UNIVERSIDAD COMPLUTENSE MADRID
Facultad de Informática

PROYECTO DE SISTEMAS INFORMÁTICOS
Curso 2005-2006

PRG-eLearn: Desarrollo y Simulación de Una Plataforma de Registro y Gestión de Información eLearning

por Oscar García Bolívar, Gonzalo Pinto Pajuelo y Raúl Rico Vega

Profesora directora: Ana M. González de Miguel

Resumen

En la actualidad, el e-learning o aprendizaje a través de medios electrónicos está en plena eclosión en lo que se refiere a las normativas y estándares universales de manejo de la información electrónica, cuando hasta hace poco tiempo cada plataforma y casi cada empresa utilizaba sus propias normas para almacenar y manipular esta información.

La estandarización de la información electrónica tiene ventajas como consecuencia de que toda ella se construye de la misma forma, respetando unas normas universales, y a la desventaja inicial de no ser bien conocida por el momento por todos los usuarios de información electrónica, podemos añadir la dificultad intrínseca en el uso de los estándares, dado que no todos los creadores de contenidos electrónicos tendrían por qué conocer estos estándares.

La propuesta de este proyecto trata de dar una solución a estos problemas por medio de una herramienta que permita registrar y gestionar información electrónica de forma sencilla y respetando el estándar IMS mientras se mantiene transparente para los usuarios, con el propósito de poder ser utilizados estos contenidos en cualquier plataforma cuyo formato de información contemple el estándar IMS.

Abstract

Nowadays, the e-learning is beginning to create its universal rules and standards of handling the electronic information, meanwhile a little time ago almost every platform or company applied its own rules in order to store y handle these contents.

Standardization of the electronic information is useful because it all is made in the same way, keeping universal rules, but so do has the problem of not being well known so far by electronic information users. We can add to this the intrinsic difficulty at standards using, because not all contents creators should know these standards.

The proposal of this project tries to give a solution to these problems through a tool which let us register and manage electronic information easily and apply the IMS standards while it is kept transparent for users, with the object of being this contents used in any platform which information format applies the IMS standard.

El grupo de trabajo que ha desarrollado este proyecto, formado por Oscar García Bolívar, Gonzalo Pinto Pajuelo y Raúl Rico Vega, autorizan a la Universidad Complutense a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria, como el código, la documentación y/o el prototipo desarrollado.

En Madrid, a 27 de septiembre de 2006,

Oscar García Bolívar Gonzalo Pinto Pajuelo Raúl Rico Vega

Palabras clave

e-learning, contenidos de aprendizaje, estándar, IMS, repositorio.

Tabla de Contenidos

1. Introducción

En la actualidad las normativas y estándares universales del e-learning siguen definiéndose, situándose en un estado relativamente inicial su aplicación, y a día de hoy siguen publicándose nuevas ideas relacionadas con formas óptimas de ordenar contenidos de aprendizaje de forma que ésta pueda ser aprovechada desde elementos (plataformas, software en general) distintos a su origen. Muchas de las plataformas de e-learning utilizadas actualmente cuentan con serios inconvenientes para exportar los contenidos que generan a otros sistemas, y esto es un problema que puede tener solución en tomar decisiones y establecer normativas con criterio, que sean cumplidas por todos, como son los estándares.

La estandarización en los contenidos electrónicos consigue que la información tenga una estructura uniforme, dentro de todas las posibilidades que hay para crear dicha información, y como consecuencia de esto, todas las herramientas pueden tener el mismo punto de partida a la hora de "comprender" el orden y la estructura de la información, y así poder utilizarla sin tener importancia el método de su creación, siempre que respete los estándares.

Utilizando los estándares nos encontramos con una desventaja salvable. El hecho de desarrollar contenidos electrónicos respetando un estándar para ello, a priori puede implicar el tener conocimientos del propio estándar, cosa no muy razonable si tenemos en cuenta la variedad en los posibles orígenes tanto de los contenidos electrónicos como de los usuarios que los crean, que pueden ser de muchísimos tipos. Pero siempre pueden simplificarse al usuario los detalles del estándar, permitiéndole a la vez toda la libertad que aquél da para crear los contenidos electrónicos.

1.1. Objetivos del Proyecto

La propuesta de este proyecto consiste en desarrollar una interfaz gráfica J2EE que facilite el registro y la gestión de información educativa en un entorno heterogéneo y complejo de eLearning. De esta forma podemos conseguir que los usuarios que registren y gestionen la información puedan dejar de depender de una plataforma en concreto y de esta forma puedan utilizar los contenidos que creen en cualquiera de ellas que respeten el estándar IMS.

De la misma manera, a través de una interfaz cómoda evitamos a los usuarios tener que profundizar en el propio estándar IMS, trabajando con unas opciones que hacen transparente toda la normativa y que en todo momento modelan los datos siguiendo el estándar.

1.2. Actividades del Proyecto

Para llegar a una solución para la propuesta, hemos seguido una serie de pasos que nos han llevado a tomar caminos determinados, de entre muchas posibilidades. Las actividades seguidas pueden separarse en dos fases:

Fase I: Exploraciones en eLearning y Diseño de la Arquitectura PRG-eLearn.

Fase II: Desarrollo y Simulación de un Prototipo PRG-eLearn.

2. Proyecto PRG-eLearn

Con este proyecto se aporta una solución a la necesidad de un espacio independiente de otras plataformas donde poder almacenar contenidos de e-learning. De esta forma, las plataformas cuyos contenidos contemplen el estándar elegido, podrían utilizar los contenidos de PRG-eLearn.

PRG-eLearn proporciona un diseño y una implementación básica, por medio de un prototipo, de las funcionalidades básicas necesarias para el registro y la gestión de esos contenidos.

2.1. Fase I: Exploraciones en eLearning y Diseño de la Arquitectura PRG-eLearn

La existencia de un marco de trabajo en donde las arquitecturas, las plataformas y los estándares avanzan en el camino de resolver los problemas derivados de la utilización de contenidos de e-Learning, derivan en el surgimiento de necesidades nuevas, como la que plantea la propuesta de nuestro proyecto.

En esta primera fase revisaremos primeramente algunas de las arquitecturas y plataformas utilizadas en la actualidad, con el objeto de tener en cuenta sus virtudes y poder llegar a una solución mejor, así como los estándares empleados, pudiendo así normalizar nuestros contenidos de e-learning, para lo que después plantearemos un diseño adecuado de una arquitectura satisfactoria.

2.1.1. Investigación del Estado Actual de las Arquitecturas eLearning

En la actualidad los sistemas LMS tienen cada uno su propio formato de estructuración de contenidos y almacenamiento de información, dada la ausencia de un estándar común hasta tiempos recientes. Por los diversos orígenes y plataformas a la hora de desarrollar contenidos, se echa de menos una estructura de repositorios adecuada, en la que puedan registrarse y gestionarse los contenidos de forma que pueda conseguirse recuperar información de manera sencilla sobre estos contenidos y utilizarse en cualquier plataforma que soporte los estándares.

En este apartado veremos algunas de las características de arquitecturas y plataformas en las que nos hemos fijado, observando sus funcionalidades más interesantes en relación a la propuesta de nuestro proyecto.

La arquitectura CORDRA

La Content Object Repository and Registration/Resolution Architecture es un modelo desarrollado por la universidad norteamericana de Carnegie Mellon. Especifica cómo diseñar e implementar sistemas con el propósito de compartir y reutilizar contenidos de aprendizaje creando federaciones locales que lleven a una estructura de contenidos de aprendizaje a nivel global.

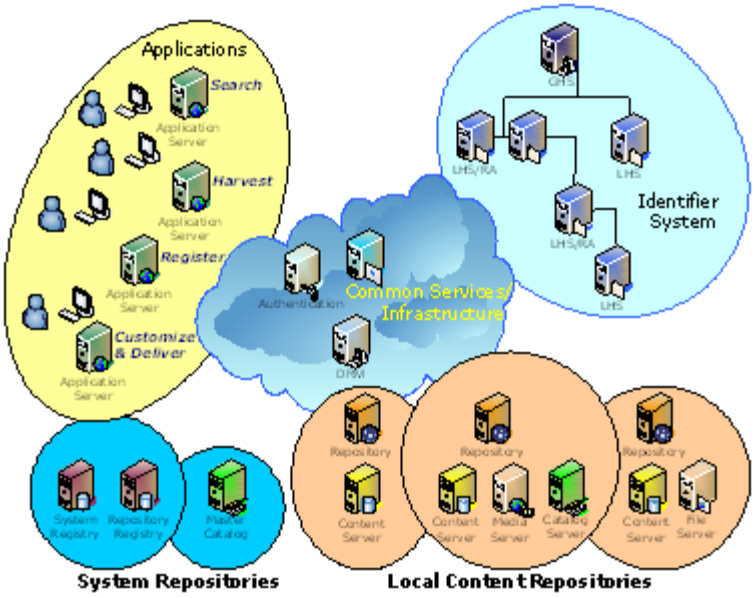
Esta arquitectura trata siempre de basarse en estándares y tecnologías existentes. Respeta el estándar SCORM, que especifica los medios para etiquetar contenidos con el objeto de su posterior búsqueda y acceso, pero no dice nada sobre cómo implementar la recuperación de contenidos. Aquí es donde incide esta arquitectura.

Los contenidos publicados en los repositorios están disponibles en un sentido amplio. Por tanto, existirán contenidos persistentes más allá de los cursos que los contengan. Para ello se hacen necesarios descriptores adecuados para los contenidos del repositorio. Los mecanismos de acceso a estos contenidos son estándar.

