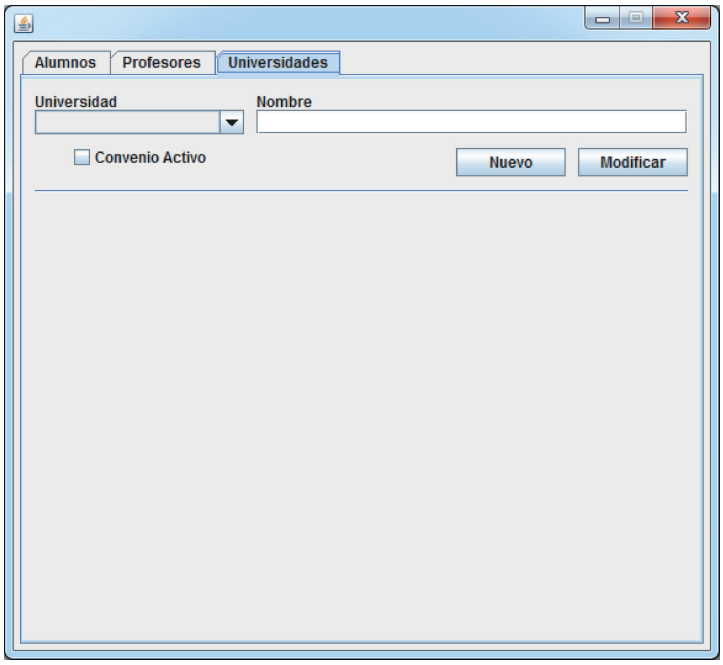

Figura 4.1: Interfaz para la *Gestión de Universidades*

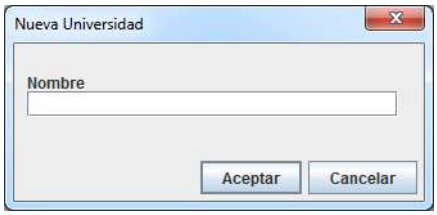

Figura 4.2: Diálogo para dar de alta una universidad

de la Figura 4.2. A través de este diálogo introducimos el nombre de la nueva universidad y pulsamos *Aceptar*, tras lo cual se cursaría la petición de alta a la aplicación. Para cursar dicha petición, la interfaz invoca el método *nuevaUniversidad*, proporcionado por la capa de negocio a través de la interfaz *GestionUniversidades* (ver Figura 2.8). Como es de esperar, se puede volver al formulario anterior sin cursar petición alguna pulsando el botón *Cancelar*. Por defecto, al añadir una nueva universidad se da de alta con el campo convenio activo a verdadero, de acuerdo con los deseos del cliente.

Para modificar los datos de una universidad, debemos seleccionarla en la lista desplegable que aparece en la parte superior izquierda, bajo la etiqueta *Universidad*. Tras su selección, sus datos aparecerán en la interfaz. Dichos

datos son el nombre de la universidad, que aparecerá en la caja de texto bajo la etiqueta *Nombre*, y si el convenio con ella sigue activo o no, lo cual se indica a través del *checkbox* con la etiqueta *Convenio Activo*. Dichos datos pueden ser modificados desde la propia interfaz. Una vez modificados, pulsaremos el botón *Modificar* para comunicar los cambios a la aplicación.

En ambos casos, altas y modificaciones, la interfaz gráfica comprueba que el nombre de la universidad no sea la cadena vacía antes de enviar los datos a la aplicación para su procesamiento. En el caso de que lo fuese, el sistema no enviaría los datos a la aplicación y notificaría al usuario el problema para que lo solucione. Además de lo anterior, también se muestra una notificación de error si se pulsa el botón *Modificar* sin tener seleccionada una universidad.

Una vez desarrollada la interfaz, se realizaron una serie de pruebas sistemáticas destinadas a comprobar el correcto funcionamiento de la interfaz. Más concretamente, se verificó que:

1. Se podía navegar de forma correcta por la interfaz, moviéndonos sin problemas entre los distintos formularios.

2. Cada elemento gráfico funcionase de manera adecuada, por ejemplo, que las listas se desplegaban sin problemas y se podía navegar por sus opciones de manera amigable al usuario.

3. Los datos enviados a la aplicación eran correctos.

4. Las comprobaciones que debía realizar la interfaz estaban correctamente implementadas, y no era posible enviar, ni para alta ni modificación, datos relativos a una universidad donde su nombre fuese la cadena vacía.

Tras este proceso, la interfaz gráfica para esta primera iteración se podía dar por concluida, por lo que sólo restaba implementar la lógica necesaria para el procesamiento de las peticiones realizadas desde la capa de negocio, lo cual se describe en la siguiente subsección.

4.2.2. Desarrollo de la Capa de Negocio

Tal como se ha comentado en la introducción a este capítulo, la lógica de la capa de negocio es relativamente sencilla, ya que los procesos de intercambios no poseen reglas de negocio demasiado complejas. Por tanto, básicamente el cometido de la capa de negocio es hacer de pasarela o puente entre la capa de presentación y la de persistencia.

En esta iteración concreta, la capa de negocio debía implementar las operaciones de la interfaz *GestionUniversidades*, definida durante el proceso de diseño arquitectónico (ver Figura 2.8). Para implementar dichas operaciones, la capa de negocio simplemente recoge los datos enviados por la capa

Listado 4.1: Clase *UniversidadNegocio*

```
1  public int nuevaUniversidad(Universidad u) {
2    return universidadDaoInterfaz.nuevaUniversidad(u);
3  }
4
5  public int modificarUniversidad(int idUniversidadAntigua, Universidad
       nueva) {
6    return universidadDaoInterfaz.modificarUniversidad(
         idUniversidadAntigua, nueva);
7  }
```

de presentación, los convierte en objetos si ello fuese necesario e invoca el método o los métodos que corresponda de la capa de negocio.

El Listado 4.1 muestra la implementación de los métodos *nuevaUniversidad* y *modificarUniversidad* de la interfaz *GestionUniversidades*.