Dentro del modelo, además de los repositorios individuales de contenidos, una federación tiene tres sistemas de repositorios:

- un catálogo principal o registro de contenidos, que contiene metadatos de los contenidos de cada repositorio para realizar las búsquedas y los accesos;
- un repositorio registro, que contiene descripciones de todos los repositorios de la federación;
- el sistema de registro, que contiene las descripciones de procesos del modelo CORDRA y su implementación dentro de la federación.

Todos estos repositorios están registrados en el registro de repositorios, teniendo el sistema, por tanto, capacidad de auto-descripción.



Modelo de distribución de una arquitectura CORDRA

Arriba podemos ver globalmente la arquitectura de CORDRA. Los componentes clave de este diseño son:

- Los repositorios de contenidos: repositorios locales de contenidos de aprendizaje y sus datos asociados (como listados locales y metadatos).
- Los *repositorios del sistema*: repositorios del sistema CORDRA de datos del sistema, modelos, registros, etc.
- El *sistema identificador*: infraestructura para la identificación de objetos, su registro y resolución.
- *Infraestructura de servicios comunes*: núcleo de servicios técnicos y de administración usados en toda la implementación de CORDRA (autenticación, derechos de administración, etc)
- *Aplicaciones*: sistemas de aplicación e interfaces (búsqueda, acceso, autoría, distribución...) utilizados para administrar y distribuir los objetos de contenidos a los usuarios finales.

Son cualidades interesantes de esta arquitectura:

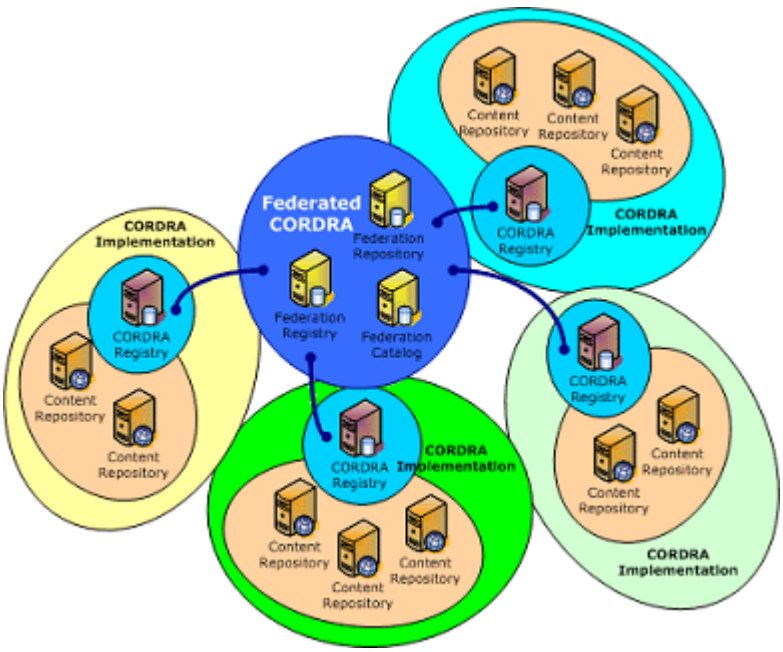
- Los contenidos de aprendizaje de cada repositorio local se gestionan según las respectivas reglas de cada repositorio local.
- Los repositorios y contenidos son registrados en el contexto de la federación para hacer posible su búsqueda, acceso y administración.

- La federación de registros es un conjunto de sistemas de repositorios que mantiene una catálogo principal con todos los metadatos de los contenidos de aprendizaje, el registro de la federación que lista todos los repositorios de la federación, y un repositorio más con el sistema de datos, los modelos, etc.

CORDRA propone una propiedad interesante como es la escalabilidad. Soporta la integración en sistemas actuales, detalle importante a la hora de estandarizar y compartir contenidos. Otra cualidad interesante es la existencia de repositorios en distintos lugares físicos (su "federación local"), aunque no tiene demasiada importancia, ya que la información de cada contenido registrado es a lo que accedemos primeramente, y ésta a su vez contiene información sobre la localización de los contenidos en sí. Entonces no es necesaria la aparición de un listado con toda la infomación de los repositorios en cada repositorio, sino que accedemos a ésta en un repositorio de contenidos centralizado.

En esta arquitectura contamos con un conjunto de operaciones principales que permiten el acceso y registro de contenidos en un conjunto de repositorios CORDRA, como

- creación y registro de contenidos.
- Búsqueda y localización a objetos de contenidos
- Acceso y recuperación de objetos de contenidos
- El registro de un nuevo repositorio, que sólo puede realizarla un administrador



El sistema Chasqui

Desde la construcción de objetos de aprendizaje relacionados con la arqueología, Chasqui parte de dificultades similares a las nuestras, como son la necesidad de desarrollar materiales educativos en formatos independientes de plataforma y con la posibilidad de mejorar la eficiencia y utilidad de los mismos reutilizando materiales.

Chasqui se basa, desde la utilización de un estándar común, en la posibilidad de reutilizar los mismos contenidos en contextos diferentes. Si un material se diseña para ser utilizado en múltiples contextos, puede ser reutilizado más fácilmente que el material que tiene que ser reescrito para cada nuevo contexto. Considera igualmente la definición de metadatos con capacidad expresiva suficiente como algo fundamental para ello.

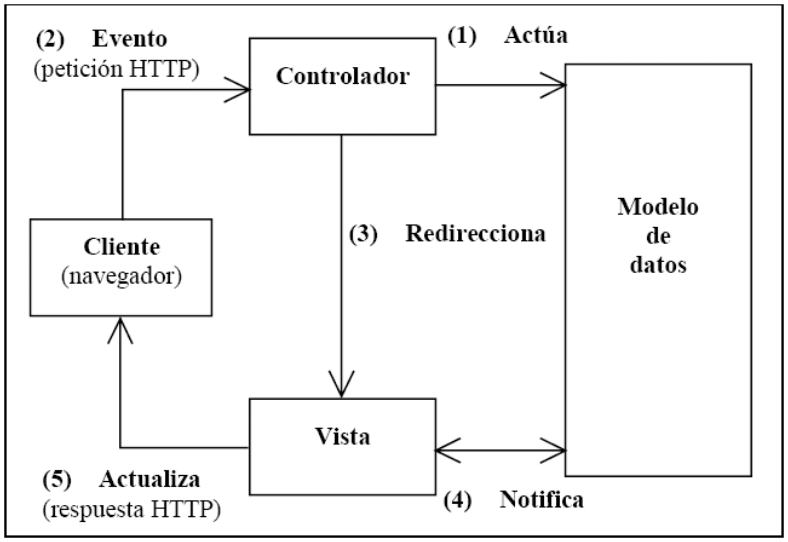
Este sistema tiene en cuenta el concepto de objeto virtual, que se compone de:

- datos que describen todas las características físicas o conceptuales del objeto real.
- metadatos que describen y clasifican el objeto desde, en el caso de Chasqui, el punto de vista del aprendizaje,
- recursos, que son colecciones de representaciones virtuales de ciertos aspectos, ya sean conceptuales o físicos del proyecto. En el caso de Chasqui se corresponden con archivos de diverso tipo: imágenes, texto, audio...

El desarrollo de los primeros prototipos de Chasqui llevó a varios problemas, que se resolvieron en buena medida al adoptar como arquitectura el modelo-vista-controlador, que divide a ésta en capas para separar funcionalidades:

- *Modelo*. Lleva a cabo la funcionalidad y el estado de la aplicación. Aquí se define la estructura de los datos relevantes al dominio y se definen las funciones que operan con esos datos. En esta parte se integra el acceso, recuperación y modificación de la información relativa a los objetos virtuales y los metadatos asociados a cada objeto; de esta forma se consigue la independencia entre el modelo y el resto de la aplicación.
- *Vista*. Contiene la interfaz del modelo y los mecanismos de interacción con el usuario; la vista puede acceder al estado del modelo, pero no puede modificarlo; hay que “informar” a la vista cuando se produzca algún cambio en el modelo.

- *Controlador*. El controlador establece la conexión entre los elementos de la interfaz y los datos que estos representan. El controlador implementa el flujo de la aplicación, se ejecuta en el servidor y depende de la situación actual del modelo y de las acciones realizadas por el usuario en la vista.



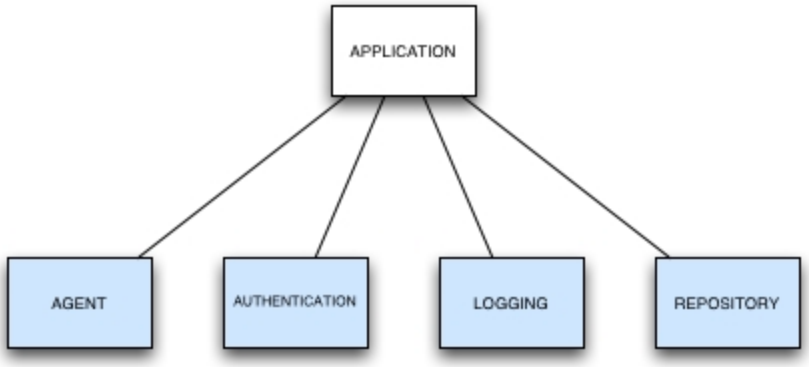
Esquema del modelo-vista-controlador planteado en Chasqui

Chasqui II contempla la posibilidad de comunicación entre arquitecturas diferentes a través de interfaces que se comunican con un lenguaje estándar como es XML, por medio de web services, detalle que contemplará el diseño de nuestro proyecto.