Una vez concluida la implementación de la capa de negocio, se procedió a comprobar su correcto funcionamiento. Para ello se realizó un código de prueba que comprobaba el funcionamiento del código desarrollado para un conjunto de casos de prueba sistemáticamente diseñado para cubrir el abanico de posibles entradas al sistema. Tras certificar el correcto funcionamiento de la capa de negocio como componente, se procedió a su integración con las capas de presentación y persistencia.

4.2.3. Pruebas de Integración

Una vez desarrolladas las tres capas, se procedió a verificar el funcionamiento de las mismas una vez integradas. Para ello se ejecutaron casos de prueba destinados a verificar que:

- Al introducir los datos de una nueva universidad en el diálogo de la Figura 4.2 se creaba una nueva fila en la tabla *Universidad*, con los mismos datos que habían sido introducidos en el formulario. Además, la nueva universidad debía aparecer en la lista desplegable bajo la etiqueta *Universidad* (ver Figura 4.1), pudiendo además ser seleccionada sin problema alguno.

- Al seleccionar una universidad de la lista desplegable, los datos de la misma se incorporan al formulario. Además, se comprobó que tras esta primera selección, al seleccionar otras universidades, se realizaban los cambios pertinentes en la interfaz gráfica.

- Tras seleccionar una universidad, al cambiar de pestaña y volver a la pestaña original, los valores de la universidad seleccionada permanecían en la misma.

- Tras seleccionar una universidad y realizar cambios en los valores mostrados, al pulsar el botón *Modificar*, los cambios realizados se hacían efectivos en la base de datos y se reflejaban convenientemente en la interfaz gráfica, la cual debía refrescarse y mostrar los cambios realizados al instante.

- Al pulsar el botón *Modificar* sin tener una universidad seleccionada se notificaba del problema al usuario.

- Al pulsar el botón *Cancelar* o *Cerrar* en el diálogo de la Figura 4.2 no se efectuaba ningún cambio en la base de datos del sistema.

- Al pulsar el botón *Aceptar* del diálogo de la Figura 4.2 con el campo de texto vacío se notificaba del problema al usuario.

Tras la realización de estas pruebas de integración, se daba por concluida la primera iteración del desarrollo de las capas de presentación y negocio.

4.3. Iteración 3: Gestión de Asignaturas

Debido a que el desarrollo de la segunda iteración fue muy similar al de la primera, tanto en forma como en contenido, se ha optado por explicar la tercera iteración, la cual entendemos que ayuda mejor a comprender el proceso de desarrollo realizado.

Las funcionalidades que se añadieron en esta iteración fueron las correspondientes a la *Gestión de Asignaturas*. Los requisitos asociados a la gestión de asignaturas son *R03* y *R11* (Ver Figura 2.1). En estos requisitos están comprendidas las operaciones de dar de alta nuevas asignaturas y modificar las existentes, además de los datos que debemos guardar de ellas.

El primer paso fue diseñar los formularios necesarios para satisfacer las funcionalidades de la gestión de asignaturas. A continuación se procedió a probar su correcto funcionamiento. Después se implementó la capa de negocio, seguida de su correspondiente proceso de pruebas. En el siguiente paso se integraron todas las capas y se realizó una última fase de pruebas que permitió corroborar que el sistema funcionaba adecuadamente. Por último se mostró al cliente la nueva funcionalidad para que éste pudiera validarla. A continuación se describe más detalladamente todo el proceso que se llevó a cabo en esta iteración.

4.3.1. Desarrollo de la capa de presentación

El primer paso en esta etapa fue crear los formularios que soportarían las funciones relacionadas con la gestión de asignaturas. Estos formularios debían permitir el alta de nuevas asignaturas así como su modificación. Estas

operaciones se implementan mediante el formulario que puede verse en la parte inferior de la Figura 4.3.

Para dar de alta una nueva asignatura se pulsará el botón *Nuevo* que desplegará el diálogo de la Figura 4.4. En este diálogo podremos introducir los datos requeridos por las asignaturas, es decir, la universidad a la que pertenecen, el centro en el que se imparten, su nombre, su semestre y curso correspondientes, así como el número de crédito ECTS que comprenden. Cabe destacar que el desplegable bajo la etiqueta *Centro* se encuentra inactivo hasta que se selecciona una universidad. A partir de ese momento, el deplegable nos mostrará los centros correspondientes a la universidad seleccionada y nos permitirá seleccionar el que corresponda. Una vez introducidos estos datos pulsaremos el botón *Aceptar*, que invocará al método *nuevaAsignatura* que la capa de negocio nos proporciona mediante la interfaz *GestionAsignaturas* (ver Figura 2.8). En caso de que no queramos añadir la asignatura volveremos al formulario anterior pulsando el botón *Cancelar*.

Para modificar una asignatura en primer lugar debemos seleccionarla en el desplegable etiquetado como *Asignatura*. Éste es el único elemento del formulario con el que podemos interactuar, ya que el resto permanecen inactivos hasta que se haya seleccionado una asignatura. Una vez seleccionada, sus datos aparecerán en la interfaz. A partir de ese momento, el resto de los elementos de la interfaz quedarán habilitados. Podremos, entonces, modificar los que se correspondan con los cambios que queramos realizar en la asignatura seleccionada. El funcionamiento de los desplegables marcados como *Centro* y *Universidad* presentan la misma particularidad que se vió cuando se explicó el proceso de alta de una asignatura, es decir, no podremos seleccionar un centro que no pertenezca a la universidad que esté seleccionada en ese momento. Los cambios se harán efectivos tras pulsar el botón *Modificar* que hará una llamada al método *ModificarAsignatura* proporcionado por la interfaz *GestionAsignaturas* (ver Figura 2.8).

Tanto para el alta como para la modificación de una asignatura la interfaz comprueba que todos los datos hayan sido introducidos correctamente. Comprueba que los campos de texto no son la cadena vacía, que los desplegables tienen seleccionada una opción válida y que los campos numéricos como el curso o los créditos son mayores que 0. Si alguno de estos datos no son correctos, el sistema no modifica los datos y envía una notificación al usuario para que lo solucione. La interfaz también comprueba y notifica si se pulsa el botón *Modificar* sin tener una asignatura seleccionada.