La arquitectura OKI y el repositorio OSID.

La Open Knowledge Iniciative define puntos de interoperatibilidad entre componentes de aprendizaje electrónico, teniendo en cuenta diseños incrementales que vayan a utilizar esta arquitectura. Su repositorio OSID (Open Service Interface Definition) tiene interfaces para la integración entre aplicaciones (por parte de los consumidores) y repositorio (por parte de los proveedores), por encima del lenguaje de programación en que estén implementadas aquéllas.

De un primer vistazo a la especificación de OSID, observamos una serie de servicios diseñados para ser usados por aplicaciones, tal como muestra la figura inferior. Aunque la figura es simple, la aplicación en este caso debería “entender” las dependencias entre estos módulos así como ser la encargada del intercambio de datos entre ellos. El conocimiento de la aplicación debería llevarnos a unir aplicación e implementaciones.

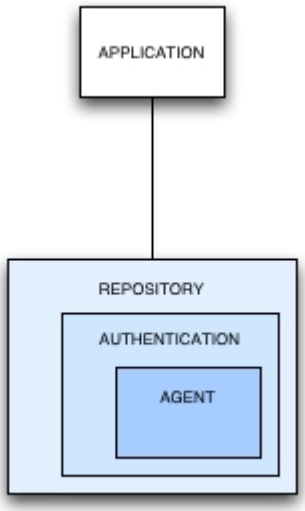


Un uso incorrecto de los OSID

Observemos este ejemplo de servicio repositorio cuya implementación incluye un servidor remoto. Utilizando directamente el OSID de autenticación, la aplicación decide qué formulario de autenticación necesita el servicio repositorio. Pero el mecanismo de autenticación es específico de la implementación del repositorio y debería decidirlo ésta. Además, encargando la autenticación a la aplicación, entonces estorba a la implementaciones de los repositorios que comparten el mismo sistema de autenticación.

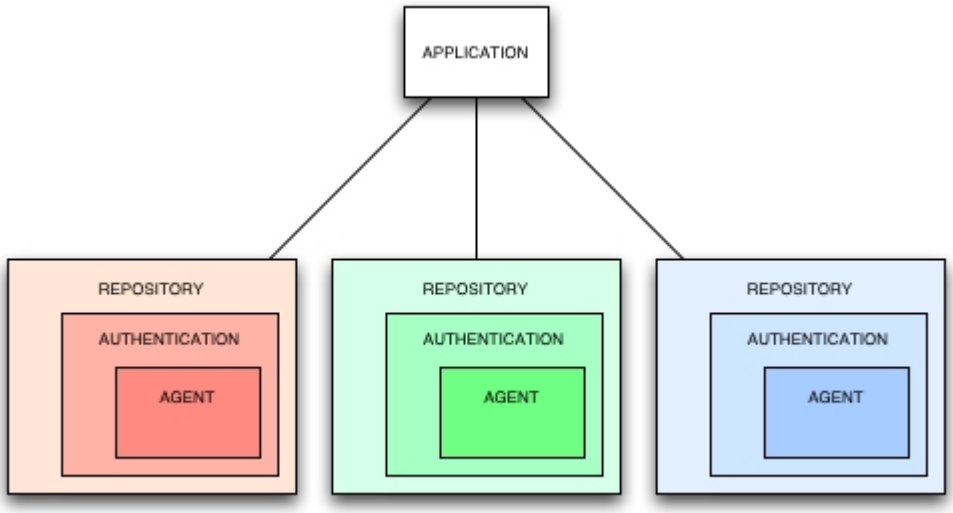
El hecho de que el repositorio necesite autenticación es una característica específica de la especificación, que debería ser enmascarada desde la aplicación. Es responsabilidad del repositorio hacer lo necesario para implementar el servicio.

Supongamos que una aplicación quiere utilizar un repositorio. En la figura de debajo vemos que la aplicación necesita saber del repositorio. Aunque no siempre será necesaria la autenticación. Los repositorios, en este caso, son parte de la implementación del OSID: la implementación del repositorio contiene el OSID de autenticación.



Un uso mejor de OSIDs

Las implementaciones podrían usar diferentes OSIDs para cumplir sus propias especificaciones de servicio. En el siguiente ejemplo, una aplicación accede simultáneamente a varios servicios de repositorio cuyas implementaciones dependen de diferentes servicios de autenticación. El hecho de que la autenticación exista dentro del repositorio es transparente desde la aplicación mientras que la existencia del agente es a su vez transparente desde el repositorio. Además, cada implementación no necesitaría siempre tener por debajo un OSID a su medida. Con un diseño adecuado, sería modulable. Entonces la arquitectura OKI define puntos de interacción entre LMS's, haciendo incrementales las arquitecturas. Una cualidad que tendremos en cuenta es la independencia de la tecnología.



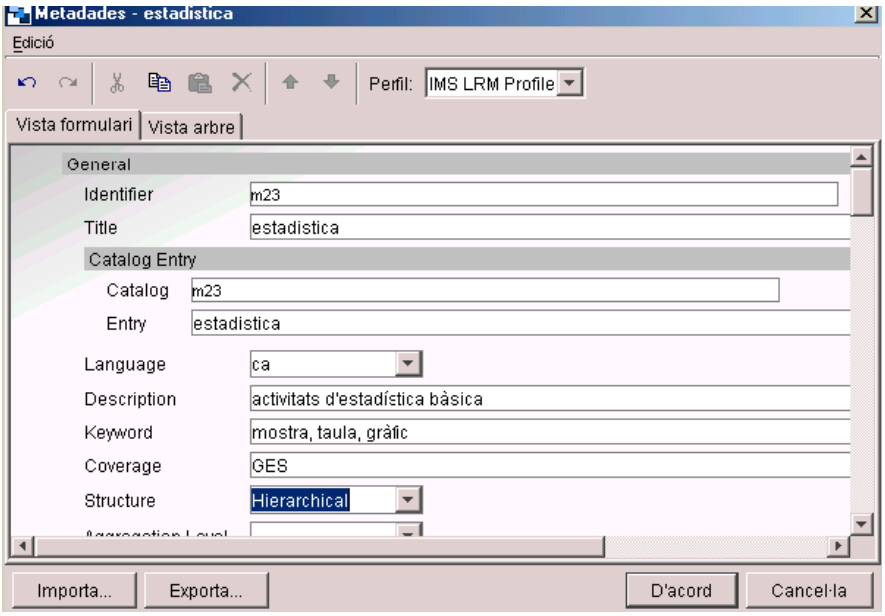
Una aplicación manejando distintas implementaciones

Los paquetes de la herramienta Reload

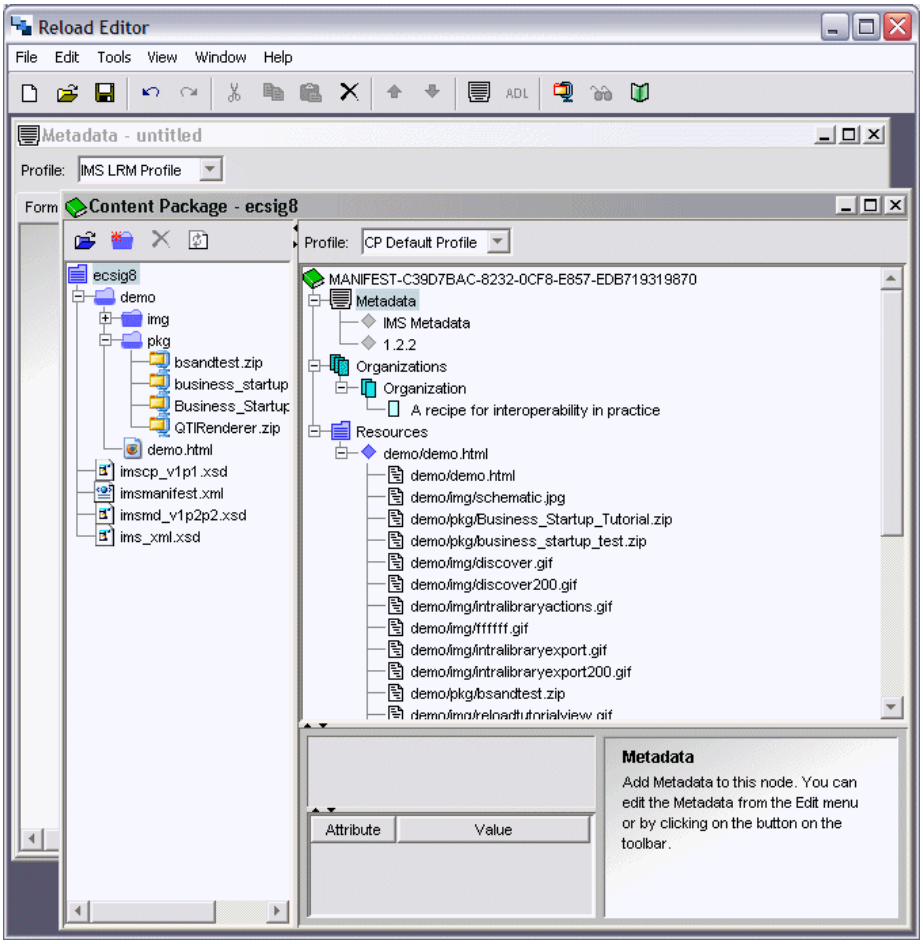
Reload es un empaquetador de contenidos que se ajusta a los estándares y que permite añadir metadatos a los contenidos. La idea es que alguien crea los objetos de aprendizaje, les da una estructura que piensa que facilita el aprendizaje y lo empaqueta en un único fichero. Este paquete se deja en un repositorio (es importante la idea de compartirlos) o bien se distribuye por la red y, para que no se pierda la organización que le dio el autor, va acompañado de un manifiesto, es decir, de un documento donde queda reflejado el contenido y el orden o secuencia con que se puede seguir para lograr los conocimientos. El contenido del manifiesto son, por lo tanto, metadatos, es decir datos que proporcionan datos de los objetos de aprendizaje que contiene el paquete.

Lo que está estandarizado es el manifiesto, que no es otra cosa que un documento XML donde quedan reflejados los metadatos, es decir, la información sobre la estructura en que se organizan y los objetos de aprendizaje. Este manifiesto (el fichero imsmanifest.xml) es interpretado por unas hojas de estilo que transforman los metadatos escritos en lenguaje XML a lenguaje comprensible por los humanos.

Reload nos ayuda aconstruir un paquete de contenidos organizados, y a acompañar los contenidos con metadatos, pero funciona como complemento de otras herramientas, ya que no nos permite registrar en un repositorio o gestionar estos paquetes.



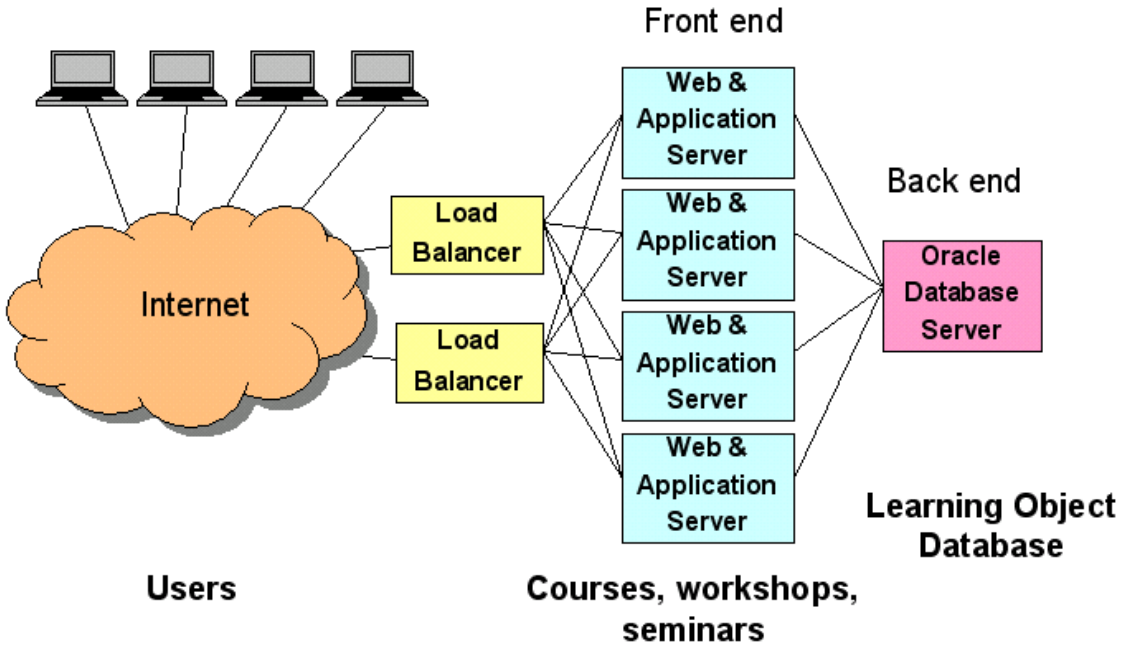
Fijación de algunos metadatos en un paquete Reload



Estructura de un paquete Reload

La plataforma WebCT y sus e-packs

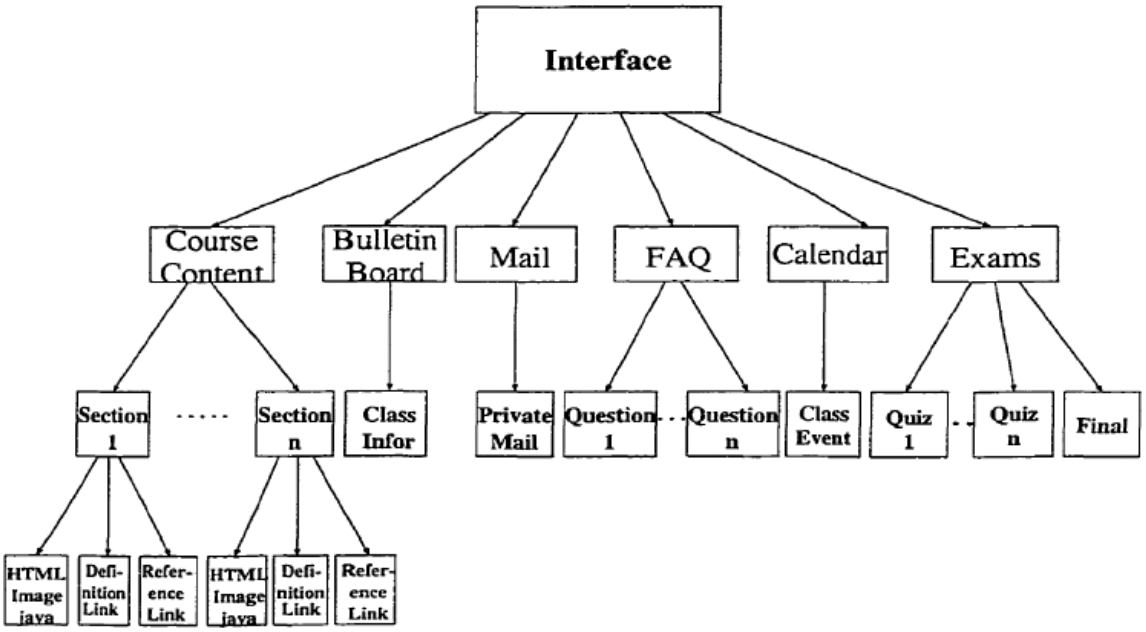
Algunos detalles sobre la arquitectura de WebCT son su entorno completamente web y los datos de contenidos y configuración son archivos de texto pero sin XML.



Esquema de una arquitectura WebCT

El sistema WebCT utiliza una arquitectura de cliente/servidor. El modelo se basa en la distribución de funciones entre dos tipos de procesos independientes: cliente y servidor. El cliente interactúa con el servidor pidiéndole determinados servicios en nombre del usuario. El servidor proporciona los servicios a los clientes. Servidor y cliente pueden ser la misma máquina distintas o máquinas distintas conectadas por una red. El sistema WebCT consiste en el software WebCT y una serie de archivos jerarquizados por cada curso. El usuario accede a los datos a través de un navegador web. Todo el software de WebCT está en uno o varios servidores y se ejecuta desde ellos, lo que significa que cualquier cambio en los cursos puede apreciarse por los estudiantes desde el momento en que se haga. Nos proporciona un medio con varios tipos de distintas herramientas para crear lecturas, exámenes, grupos de discusión... y siempre con un acceso a través de usuario y contraseña.

La siguiente figura nos muestra la estructura interna de un curso WebCT, que es visible de esta forma sólo al diseñador del curso. El diseñador del curso crea materiales y configura el curso manipulando enlaces (que en el diagrama son los nodos).



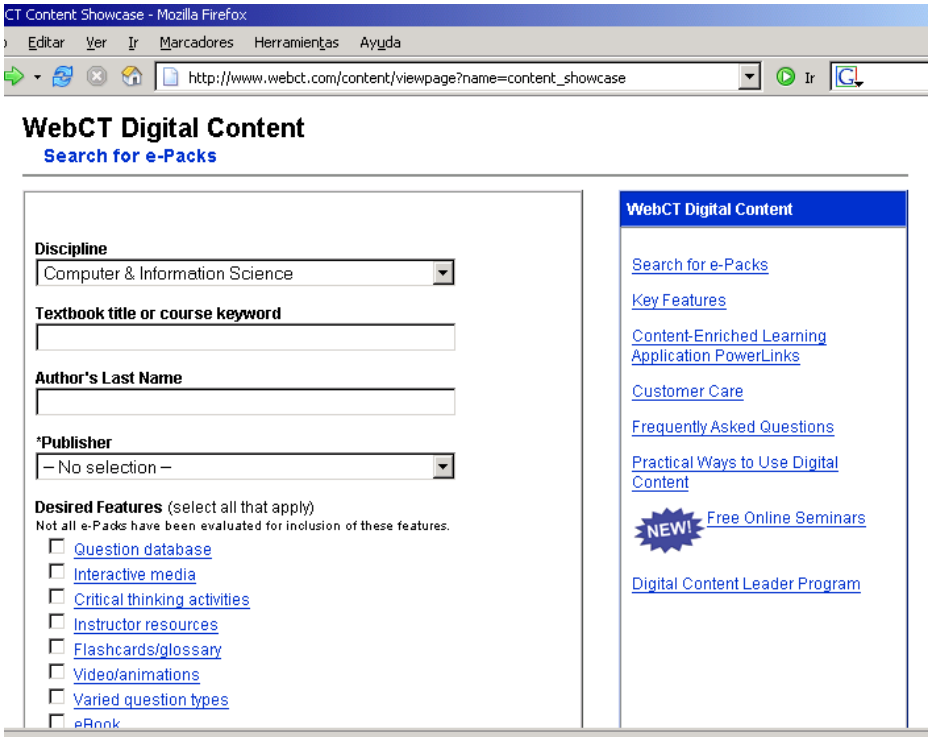
Estructura jerárquica de un curso WebCT

Los diseñadores de cursos deben saber trabajar con las restricciones funcionales de la arquitectura WebCT. La base de datos WebCT no soporta integración con otras bases de datos relacionales, aunque pueden importarse datos desde algunos formatos como es CSV. La capacidad de interacción con otras infraestructuras de eLearning está por tanto bastante limitada.