Cuando la interfaz estuvo completamente desarrollada se realizaron varias pruebas para comprobar que funcionaba de acuerdo a lo esperado. En las pruebas se verificó, al igual que en las iteraciones anteriores, que:

1. Se podía navegar de forma correcta por la interfaz, moviéndonos sin problemas entre los distintos formularios.

2. Cada elemento gráfico funcionase de manera adecuada, por ejemplo,

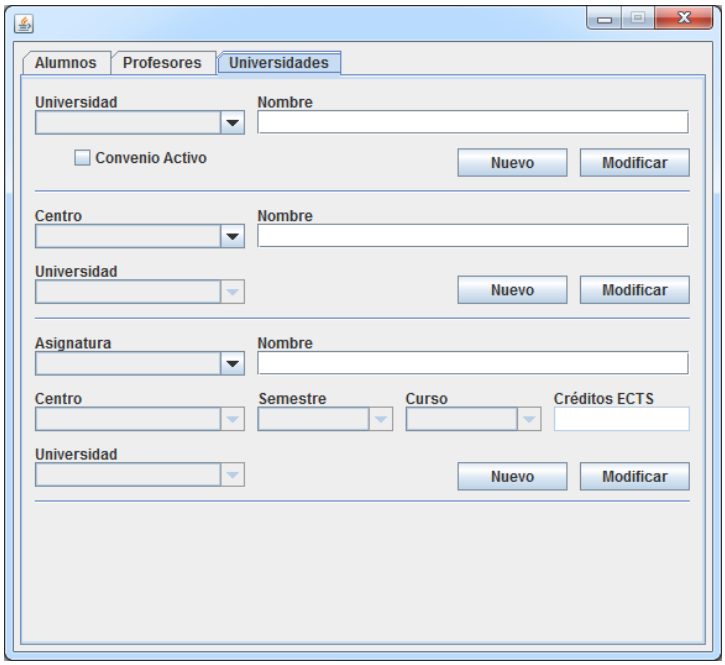

Figura 4.3: Interfaz para la *Gestión de Asignaturas*

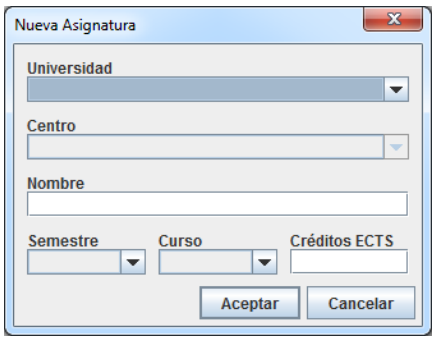

Figura 4.4: Diálogo para dar de alta una asignatura

que las listas se desplegaban sin problemas y se podía navegar por sus opciones de manera amigable al usuario.

3. Los datos enviados a la aplicación eran correctos.

4. Las comprobaciones que debía realizar la interfaz estaban correctamente implementadas, y no era posible enviar, ni para alta ni modificación, datos con campos vacíos o incorrectos.

Listado 4.2: Clase *AsignaturaNegocio*

```
1  public int nuevaAsignatura(Asignatura a) {
2    return asignaturaDaoInterfaz.nuevaAsignatura(a);
3  }
4
5  public int modificarAsignatura(int idAsignaturaAntigua, Asignatura
       nueva) {
6    return asignaturaDaoInterfaz.modificarAsignatura(
         idAsignaturaAntigua, nueva);
7  }
```

4.3.2. Desarrollo de la capa de Negocio

Como se ha dicho anteriormente, la capa de negocio no posee reglas de negocio complejas por lo que su implementación es bastante sencilla, siendo su principal cometido hacer de pasarela entre las capas de presentación y persistencia.

En esta iteración se implementaron las operaciones correspondientes a la interfaz *GestiónAsignaturas* que se definió anteriormente (ver Figura 2.8). Para llevar a cabo esta implementación la capa de negocio recibe los datos que la capa de presentación le envía y los transmite a la capa de persistencia mediante la invocación de los métodos pertinentes. En el Listado 4.2 puede verse la implementación de los método *nuevaAsignatura* y *modificarAsignatura* de la interfaz *GestionAsignaturas*

Con la capa de negocio debidamente implementada se pasó a comprobar su funcionamiento. Para realizar estas pruebas se diseñó un código encargado de comprobar un conjunto de casos de pruebas que cubriera todas las posibles entradas al sistema. Una vez comprobado que el funcionamiento de la capa de negocio era el esperado, se procedió a integrarlo con las capas de presentación y persistencia.

4.3.3. Pruebas de Integración

Tras el desarrollo de las tres capas y de su integración se comprobó que su funcionamiento era correcto mediante una serie de pruebas que verificaron que:

- Al introducir los datos de una nueva asignatura en el diálogo de la Figura 4.4 se creaba una nueva fila en la tabla *Asignatura*, con los mismos datos que habían sido introducidos en el formulario. Además, la nueva asignatura debía aparecer en la lista desplegable bajo la etiqueta *Asignatura* (ver Figura 4.3), pudiendo además ser seleccionada sin problema alguno.

- Al seleccionar una asignatura de la lista desplegable, los datos de la

misma se incorporan al formulario y los campos inactivos pasaban a ser utilizables. Además, se comprobó que tras esta primera selección, al seleccionar otras asignaturas, se realizaban los cambios pertinentes en la interfaz gráfica.

- Tras seleccionar una asignatura, al cambiar de pestaña y volver a la pestaña original, los valores de la asignatura seleccionada permanecían en la misma.
- Tras seleccionar una asignatura y realizar cambios en los valores mostrados, al pulsar el botón *Modificar*, los cambios realizados se hacían efectivos en la base de datos y se reflejaban convenientemente en la interfaz gráfica, la cual debía refrescarse y mostrar los cambios realizados al instante.
- Al pulsar el botón *Modificar* sin tener una asignatura seleccionada se notificaba del problema al usuario.
- Al pulsar el botón *Cancelar* o *Cerrar* en el diálogo de la Figura 4.4 no se efectuaba ningún cambio en la base de datos del sistema.
- Al pulsar el botón *Aceptar* del diálogo de la Figura 4.4 con alguno de los campos vacío o incorrectamente completado se notificaba del problema al usuario.