Una restricción importante es la imposibilidad de crear usuarios “a la carta”, es decir, con los privilegios exactos, que se deseen, ni uno de más ni uno de menos. Se nos restringen a tres tipos de usuario que no satisfarán las necesidades de registro. El usuario administrador tiene la capacidad de crear usuarios, y gestionar contenidos, pero esta gestión no comprende el uso de información sobre dichos contenidos para su gestión.

Respecto a contenidos prediseñados, desde la página de WebCT existe una opción que nos permite acceder a paquetes predefinidos de contenidos. Estos contenidos tienen una gran variedad, pero siempre partiendo de la premisa de que son fijos, o al menos no puede gestionarlos un administrador de WebCT.

La siguiente imagen nos presenta la forma de obtener contenidos, que si a priori es buena en cuanto a que podemos hacer un primer filtrado por temas y después buscar por palabras clave, más adelante entraremos más en detalle sobre esto.



Búsqueda de un e-pack en el repositorio de WebCT

La forma de acceder a estos paquetes viene precedida por una descripción de los paquetes por título, disponibilidad, autor, editor y tipos de contenidos. Pero no se nos da la capacidad de gestionar nuestros propios contenidos, por lo que nadie podrá acceder a ellos de la misma forma en la que se accede a los e-packs.

2.1.1.2. Estándares y Tecnologías eLearning

Introducción

Existe una gran variedad tanto de LMS (Learning Management Systems) como de cursos. Esto hace necesaria una normativa que haga compatibles los sistemas y cursos de diferentes fabricantes para lograr los siguientes objetivos:

- Que un curso de un fabricante pueda ser cargado en el LMS de otro.
- Que los resultados de las actividades de los usuarios de un curso puedan ser registrados por el LMS.

Los estándares de e-learning que se están desarrollando se centran en diferentes aspectos:

- **Contenido del curso**. Estructura y empaquetamiento de los contenidos y seguimiento de los resultados. Los contenidos son empaquetados como objetos de aprendizaje (learning objects).
- **El alumno**. Almacenamiento e intercambio de información del alumno, competencias del alumno, privacidad y seguridad.
- **Interoperabilidad**. Integración de componentes del LMS e interoperabilidad entre distintos LMS.

Más concretamente, con la aplicación de dichos estándares se persigue lo siguiente:

- Durabilidad. Evitar que los cursos se queden obsoletos rápidamente.
- Interoperabilidad. Permitir el intercambio de información entre LMS diferentes.
- Accesibilidad. Permitir un seguimiento del comportamiento de los alumnos.
- Reusabilidad. Que los cursos y objetos de aprendizaje puedan ser reutilizados con diferentes herramientas y plataformas.

Las ventajas que proporciona la aplicación de los estándares a los fabricantes y consumidores de e-learning son las siguientes:

- La **independencia** de una única tecnología.
- La **reducción de costes**, al aumentar la oferta de cursos disponibles y evitar en muchos casos desarrollos a medida.

Estándares de e-learning

AICC (Aviation Industry CBT Comitee)

Fue la primera organización que creó un conjunto de normas que permitiese el intercambio de cursos CBT (Computer Based-Training) entre diferentes sistemas. Sus especificaciones cubren nueve áreas principales, que van desde los objetos de aprendizaje hasta los LMS.

La AICC ha publicado varias guías, entre las que destaca la AGR 010, que habla de la interoperabilidad de las plataformas y los cursos. Esta guía resuelve dos problemas fundamentales:

- La **carga** en un LMS de cursos de otros fabricantes. Esto se consigue definiendo el curso como una entidad totalmente independiente de la plataforma, y creando un sistema de descripción del curso que pueda ser entendido por cualquier plataforma.
- La **comunicación** entre el LMS y el curso, de forma que el curso pueda obtener información sobre el usuario y transmitir los resultados de las interacciones y evaluaciones realizadas por el mismo a la plataforma. Esto se logra mediante la definición de un mecanismo de comunicación entre el curso y la plataforma, y un conjunto de datos que deben ser transmitidos del curso a la plataforma y viceversa. La AICC describe dos mecanismos: uno basado en el protocolo http y otro mediante una API.

IEEE Learning Technologies Standards Comité (LTSC)

Es un organismo que promueve la creación de una norma ISO para e-learning. Básicamente mejoró el trabajo de la AICC creando el concepto de *metadatos*, esto es, información sobre los datos, ofreciendo una descripción más detallada que la ofrecida por la AGR 010 de la AICC sobre los contenidos de un curso.

IMS Global Learning Consortium, Inc.

Es un consorcio formado por miembros provenientes de organizaciones educacionales, además de empresas públicas y privadas. El IMS definió un tipo de fichero XML para la descripción de los contenidos de los cursos, de tal forma que cualquier LMS pueda, leyendo su fichero de configuración IMSMANIFEST.XML, cargar el curso.

Las principales iniciativas de este organismo son las siguientes:

- ***Learning Object Metadata* (*LOM*)**. Trata de cómo los contenidos deben ser identificados o "etiquetados" y sobre cómo se debe organizar la información de los alumnos, de forma que se puedan intercambiar entre los distintos servicios involucrados en un LMS.
- **Empaquetamiento de Contenidos (*Content Packaging*)**. Habla de cómo describir y empaquetar material de aprendizaje, ya sea un curso individual o una colección de cursos, en paquetes portables e interoperables.
- **Interoperabilidad de Preguntas y Tests (*Question and Test Interoperability*, *QTI*)**. Propone una estructura de fichero XML para codificar preguntas y tests online, con la finalidad de permitir el intercambio de estos tests y datos de evaluación entre distintos LMS.
- **Empaquetamiento de Información del Alumno (*Learner Information Packaging*, *LIP*)**. Define estructuras XML para el intercambio de información de los alumnos entre LMS y otros sistemas utilizados en el proceso de aprendizaje.

- **Secuencia Simple (*Simple Sequencing*)**. Define reglas que describen el flujo de instrucciones a través del contenido según el resultado de las interacciones de un alumno con el contenido. Esta representación puede ser creada manualmente o a través de herramientas, y posteriormente ser intercambiada entre sistemas diseñados para entregar componentes instruccionales a los alumnos.
- **Diseño del aprendizaje (*Learning Design*)**. Investiga sobre las maneras de describir y codificar las metodologías de aprendizaje incorporadas en una solución e-learning.
- **Almacenes Digitales (*Digital Repositories*)**. Investiga sobre la interoperabilidad entre almacenes digitales.
- **Definición de competencias (*Competency Definitions*)**. Busca crear una manera estandarizada de describir, referenciar e intercambiar definiciones de competencias. Aquí el término competencia es usado en un sentido muy general, que incluye habilidades, conocimiento, tareas y resultados de aprendizaje.
- **Accesibilidad (*Accesibility*)**. Promueve el contenido de aprendizaje accesible a través de recomendaciones, guidelines y modificaciones de otras especificaciones.

ADL SCORM

ADL (Advanced Distributed Learning) es una iniciativa que trató de reunir lo mejor de las anteriores, en concreto, el sistema de descripción de cursos en XML de la IMS y el mecanismo de intercambio de información mediante una API de la AICC.

Las especificaciones de SCORM se organizan como “libros” separados. A continuación resumimos el contenido de cada uno de ellos.

Libro 1: ***Scorm Overview***. Contiene una descripción general del estándar.

Libro 2: ***Scorm Content Aggregation Model***. Contiene una guía para identificar y agregar recursos dentro de un contenido de aprendizaje estructurado.

Libro 3: ***Scorm Run-Time Environment***. Incluye una guía para lanzar contenidos y hacerles un seguimiento en un entorno web.

SCORM distingue entre Learning Management Systems (LMS) y Shareable Content Objects (SCO). Un SCO es un objeto de aprendizaje reusable y estandarizado.

2.1.2. Investigación de Tecnologías J2EE y Web Services

Se ha realizado un estudio relacionado con las diferentes tecnologías J2EE disponibles actualmente, incidiendo en el estudio de los Web Services debido a que son posibles tecnologías para el desarrollo de la Plataforma de Registro y Gestión que se pretende diseñar.

Teniendo en cuenta las virtudes de las tecnologías J2EE nos decantamos por éstas por su fiabilidad y portabilidad a cualquier superficie donde fuera a instalarse la Plataforma. Inicialmente, resumimos los principales elementos que integran estas tecnologías con sus características y funciones principales:

- **JSP:** En orden de la utilización de la Plataforma vía Web que es como se pretende acceder, utilizamos este tipo de tecnología para desarrollar páginas Web dinámicas. A favor podemos decir que conocemos la tecnología y su fiabilidad e integración con los demás módulos de la aplicación. Podríamos decir que lo utilizamos para desarrollar la Vista de la aplicación.

- **JAVA:** Con esto nos estamos refiriendo a clases Java que implementarán la Lógica de negocio de nuestra aplicación y sabemos ya cuales son las ventajas que aporta este lenguaje a la hora de afrontar cualquier desarrollo, como puede ser portabilidad, fiabilidad, conocimiento del lenguaje por nuestra parte, documentación asociada, etc. Es obvio su inmediata afinidad con las otras tecnologías que vamos a estudiar para diseñar el sistem como son: jsp, struts, web services, etc.

- **Web Services:** Resumimos los principales elementos y características en este punto.

    - **UDDI** es un repositorio donde se registran cada uno de los documentos que describen cada Web Service determinado, y que describen los métodos proporcionados por ellos y la forma de acceso. Estos documentos están implementados en un lenguaje específico, llamado WSDL.

    - **WSDL** se refiere a archivos o ficheros XML formados de una manera determinada, donde se especifican los métodos que ofrece el Web Service, así como una definición de cómo utilizarlos y qué parámetros serían necesarios. Esto lo podemos relacionar con los elementos de aprendizaje, que llamamos ***Manifiestos*** y que contienen el XML donde indicamos cómo se llama el recurso pedagógico y otros apectos y definiciones del mismo, además de los archivos físicos del recurso. El WSDL está asociado con ese XML de definición del recurso o elemento de aprendizaje que estamos definiendo y que será el que se registre en la plataforma, de forma similar a un Web Service que se registra en el UDDI.

    - Nuestra plataforma es de Registro y Gestión de objetos de aprendizaje, por tanto no tenemos que realizar entrega de los elementos definidos en el objeto virtual y no se va a trabajar elementos **SOAP.**

2.1.3. Análisis de la Información eLearning

En este apartado describiremos el tipo de información que manejará nuestra plataforma. Cabe indicar que la información que manejaremos consiste en referencias a contenidos y no a contenidos directamente.

En los siguientes apartados describiremos los objetos de aprendizaje de nuestra plataforma, las funciones de registro y gestión que realizaremos con ellos y su relación con lo estándares de IMS y la tecnología Web Services.

2.1.3.1. Objetos de aprendizaje

En este apartado describiremos los objetos de aprendizaje de nuestra plataforma de forma genérica, sin entrar en detalle de cómo su estructura se relaciona con las funciones de registro y gestión.

Estructura general de un paquete de contenidos.

El paquete de contenidos de nuestra plataforma se basa en la versión 1.1.4 de la especificación *IMS Content Packaging*. Para los metadatos nos basamos en la versión 1.2.1 de la especificación *IMS Learning Resource Meta-Data*. En concreto, y dado que utilizamos XML, nos basamos en los documentos *IMS Content Packaging XML Binding* en su versión 1.1.4 para el paquete de contenidos y en *IMS Learning Resource Meta-Data XML Binding* en su versión 1.2.1 para los metadatos.

No obstante, debemos aclarar que el seguimiento de estas especificaciones está supeditado a los objetivos que debe cumplir nuestra plataforma, y no pretende ser una implementación completa de todos los aspectos que incluyen dichas especificaciones. A continuación hacemos una descripción de alto nivel de nuestro paquete de contenidos u objeto de aprendizaje y de cómo aplicamos los estándares de IMS en su definición.

El manifiesto de nuestro paquete contiene todos los elementos recomendados por el estándar *IMS Content Packaging*, excepto el elemento <file>. Por otra parte, no introducimos etiquetas nuevas, de modo que no es necesario generar un nuevo esquema, aumentando la interoperabilidad presente y futura.

En cuanto a los metadatos, tampoco introducimos nuevos elementos. Recordemos que según el estándar pueden aparecer en 5 secciones distintas del manifiesto:

- **<manifest>**
- **<organization>**
- **<item>**
- **<resource>**
- **<file>**

A continuación haremos un estudio de qué metadatos son apropiados incluir en cada sección del manifiesto para hacer un paquete virtual lo más genérico posible, con independencia de los que finalmente incluiremos en el prototipo de PRG-eLearn. Dado que muchos metadatos pueden ser utilizados de diversas maneras, sólo evitamos la aparición de un metadato donde su uso esté claramente fuera de lugar.

A continuación comentamos en detalle qué metadatos podrían aparecer en qué secciones del manifiesto de un paquete de nuestra plataforma y las razones que nos han llevado a considerarlo así.

Sección <manifest>

En esta sección se incluyen metadatos para describir el objeto en su conjunto. Permitiremos añadir todos los metadatos definidos por IMS Learning Resource Metadata, excepto los siguientes:

- Elemento <structure> de <general>. La razón de no incluir este elemento es que sirve únicamente para describir la organización subyacente del paquete (jerárquica, lineal, etc.), lo cual en todo caso debe hacerse en la sección <organizations>.
- Elemento <duration> de <technical>. Podría utilizarse para definir el tiempo que se necesita para trabajar con el objeto, pero dado que ya existe un metadato para este fin (el elemento <typicallearningtime>), no tiene sentido utilizar este atributo para describir un objeto en su conjunto.

Sección <organization>

En esta sección se describe una organización particular de los contenidos del paquete. El único elemento adecuado es <structure>, de <general>. En nuestro paquete la estructura será siempre jerárquica (en forma de árbol) y por defecto IMS-CP asume dicha organización, por lo que no utilizaremos este elemento.

Sección <item>

Representa un nodo dentro de una organización, y puede ser hijo tanto de <organization> como de otro <item>. En general, un <item> referenciará a un recurso (esto es, un elemento <resource>) mediante su atributo identifierref, y su valor será el atributo identifier del elemento <resource> referenciado. Por lo demás, un <item> sólo indica una posición dentro de una organización. Es por eso que en nuestra plataforma los metadatos irán asociados a los elementos <resource>, y no a los elementos <item> que los referencian.

Sección <resource>

Describe un recurso que utiliza el paquete. Más concretamente, es un conjunto de ficheros que pueden ser accedidos mediante una una URL o ser internos al paquete. Un <resource> posee un punto de entrada al recurso, que es la localización de uno de los ficheros que lo contiene. Un recurso también puede estar formado por un único fichero. Los metadatos que podríamos incluir son los siguientes:

- **<catalogentry>** de **<general>** y todos sus elementos descendientes. Según el documento "IMS Content Packaging Best Practice and Implementation Guide", es recomendable utilizar metadatos para la búsqueda e idexado de ficheros en un repositorio. Decidimos utilizar estos metadatos para este fin también para los recursos.

- **<metametadata>** y todos sus elementos descendientes, por si se desea describir los metadatos.

- **<technical>** y sus elementos descendientes. Estos elementos son especialmente adecuados para describir recursos y ficheros.

- **<learningresourcetype>** de **<educational>** debido a que puede tomar valores adecuados para un recurso concreto, como "Figure" o "Graph".

- <**classification>,** para indicar los contenidos accesibles y no accesibles a un usuario.

Sección <file>

Representa un fichero de un recurso. Dado que no consideraremos el uso de ficheros locales no comentaremos los posibles metadatos que pueden incluirse en esta sección.

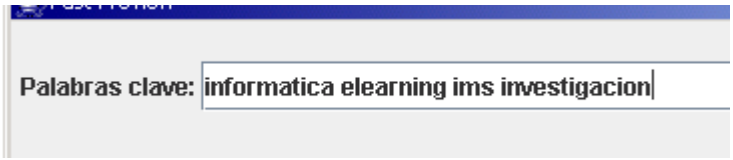
Descripción de los contenidos.

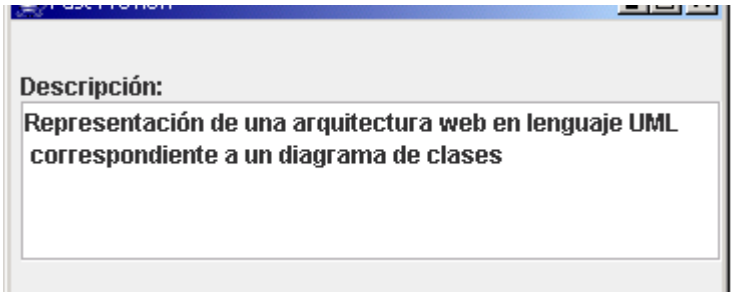
Una vez descrita la estructura general de un paquete de contenidos, entraremos en detalle en la forma de disponer cada uno de dichos contenidos. Uno de los cometidos importantes de registrar la información electrónica es adjuntar a los contenidos unos metadatos que los describan. La propuesta para construir paquetes de contenidos es dar la posibilidad de añadir dichos metadatos con todas las posibilidades en cuanto a la descripción de los contenidos se refiere que nos ofrece IMS en el documento *IMS Learning Resource Meta-Data XML Binding Version 1.2.1*. Como la descripción de los contenidos a través de las reglas que plantea IMS no es sencilla, se simplifica la forma de añadir estos metadatos, que se hará a través de una interfaz cómoda que hará más sencillo el procedimiento y añadirá las etiquetas al archivo XML según los datos que vaya aportando el autor.

De esta forma, el autor rellenará los datos relacionados con los tipos de información que define el citado documento de IMS, según las características de cada tipo de información. Podrá encontrar espacios apropiados para añadir información obligatoria u opcional de cada elemento.

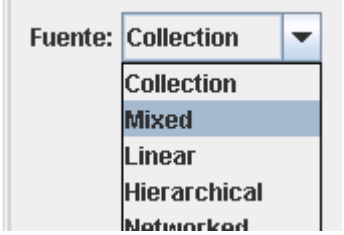
La unidad básica de contenido consiste en *una referencia a otro contenido* (y no un contenido directamente). Las informaciones que deben aparecer obligatoriamente en cada contenido podrían ser incluidas de dos formas distintas:

- libremente en forma de texto, ya sea en formato corto o extenso.