Una vez completadas estas pruebas se daba por finalizada la tercera iteración del desarrollo de las capas de presentación y negocio.

4.4. Despliegue

Una vez finalizadas todas las iteraciones la aplicación estaba lista para pasar a la fase de despliegue, tras la cual podría ser utilizada por el cliente.

En primer lugar la aplicación debía ser empaquetada de manera adecuada para su distribución. Se decidió que la mejor manera para realizarlo sería mediante un archivo *jar* ejecutable. Mediante la opción *exportar* que *Eclipse* nos ofrece podemos empaquetar un proyecto *Java* con todas las librerías y recursos necesarios para su funcionamiento en un archivo *jar* que podrá ser ejecutado en cualquier equipo que tenga instalada una *Máquina Virtual Java* [18].

Los requisitos que la máquina en la que quiera ejecutarse la aplicación debe cumplir son los siguientes:

- Sistema compatible con Java
- Java 1.6.0 o superior
- 128MB de Memoria

Además de entregar el ejecutable al cliente se dejó alojado en el servidor para que pudiera volver a ser descargado en cualquier momento si fuera necesario.

La última fase del despliegue de la aplicación consistió en enseñar su funcionamiento al cliente para que este no tuviera problemas al utilizarla. Esto se llevó a cabo mediante sesiones de entrenamiento en las que se le explicó con detalle como debía interactuar con la aplicación.

4.5. Anticipando el Cambio: Modificación de la Interfaz de Usuario

Como se ha comentado anteriormente, la principal ventaja de las arquitecturas de tres capas [10] es que permiten una evolución independiente de cada una de sus capas. El objetivo de esta sección es demostrar que el requisito de modularidad exigido para el desarrollo de la aplicación, comentado en la Sección 2.4, se satisface. Para ello realizaremos un cambio que escenifique y pruebe la modularidad de la aplicación desarrollada.

Uno de los cambios fácilmente predecibles para nuestro sistema sería el reemplazamiento de la interfaz de escritorio por una interfaz web que eliminase problemas de instalación y mantenimiento en presencia de usuarios concurrentes, así como su integración en las herramientas que la Universidad de Cantabria proporciona a través de su campus virtual. Gracias a la arquitectura de tres capas empleada en el desarrollo de nuestra sistema (ver Sección 2.5), es posible realizar este cambio sin que ello afecte a las otras capas. Es decir, realizar este cambio sólo implicaría el desarrollo de una nueva capa de presentación, permaneciendo las capas de negocio y persistencia sin modificación alguna.

Para probar estas afirmaciones, crearemos una serie de formularios web que sirvan de reemplazo a los formularios de escritorio actualmente existentes y que a su vez se integren con las capa de negocio y persistencia creadas.

Para ello, en primer lugar explicaremos en funcionamiento general de las interfaces web y de las *JSP (JavaServer Pages)* [1], que es la tecnología que el entorno *Java* proporciona para la creación de interfaces web dinámicas.

4.5.1. Procesamiento de Peticiones Web

Al sustituir la interfaz original por una interfaz web, ésta será la encargada de realizar las llamadas a los métodos que la interfaz de negocio nos proporciona. Por tanto, el principal problema al que nos tenemos que enfrentar, aparte del de tener que rediseñar el aspecto gráfico de nuestra interfaz, es cómo procesar dichas peticiones. Es decir, cómo convertir una petición recibida vía web en una llamada a un método *Java*.

Las peticiones se realizan a través de un navegador web, desde una página

web codificada en lenguaje *HTML*. Desde dicha página web se puede cursar una petición HTTP normal, tal cómo si estuviésemos solicitando una página web normal. Los métodos del protocolo HTTP, *POST* y *GET* permiten añadir a la petición de una página web una serie de parámetros.

En nuestro caso, las páginas web que hacen de interfaz de usuario no solicitan páginas web normales, sino que invocan un tipo de página web especial, que en nuestro caso serán páginas *JSP*. Una página *JSP* es como una página *HTML*, pero que posee porciones de código *Java* incrustado. Dicho código *Java* puede utilizarse para recuperar los parámetros pasados a través de los métodos POST o GET y componer una llamada a la capa de negocio. Además, las páginas *JSP* poseen la capacidad de modificar su contenido en cada petición. De esta forma, la estrategia para procesar las peticiones web es, por cada operación de cada interfaz de la capa de negocio, crear una página web que se encargue de convertir dicha petición en una invocación a tal operación de la capa de negocio; y, en función del valor que retorne, componer una página web *HTML* de respuesta.

De esta forma, por ejemplo, para el caso del alta de una universidad, podríamos tener un formulario web, dentro una página *HTML* común, con los mismos campos que el formulario existente en la versión de escritorio, el cual se mostraba en la Figura 4.2. Este formulario, al pulsar el botón aceptar, solicitaría una página *JSP* denominada *nuevaUniversidad.jsp*, añadiendo los parámetros de dicha petición, como parte de una solicitud POST o GET. El servidor web ejecutaría la correspondiente página *JSP*, que recuperaría los valores pasados como parámetros mediante los métodos POST o GET. Utilizando dichos parámetros, compondría una llamada al método *nuevaUniversidad* de la interfaz *GestionUniversidades* (ver Figura 2.8). A continuación, la página *JSP* compondría una página *HTML* normal como respuesta a la petición web realizada. En función del valor retornado por la operación *nuevaUniversidad*, la página web compuesta mostrará un mensaje indicando que la operación se ha realizado con éxito o no, informando en este último caso del error producido.

Merece la pena señalar que para poder procesar páginas *JSP* nos hace falta disponer de un servidor web que posea tal capacidad, que no es el caso de los servidores web normales, que sólo procesan páginas *HTML*. Un ejemplo conocido de tal servidor es *Apache Tomcat* [2]. La siguiente sección describe en mayor detalle los cambios concretos que hemos realizado para comprobar que la modularidad de nuestra aplicación era la deseada.

4.5.2. Desarrollo de la Interfaz Web

Para probar y analizar la modularidad de nuestra aplicación, y dado que el objetivo del desarrollo de esta interfaz es puramente demostrativo y no se pretendía obtener una interfaz completamente nueva y funcional, refactorizamos exclusivamente el diseño de los formularios que nos permitían

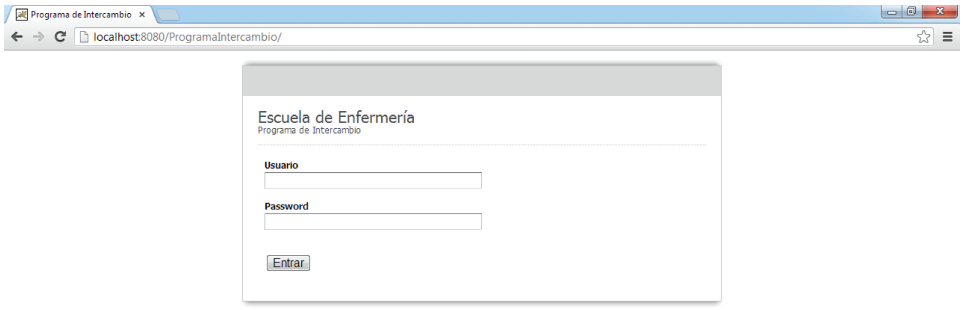

Figura 4.5: Formulario de autenticación

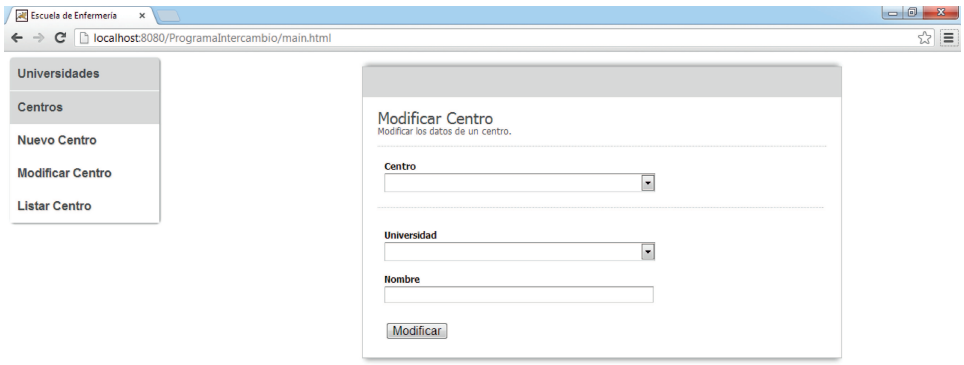

Figura 4.6: Formulario *Nuevo Centro*

manejar las operaciones correspondientes a la gestión universidades y centros. Por lo tanto las operaciones que se podrán realizar desde esta interfaz serán *añadir*, *listar* y *modificar* tanto las universidades como los centros. Mostramos a continuación, los formularios web creados, explicando su funcionamiento.

En primer lugar, para acceder a la aplicación es necesario autenticarse a través del formulario de la Figura 4.5. Este formulario invoca una página *JSP* que se encarga de obtener una conexión válida con la base de datos, invocando el método *conectar* de la interfaz *GestionSeguridad*, pasándole como parámetros el nombre y la contraseña que el usuario haya introducido. Si la conexión se realiza con éxito se podrá acceder a la aplicación, en caso contrario se notificará del error.

Tras autenticarse, el sistema nos redirige a la interfaz de gestión que puede verse en la Figura 4.6. Esta interfaz consta de dos partes, un menú desplegable situado en el lateral izquierdo y una zona central en la que irán apareciendo los formularios correspondientes a medida que se vayan seleccionando en el menú.

A modo de ejemplo, mostramos el funcionamiento del código para dar de alta un nuevo centro. En primer lugar obtenemos el formulario para dar de alta un nuevo centro, el cual se muestra en la Figura 4.6. Dicho formulario no puede ser codificado como una página *HTML* normal, ya que los valores de la lista desplegable etiquetada como *Universidad* dependen de los valores almacenados en la base de datos. Por tanto, dicha lista desplegable se compone de forma dinámica por cada petición.

Para generar dinámicamente la lista de universidades debemos realizar una llamada al método *getUniversidades* que nos proporciona la capa de negocio a través de la interfaz *GestionUniversidades*. Para ello necesitamos incrustar el código *Java* necesario en nuestro *HTML*. Cuando se incrusta código *Java* en una página *HTML* este siempre debe estar precedido por el símbolo $<$ % y finalizar con % $>$ de manera que el servidor pueda identificarlo cuando procese la página. En el Listado 4.3 puede verse parte del código que genera el formulario para crear un nuevo centro y cuyo funcionamiento veremos a continuación.

En este código se van declarando mediante *HTML* los elementos que nuestro formulario vaya a emplear. De esta manera creamos el formulario y además de indicarle que pase los parámetros mediante *POST* le indicamos cual será la página que los procesará y que debe se llamada al pulsar el botón aceptar (Listado 4.3, Línea 2). Para crear la lista dinámica debemos declararla de la manera habitual (Listado 4.3, Línea 12), sin embargo a la hora de rellenar las opciones debemos incluir el código *Java* para que la rellene dinámicamente (Listado 4.3, Líneas 13-19). Lo que hacemos en estas líneas es crear una interfaz *GestionUniversidades* que nos permita invocar el método *getUniversidades* el cual nos devolverá la lista de universidades. Seguidamente recorreremos esta lista mediante un bucle *for* e iremos imprimiendo los nombres de las distintas universidades para hacerlas visibles cuando se despliegue la lista en el formulario.