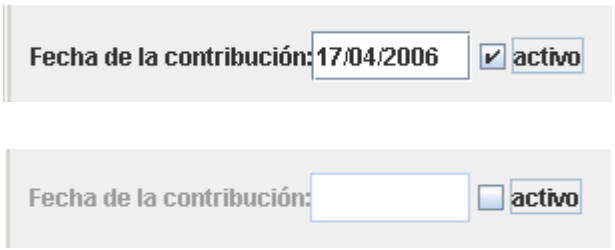




- Pudiendo elegir dentro de varias opciones, sin posibilidad de quedar vacío.



Otras informaciones pueden ser opcionales y el autor decidirá en cada caso si se incluirá o no. En las imágenes siguientes observamos cómo puede decidirse incluir o no, de forma cómoda, un metadato para un dato concreto.



2.1.3.2 Perfiles de Usuario de la Plataforma

A continuación, vamos a comentar los diferentes perfiles de usuario que van a interactuar con la Plataforma y cuáles serían las posibilidades y acciones que podrían realizar en la misma.

Administrador del Sistema-Infraestructura

Se encarga de la gestión del Sistema al margen de los procesos de registro y gestión de objetos e-Learning que tendrá la plataforma. Su tarea podría ser la administración de los usuarios del sistema, con los siguientes posibles casos de uso:

- Crear nuevos usuarios de los perfiles que sean requeridos: Gestores, registradores, etc.
- Modificar datos de usuarios ya creados.
- Eliminar usuarios no necesarios, erróneos y/o obsoletos.
- Administración del Servidor de aplicaciones donde se ejecuta la plataforma: comprueba que la aplicación funciona correctamente y arregla posibles fallos o cuelgues de la aplicación.

Hay que tener en cuenta que este tipo de usuario del sistema se encargaría de tareas que no son las principales del objetivo de nuestra plataforma de Registro y Gestión pero que serían imprescindibles para su implantación y correcto funcionamiento una vez que se desarrollara el prototipo de la plataforma, y por tanto, de dichas funciones se encargarán los desarrolladores del proyecto, en este caso.

Este tipo de usuario no se implementará en la parte del Diseño e Implementación del Prototipo de la Plataforma debido a que sólo vamos a implementar alguna de las funciones más importantes que están más relacionadas con el registro y gestión de los objetos virtuales que con la administración de usuarios, pero que no deja de ser importante para el correcto funcionamiento de la Plataforma en el caso en que se implementaran en el Prototipo todas sus funciones.

Registrador de Objetos

Definimos al usuario básico de nuestra Plataforma de Registro y Gestión al margen de los posibles usuarios finales que serían los que podrían utilizar los objetos registrados y que serán comentados más adelante. El usuario registrador se encarga únicamente de las funciones típicas que son las asociadas al registro de los objetos virtuales:

- Creación de un objeto virtual con la información y características que cada usuario registrador crea oportunas. Aquí se determinan los datos del objeto.

- Modificación de los Objetos Virtuales propios que se hayan creado o ya hayan sido registrados. Tipos de objetos:

    - Objetos que no estén completos aún y no se hayan pasado al Registro (publicado).

    - Objetos con errores tanto en la estructura y/o en los contenidos, referencias, información, etc., descubiertos en la fase de validación y que haya sido avisado el registrador para solventarlos.

    - Objetos que estén ya publicados o registrados en el Repositorio y que quieran ser modificados por el registrador propietario del objeto.

- Eliminación de objetos virtuales propios creados y/o registrados.

- Consulta de objetos virtuales propios y podemos dar la posibilidad de consulta de otros objetos virtuales ya registrados en la plataforma según el nivel de privilegios del registrador y/o del objeto, así podría comparar sus propios objetos con la estructura de otros que consulte o para comprobar que no se repiten objetos ya registrados.

Todos los objetos virtuales que se registren en la plataforma tienen que ajustarse a los estándares IMS-CP en los que nos basamos para desarrollar los objetos y que determinan la estructura y las características que tienen que cumplir.

Todo esto será revisado y validado por el Gestor de estructura antes de dar como válido el objeto registrado en sentido estructural, así como el Gestor de Información del objeto validará si la información referenciada es coherente y válida con la definición y características del objeto.

Una vez validado y comprobado que el objeto es correcto, sería finalmente introducido en el repositorio y puesto a disposición de los usuarios de la plataforma. Esta tarea se llevará a cabo cuando los gestores hayan validado tanto la estructura como los contenidos referenciados del objeto y será realizado por alguno de los perfiles Gestor.

Gestor de Validación de Estructura de Objetos

En nuestro sistema podemos encontrar dos tipos de gestores y uno de ellos será el de validación o comprobación de la estructura de los objetos.

Debido a que no existe un nivel predefinido mínimo de conocimiento para los registradores de objetos virtuales para poder abarcar a cualquier tipo de usuario que vaya a ser registrador de información, sino simplemente que el registrador rellenará los campos como el crea conveniente según las posibles directrices que se le indiquen y sus puntuales conocimientos, tendrá que haber un encargado de que lo que se registra tiene una estructura válida de acuerdo con los estándares utilizados para la construcción de los objetos virtuales (en nuestro caso IMS).

Es por ello que existe este tipo de gestor, que controlará los objetos que se registren en la plataforma para detectar posibles errores y demás gestiones. A continuación, las principales funciones que tendría este usuario:

- Consultar y validar los objetos virtuales registrados en la Plataforma y que estén pendientes de validación.
- Realizar modificaciones inmediatas en los objetos en caso de errores subsanables por parte del gestor que pueda intuirse que son fallos simples cometidos por el registrador.
- Introducir algún tipo de nota relacionada con el objeto que se está validando, indicando al propietario que haya registrado el objeto que debe realizar alguna modificación sobre el mismo para que sea estructuralmente correcto.
- Eliminar objetos que no sean adecuados para ser registrados en la plataforma debido a problemas estructurales, etc.
- Confirmar la validez de un objeto que cumpla con los estándares estructurales y que pueda publicarse u ofrecerse desde el registro de la Plataforma en el caso avanzado de que proporcionásemos los objetos registrados a través de algún tipo de protocolo.
- Publicar finalmente el objeto en el registro si ha sido validado por ambos gestores y a partir de ese momento podrá ser utilizado por los usuarios de la plataforma. Esta

publicación puede ser realizada por cualquiera de los dos gestores pero sólo si se comprueba que está validada la estructura y el contenido.

Hemos decidido no limitar el tipo de validación de estructura ni limitarlo según unos privilegios o similar, sino que se podrán registrar objetos públicos y/o privados, y es ahí donde entra el tema de la accesibilidad de los usuarios a los objetos.

Gestor de Contenido de Objetos

Comprueba que los objetos registrados en la plataforma presenten una información correcta o que las referencias que se recogen en los objetos, se refieran a información válida y que esté relacionada con el objeto registrado por el registrador. Para ello debería ser especialista en los contenidos de cada paquete, conocer un poco lo que el registrador está referenciado en el objeto para validar dicha información y comprobar si el objeto es correcto. Sus funciones son:

- Validar que el objeto registrado no es vacío o no referencia ninguna información no válida.
- Comprobar que la información referenciada tiene sentido o se corresponde con la temática o el estilo del objeto registrado.
- Eliminar objetos que no sean correctos o que estén ya registrados y supongan la presencia de objetos repetidos en el registro.
- Introducir avisos relacionados con el objeto para que el usuario registrador realice las modificaciones que sugiera el gestor para la obtención de un objeto virtual óptimo.
- Publicar finalmente el objeto en el registro si ha sido validado por ambos gestores y a partir de ese momento podrá ser utilizado por los usuarios de la plataforma. Esta publicación puede ser realizada por cualquiera de los dos gestores pero sólo si se comprueba que está validada la estructura y el contenido (lo hemos comentado anteriormente el el otro tipo de Gestor).

Como se presupone, los posibles errores o deficiencias que presente el objeto podrían considerarse opiniones subjetivas del gestor, y en cierto modo lo serían, pero se considera que el gestor que vaya a validar cada objeto lo hace porque tiene un conocimiento amplio sobre la temática y la información del objeto y se confía en que no va a ser demasiado estricto con los objetos que tenga que validar.

Hay que tener en cuenta que las indicaciones de modificación que se apliquen al objeto en caso de ser necesarias, estarán relacionadas con el tipo de información referenciada y no tendrá nada que ver con la estructura final del objeto ya que este tipo de gestor no tiene que conocer la estructura necesaria que deben presentar los objetos ni los estándares IMS en los que se basa.

El posible usuario final de los contenidos

Comentaremos el último tipo de usuario que podría acceder a la Plataforma si se diera la opción de que los objetos registrados en la misma pudieran ser ofrecidos además de registrarse y gestionarse.

Esto no entra dentro del ámbito de nuestro proyecto pero nos ha parecido importante comentar el tipo de usuario por si en un futuro se fueran a ampliar las funciones de la nuestra plataforma, y podría servir como punto de partida para posibles usuarios que trataran con la plataforma además de los arriba comentados. Por ello comentamos aquí un tipo de usuario que interactuara simplemente con la plataforma y no tuviera perfil ni de gestor ni de registrador de objetos, sino alguna otra función.

Este tipo de usuario ya hemos comentado que no entra dentro del ámbito de nuestro proyecto, pero comentamos las posibilidades que tiene para interactuar con el sistema que se reducirían básicamente a estas funcionalidades:

- Conectarse a la Plataforma y consultar los diferentes tipos de objetos virtuales registrados y que están disponibles para el usuario teniendo en cuenta su nivel de privilegios y el nivel de privacidad de los objetos.