Tras seleccionar la opción nuevo centro e introducir los datos en el formulario de la Figura 4.6 se pulsa *Aceptar* y el formulario se encarga de llamar a la página *JSP* cuyo código puede verse en el Listado 4.4

Este código lo primero que hace es crear un objeto de la clase *Centro* (Listado 4.4, Línea 10). A continuación se obtienen los datos del formulario, que están almacenados en un objeto tipo *Request* accesible desde la página *JSP*. Con esos parámetros, inicializamos los correspondientes atributos de nuestro objeto de tipo *Centro* (Listado 4.4, Líneas 12-17). Por último, creamos un objeto de la interfaz *GestionCentros*, correspondiente a la capa de negocio creada anteriormente, e invocamos el método *nuevaUniversidad* (Listado 4.4, Líneas 19-22). En función del resultado de la ejecución del método, devolvemos una página notificando el éxito de la operación o reportando su fracaso (Listado 4.4, Líneas 21-26). El resto de formularios funcionarían de forma muy similar.

Tras el desarrollo de los formularios, se comprobó que todos funcionaban

Listado 4.3: Código para crear el formulario *nuevoCentro*.

```
1    <h1><a>Nuevo Centro</a></h1>
2    <form id="form_565735" class="appnitro"  method="post" action="
         nuevoCentro2.jsp">
3          <div class="form_description">
4      <h2>Nuevo Centro</h2>
5      <p>Dar de alta un nuevo centro.</p>
6    </div>
7    <ul >
8    <li id="li_1" >
9    <label class="description" for="element_1">Universidad </label>
10     <div>
11       <select class="element select large" id="element_2" name="
             universidad">
12       <option value="" selected="selected"></option>
13       <%
14       GestionUniversidades gestionUniversidades = new
             UniversidadNegocio();
15         List <Universidad> lista = gestionUniversidades.
               getUniversidades();
16           for (int i=0;i<lista.size();i++){
17             out.println("<option value=\""+lista.get(i).
                   getIdUniversidad()+"\">"+lista.get(i).getNombre()+"</
                   option>");
18           }
19       %>
20     </select>
21     </div>
22   </li>
23   <li id="li_2" >
24     <label class="description" for="element_1">Nombre </label>
25     <div>
26       <input id="element_1" name="nombre" class="element text large"
             type="text" maxlength="255" value=""/>
27     </div><p class="guidelines" id="guide_1"><small>Introduce un
           nuevo nombre para el centro.</small></p>
28   </li>
29   <li class="buttons">
30     <input type="hidden" name="form_id" value="565735" />
31     <input id="saveForm" class="button_text" type="submit" name="
           submit" value="Aceptar" />
32   </li>
```

correctamente realizando una serie de pruebas muy similares a las descritas en las iteraciones anteriores.

Por último, destacar, que tal como se prometía, este cambio en la interfaz gráfica se puede realizar sin afectar en absoluto a las capas de negocio y presentación. De igual forma, podríamos desarrollar una interfaz para dispositivos móviles, tales como iOS [21] o Android [20], sin necesidad de alterar la implementación de las capas de negocio o presentación.

Listado 4.4: Código para dar de alta un nuevo centro.

```
1 <%@page import="intercambioNegocio.CentroNegocio"%>
2 <%@page import="intercambioNegocio.GestionCentros"%>
3 <%@page import="domainObjects.Centro"%>
4 <%@ page import = "java.util.List" %>
5 <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
6     pageEncoding="ISO-8859-1"%>
7
8 <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://
      www.w3.org/TR/html4/loose.dtd">
9 <%
10 Centro c = new Centro ();
11
12 String nombre = (request.getParameter("nombre"));
13 String id = request.getParameter("universidad");
14 int idUniversidad = Integer.parseInt(id);
15
16 c.setNombre(nombre);
17 c.setIdUniversidad(idUniversidad);
18
19 GestionCentros gestionCentros = new CentroNegocio();
20
21 try{
22   gestionCentros.nuevoCentro(c);
23 }catch (Exception e){
24   response.sendRedirect("error.html");
25 }
26 response.sendRedirect("ok.html");
27 %>
```

4.6. Sumario

En este capítulo se ha descrito el proceso de desarrollo de las capas de presentación y negocio. También se ha especificado como se realizó el desarrollo de estas capas, mediante diversas iteraciones en las cuales se añadía nueva funcionalidad al sistema, de acuerdo a la metodología expuesta. Tras dar por finalizadas las iteraciones se detalló el proceso de despliegue de ambas capas que, junto a la capa de persistencia, conforman la aplicación completa. Por último se ha explicado el desarrollo de una iteración adicional en la que, a modo de ejemplo, se demostró como era posible realizar una interfaz web sin que esto afectase al resto del sistema.

Capítulo 5

Sumario y Trabajos Futuros

Este capítulo da cierre al documento y en el se describen de una manera resumida y clara los procesos que se han llevado a cabo durante el resto de capítulos. Seguidamente se plasman unas breves reflexiones sobre lo que este proyecto y su realización han significado para su autor y finalmente, y aunque el proyecto haya sido dado por concluido, se expondrán unas líneas de trabajo futuras que pueden ser abiertas en caso de que sean necesarias.

Índice

5.1. Sumario

En este documento se ha descrito el proceso seguido para llevar a cabo este proyecto. Este sumario pretende dar una visión global de todo el proceso explicado en el documento

El primer capítulo hace la función de introdución y en él se explicaron, a grandes ragos, los objetivos generales del proyecto. En primer lugar se explicó el problema a resolver así como un primer planteamiento de la solución al mismo. A continuación se habló de la metodológia utilizada en el proyecto, que finalmente fue una variante del modelo incremental [26] (Ver Figura 1.1). En ella quedan reflejadas todas las etapas que debíamos seguir para poder continuar con el proyecto y que luego serían explicadas con más detalle en siguientes capítulos.

Tras el capítulo de introducción se continuó con el capítulo correspondiente a las dos primeras etapas de nuestra metodología. En primer lugar se muestra como se realizaron los procesos de captura y especificación de requisitos, tanto funcionales como no funcionales. Para realizar la captu-

ra de requisitos se empleó la técnica de *análisis de documentos* y *análisis de procesos de negocio* [22] a través de los documentos que se nos proporcionaron y escenificaciones de los procesos por parte del cliente. Con esto pudimos elaborar el documento de especificación de requisitos que nuestra aplicación debría satisfacer. Una vez identificados los requisitos fue necesario modelarlos empleando la técnica de *casos de uso* [16] que fueron organizados según la jerarquía propuesta por Cockburn [4]. Con los *casos de uso* correctamente identificados se realizaron plantillas para poder especificarlos a un nivel adecuado de detalle para su diseño e implementación. Seguidamente se analizaron una series de requisitos no funcionales y su influencia en nuestra aplicación, para ello nos basamos en la norma ISO 25010 [15]. Una vez finalizada la primera fase de nuestra metodología se pasó a definir la arquitectura del sistema, la cual es explicada y convenientemente justificada. Se decidó utilizar una arquitectura en tres capas [10] ya que nos proporcionaba una organización modular muy resitente a los cambios típicos de los sistemas de información.

El siguiente capítulo, siguiendo con nuestra metodología, detalla los procesos de diseño, implementación, pruebas y despliegue de nuestra base de datos. Esta base de datos sería el elemento fundamental de nuestro sistema, ya que sería la encargada de almacenar la infomarción y se decidió que su implementación sería realizada en *MySql* [6] [7]. Para su diseño se creó un modelo conceptual de alto nivel mediante un diagrama entidad-relación extendido [25] [27] basándose en el documento de especificación anteriormente elaborado. Para implementar el código correspondiente a nuestro diagrama entidad-relación utilizamos las herramientas de Ingeniería Directa que nos proporcionaba *DBDesigner*, la aplicación de modelado utilizada para elaborar el diagrama. El siguiente paso fue realizar una serie de pruebas sobre nuestra base de datos para comprobar que su funcionamiento era el correcto. Las pruebas se realizaron mediante un procedimiento de caja blanca a través de varias pruebas unitarias implementadas como sentencias SQL. Se realizaron pruebas de dos tipos, con datos válidos y con datos incorrectos, para poder comprobar el comportamiento de nuestra base de datos en ambos casos. Tras pasar estas pruebas se desarrollaron las clases de acceso a datos. Estas clases nos proporcionaron una serie de interfaces bien definidas a través de las cuales se podría operar con la base de datos de manera totalmente independiente a la implementación de esta, tal y como se vió cuando se explicó el funcionamiento de la arquitectura de tres capas. Las conexiones de estas interfaces con la base de datos se realizaron a través de JDBC (*Java Database Connectivity)* [23] y las herramientas que nos proporciona para su manipulación. Finalmente se explica el proceso seguido para desplegar la capa de persistencia de nuestra aplicación. Este despliegue se realizó en un equipo especialmente adquirido para ello. Lo primero que se hizo fue instalar la suite *AppServ* la cual nos proporcionó tanto el sistema gestor de base de datos *MySql* como varias herramientas para su gestión. Una vez

creado en el servidor el esquema de la base de datos utilizando el script que se generó anteriormente se configuró el servidor creando un usuario con los permisos adecuados y permitiendo la conexión remota con la base de datos a través del *firewall* de *Windows*. Por último se preparó un sistema de copias de respaldo automatizadas utilizando para ello *PhpMyBackupPro*.

Una vez finalizada la fase de desarrollo de la base de datos dio paso un proceso iterativo mediante el cual se iban añadiendo funcionalidades al sistema con cada iteración. En este capítulo se explicó el desarrollo de dos iteraciones ya que el procediemiento para el resto de ellas era muy similar en cuanto a forma. La primera parte de cualquier iteración consistía en ver los requisitos que esa iteración iba a satisfacer para, a continuación, diseñar los formularios que les darían soporte. Una vez diseñado y completado el formulario sobre el que se estaba trabajando se pasaba a desarrollar su capa de negocio. La mayoría de las veces el desarrollo de esta capa no entrañaba demasiada dificultad ya que debido al carácter de nuestra aplicación su funcionamiento era hacer de pasarela entre las capas de presentación y persistencia. Cuando ambas capas estaban realizadas y probadas se realizaban las pruebas de integración en las que se comprobaba, mediante una serie de pruebas sistemáticas, que las tres capas funcionaban tal y como se esperaba que lo hicieran. Una vez finalizada la fase de desarrollo se explica el proceso de despliegue de la aplicación. En primer lugar se detalla el proceso de empaquetado así como los requisitos mínimos para su correcto funcionamiento. También se explica como se desarrollaron las sesiones de entrenamiento para que el cliente aprendiera a manejar la aplicación. Finalmente y para comprobar que la arquitectura de tres capas funciona correctamente y que sus capas son independientes, antes de finalizar el capítulo, se muestra como se desarrolló una pequeña interfaz web la cual se comunicaba con nuestra base de datos mediante las interfaces que la capa de negocio proporcionaba para tal fin.

En la siguiente sección aparecen una serie de conclusiones personales obtenidas tras la realización de este proyecto.

5.2. Conclusiones

Valoro muy positivamente la oportunidad que se me ha dado para llevar a cabo este proyecto.

En primer lugar me ha permitido enfrentarme a un problema real en un entorno de trabajo real. La actual manera de gestionar los alumnos de intercambio en la Escuela Universitaria de Enfermería iba a acabar siendo un problema si se seguía realizando como hasta ahora y me enorgullece pensar que gracias a la aplicación que he desarrollado dejará de serlo.

Este proyecto me ha permitido aplicar muchos de los conocimientos adquiridos a lo largo de la carrera. He llevado a cabo la planificación de un

proyecto aplicando conceptos vistos en *Ingeniería del Software* y he podido desarrollarlo y finalizarlo gracias a asignaturas como *Programación* y *Bases de Datos*. En cuanto al resto de asignaturas vistas a lo largo de la carrera, aunque no hayan tenido una implicación directa en este proyecto, me han permitido valorar varios aspectos desde un punto de vista más amplio.

Sin embargo el desarrollo del proyecto no ha sido el único reto al que me he tenido que enfrentar. El desarrollo de este documento junto a la futura defensa oral del mismo también tienen mucha importancia. Lo primero me ha servido para aprender la manera correcta de redactar un informe técnico de estas características y eso es algo a lo que, sin duda, volveré a enfrentarme en el futuro. En cuanto a la defensa creo que es también muy importante puesto que, de igual manera que lo anterior, volveré a encontrarme con situaciones en las que deba hacerlo.

Trabajar en un entorno real es muy diferente a hacerlo en ámbito académico y creo que es una experiencia muy positiva poder experimentarlo de una manera como la que se ha hecho en este proyecto ya que no deja de ser un trabajo académico pero enfocado hacia un entorno de trabajo real.

Actualmente me encuentro trabajando en otro proyecto para la Escuela Universitaria de Enfermería y quiero agradecer tanto la oportunidad que me dieron en un principio para este proyecto como la confianza depositada en mi para continuar con el siguiente.

Si duda ha sido una experiencia muy enriquecedora y positiva que me ha permitido comprobar que todos estos años de esfuerzo han merecido la pena y que me ayudará a estar un poco más preparado de cara a lo que está por venir.

5.3. Trabajos Futuros

Aunque el trabajo quedó concluido y aceptado y actualmente está siendo utilizado por el responsable de las relaciones internacionales de la Escuela Universitaria de Enfermería, es posible que en un futuro sea necesario realizar algunos cambios en su estructura. Gracias al desarrollo en tres capas utilizado la aplicación está preparada para que estos cambios tengan el menor impacto posible cuando se lleven a cabo.

A continuación veremos unos ejemplos de varios cambios que pueden requerirse en el futuro y como podrán ser desarrollados:

1. Migración de la interfaz de escritorio a una interfaz web: En el caso de que se quiera utilizar una interfaz web su desarrollo solamente afectaría a la capa de presentación, como ya se demostró en la Sección 4.5.

2. Diferentes fuentes de datos: Es posible que en algún momento en lugar de almacenar en nuestro sistema los datos de los alumnos se decidan utilizar los de la propioa base de datos de la Universidad de Cantabria.

En ese caso solo habría que modificar las clases de acceso a datos de la capa de persistencia que fueron descritas en la Sección 3.4.

3. Automatización de las convalidaciones válidas: Actualmente el sistema permite asignar cualquier pareja de asignaturas siempre que ambas pertenezcan a los centros asociados al alumno pero es el administrativo el encargado de establecer si la convalidcación es posible o no. Este proceso puede automatizarse de manera que ciertas convalidaciones no sean aceptadas por el sistema. Para llevarlo a cabo debería añadirse la lógica correspondiente dentro de la capa de negocio.

4. Límite de asignaturas a convalidar: El límite de asignaturas de cada plan de estudios puede ser un parámetro que se necesite variar en el futuro. Al tratarse de una regla de negocio bastará con modificar la capa de negocio para que solo permita añadir pares de convalidación hasta dicho límite

Como puede verse la aplicación presenta un alto grado de adaptabilidad gracias, sin duda, a su arquitectura de tres capas.

Bibliografía

[1] H. Bergsten. *JavaServer Pages*. O'Reilly Media, 3 edition, December 2003.

[2] J. Brittain. *Tomcat: The Definitive GuideJason Brittain*. O'Reilly Media, 2 edition, October 2007.

[3] A. B. Chaudhri. *XML Data Management: Native XML and XML-Enabled Database Systems*. Addison-Wesley Professional, 1 edition, March 2003.

[4] A. Cockburn. *Writing Effective Use Cases*. Addison-Wesley, 1 edition, October 2000.

[5] C. Date. *Database Design and Relational Theory: Normal Forms and All That Jazz*. O'Reilly Media, 1 edition, April 2012.

[6] P. DuBois. *MySQL Cookbook*. O'Reilly Media, 2 edition, January 2007.

[7] P. DuBois. *MySQL*. Addison-Wesley Professional, 4 edition, September 2008.

[8] R. Elmasri. *Fundamentals of Database Systems*. Addison-Wesley, 6 edition, April 2010.

[9] H. Feddema. *DAO Object Model: The Definitive Guide*. O'Reilly Media, 1 edition, January 2000.

[10] M. Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley, 1 edition, November 2002.

[11] J. Garbus. *Administrator's Guide to Sybase ASE 15*. Jones & Bartlett Publishers, 1 edition, April 2006.

[12] D. S. García. *Nociones generales de la Ley Orgánica de Protección de Datos y su Reglamento*. Tecnos, 1 edition, Febrero 2012.

[13] R. Greenwald. *Oracle Essentials: Oracle Database 11g*. O'Reilly Media, 4 edition, November 2007.

[14] J. L. Harrington. *Object-Oriented Database Design Clearly Explained*. Morgan Kaufmann, 1 edition, 2000.

[15] I. S. O. (ISO). Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models, March 2011.

[16] I. Jacobson. *Object Oriented Software Engineering: A Use Case Driven Approach*. Addison-Wesley Professional, 1 edition, June 1992.

[17] J. Lewis. *Oracle Core: Essential Internals for DBAs and Developers*. Apress, 1 edition, November 2011.

[18] T. Lindholm. *The Java Virtual Machine Specification*. Addison-Wesley Professional, 2 edition, April 1999.

[19] J. Marini. *Document Object Model : Processing Structured Documents*. McGraw-Hill/OsborneMedia, 1 edition, July 2002.

[20] R. Meier. *Professional Android 4 Application Development*. Wrox, 3 edition, May 2012.

[21] M. Neuburg. *Programming iOS 5: Fundamentals of iPhone, iPad, and iPod touch Development*. O'Reilly Media, 2 edition, March 2012.

[22] K. Pohl. *Requirements Engineering: Fundamentals, Principles, and Techniques*. Springer-Verlag, 1 edition, June 2010.

[23] G. Reese. *Database Programming with JDBC and Java*. O'Reilly Media, 2 edition, September 2000.

[24] P. J. Sadalage. *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*. Addison-Wesley, 1 edition, August 2012.

[25] G. Simsion. *Data Modeling Essentials*. Morgan Kaufmann, 3 edition, November 2004.

[26] I. Sommerville. *Software Engineering*. Addison-Wesley, 9 edition, March 2010.

[27] B. Thalheim. *Entity-Relationship Modeling: Foundations of Database Technology*. Springer, 1 edition, May 2000.

[28] A. Vyas. *Sybase Developer (ASE 15) Survival Guide*. lulu.com, 1 edition, November 2012.