- Esta consulta se realizaría de forma directa si se conoce el nombre del objeto, el tipo, o un índice por lo que se identifiquen los objetos, o bien a través de una búsqueda de los objetos virtuales registrados.

- Una vez localizado el objeto, el usuario pasaría a descargarlo en su equipo si lo deseara a través de los protocolos de entrega adecuados.

Cabe comentar que la consulta de los objetos por parte del usuario no tiene por qué ser como la de los perfiles que sí definimos en nuestra plataforma actual.

La consulta podría basarse en consultar únicamente en el nombre y los elementos principales del objeto, o también consultar la estructura del mismo, la distribución, etc. Se deja como idea para posibles ampliaciones.

Se podría asociar un nivel de privilegios determinado a cada usuario establecido como un número del 1 al 9, que haga que cuanto más alto sea el número mayor tipo de privilegios tiene el usuario y mayor número de objetos puede consultar u objetos con mayor privacidad.

Este nivel de privilegio para el usuario, puede determinarse según la experiencia del mismo con la plataforma, la antigüedad que lleva registrado en la ella, y otros factores que se tienen en cuenta a la hora de dar de alta al usuario pero que en todo caso del lugar donde se integre nuestra plataforma.

También puede darse el caso en que existan objetos que tengan partes privadas a los usuarios finales del objeto. En ese caso, no importaría el nivel de privilegios del usuario ya que no podría acceder a ese contenido del objeto aunque sí a los demás, si no son privados.

Pueden también existir objetos privados a los usuarios y en ese caso ni siquiera podrían acceder a sus contenidos ni descargarlos, pero sí podrían conocer de su existencia para evitar que hubiera objetos repetidos de tipo público y privado. Así nos evitaríamos saturar la plataforma.

Este tipo de perfil de usuario del sistema lo comentamos aquí pero no vamos a desarrollar el diseño de su funcionalidad en el sistema ya que no nos incumbe en el desarrollo final de la Plataforma de Registro y Gestión, pero podría servir como indicativo para una futura ampliación de la Plataforma, teniendo en cuenta sin duda, que podrían realizarse modificaciones en la misma y modificar las funciones del usuario comentado o eliminar perfiles y crear perfiles nuevos.

Todo lo último que se ha comentado no se tratará más a partir de ahora, y nuestro trabajo se basará en el diseño de la plataforma con los perfiles de usuario definidos para ella, con los diferentes tipos de objetos que va a ser posible registrar y demás funcionalidades necesarias.

2.1.3.3 Módulo de Autenticación de Usuarios

Nos referimos en primer lugar a un módulo menor de nuestra arquitectura, pero también crucial para el desarrollo y la interacción de la misma con el usuario final de la Plataforma como es este módulo de autenticación de usuarios.

Obviamente este módulo se encarga, como indica su nombre, de autenticar a los usuarios que accederán a la plataforma y de este modo podremos proporcionarles el interfaz que se ajusta al perfil que posee cada uno de ellos.

En nuestro desarrollo actual, tenemos tres perfiles principales y que dependiendo de cada uno de ellos, interactuarán con la plataforma a través de un interfaz u otro según el perfil.

Para hacer el desarrollo lo más sencillo y unificado posible, los interfaces correspondientes a los perfiles de tipo Gestor serán prácticamente iguales, salvando algunas características propias que poseen cada uno de ellos y el tipo de validación que realizan, como son la validación de estructura y la validación de contenidos. Cada uno de los perfiles gestor recibirá en la interfaz las opciones correspondientes a su tipo de validación.

Este módulo es bastante sencillo y se comportará de la siguiente forma:

- El usuario de la Plataforma accede a la misma a través de la página principal de la aplicación y lanza la aplicación para empezar a trabajar con ella.

- El Gestor de pantallas de la aplicación recogerá la petición y lanzará el elemento encargado de autenticación de usuarios, donde recogerá el login y password del usuario.

- Con estos datos, verificará si el usuario se encuentra en la Base de Datos asociada al módulo y qué tipo de perfil posee para mostrarle el interfaz gráfico correspondiente, según sea Gestor o Registrador.

Este módulo se compondrá de los elementos que proporcionen la lógica de negocio necesaria para todo lo que conlleva la conexión con una BD sencilla (llamada *Base de Datos Auxiliar*), y los métodos de consulta del perfil del usuario que accede a la Plataforma para proporcionar el interfaz adecuado para comprobrar la identidad y el perfil de los usuarios del sistema.

No se contempla el desarrollo de una interfaz que permita la inserción, actualización y borrado de usuarios sobre la Base de Datos, sino que el usuario Administrador de la Plataforma realizará estas acciones directamente en la Base de Datos a través del correspondiente programa gestor.

Las acciones de acceso a Base de Datos para realizar la Gestión de usuarios la llevaría a cabo el administrador del Sistema ya que tendría que tener conocimiento de los nuevos usuarios que vayan a poder utilizar la Plataforma y los creará según el perfil que posea el nuevo usuario, ya que no puede darse la opción de registrar un nuevo usuario como Gestor sin saber los conocimientos que posee tanto del estándar IMS si es un gestor de estructura, como del tema registrado si es un gestor de contenidos.

No podemos dar la opción de que cualquiera que acceda a nuestra Plataforma y se registre como usuario de tipo Registrador, debido a que podrían registrarse en la Plataforma usuarios sin conocimiento de lo que puede ser un objeto virtual de información e-learning y registrara objetos erróneos que saturaran el sistema.

Es por ello por lo que se limitan estas tareas y todo lo que conlleva la Gestión de usuarios al administrador de la Plataforma que tendría las funciones que definimos en el apartado anterior.

2.1.3.4 Módulo de Registro de la Información

Este es uno de los dos principales módulos que integran la Plataforma y por ello una de las partes más importantes de la arquitectura.

Este módulo se encargará de la lógica de negocio correspondiente con la parte de Registro de los objetos virtuales en el Repositorio. Básicamente contendrá los elementos necesarios encargados de implementar la lógica correspondiente a las actividades que realice el usuario Registrador.

El Registrador, una vez se haya autenticado en la Plataforma, podrá interactuar con la misma a través de un interfaz gráfico donde se muestren las opciones disponibles para este perfil, que serán diferentes a las del resto de perfiles. El módulo de Registro recogerá las acciones o peticiones lanzadas por el usuario y a través de los elementos de la lógica de negocio, actuará en consecuencia según qué opción haya seleccionado.

Por tanto, este módulo se encargará de recoger las peticiones correspondientes a:

- **Creación** de un Objeto Virtual.
- **Modificación** de alguno de los objetos realizados por el Registrador.
- **Eliminación** de alguno de los objetos propios.
- **Publicación del Objeto Virtual** definido y completado para su validación y posterior registro: el objeto registrado se recoge en alguna tabla de la BD Auxiliar, con los campos correspondientes a validación de estructura y contenido como “pendiente”. La publicación final en el repositorio se llevará a cabo por alguno de los registradores.
- **Consulta de notas asociadas** a objetos virtuales propios que presenten algún error: se muestran las notas asociadas a algunos de los objetos del Registrador consultando la tabla correspondiente de la B.D.
- **Consulta de diferentes objetos virtuales registrados**: propios o ajenos según privacidad.

Debemos tener en cuenta que nuestro objeto virtual es básicamente un archivo en XML que posee diferentes campos o etiquetas definidas y completadas por el registrador y que referencia determinados contenidos de e-learning como hemos visto en el apartado de definición de objetos anteriormente.

Todos estos elementos del objeto deberán ser validados y comprobados por los gestores de la Plataforma que indicarán algún tipo de error en el objeto o pasarán a su publicación o registro final en el Repositorio de objetos virtuales. Lo comentaremos en el siguiente módulo.

Si el usuario modifica alguno de los objetos que haya registrado en la Plataforma, debe volver a seleccionar la opción de publicar el objeto en el Registro ya que éste se pondrá automáticamente como no validado, para que siempre tengamos en el Repositorio final de objetos, únicamente Objetos Virtuales válidos, completos y correctos.

2.1.3.5 Módulo de Gestión de la Información

El otro módulo principal de la Plataforma es el correspondiente a la Gestión de la Información que se registra en la Plataforma en forma de objetos virtuales registrados por el usuario Registrador.

Este módulo va a encargarse de la gestión de estos objetos virtuales a través de la validación de una serie de características de cada uno de los objetos registrados resumidas en características estructurales y de contenido.

Esta validación la llevan a cabo los dos perfiles de tipo Gestor de la plataforma y cada uno valida una de las carácterísticas comentadas del objeto. Es necesario que las dos estén correctas para dar paso a la publicación o registro final del objeto en el Repositorio.

El registro final o publicación del objeto en la Plataforma la llevará a cabo cualquiera de los dos tipos de gestor, una vez se hayan validado las dos partes. Esto es imprescindible y no se dará la opción de publicar objetos sin validar en el repositorio, las dos comprobaciones deben ser correctas.

Es indiferente el órden de validación que se lleve por cada objeto. Se supone que el registrador selecciona la opción de publicar el objeto que ha creado y a partir de entonces, se realizarán las validaciones en este módulo sin un órden predeterminado sino según qué gestor valide qué parte del objeto.

Las opciones que contiene el menú del módulo de Gestión, serán la de consulta de **Objetos registrados pendientes de validar** y la de **Objetos validados pendientes de Publicar en el Registro**.

En ambas, nos mostrarán una lista con los objetos que se encuentren en dichas situaciones y que presentan un registro en la BD Auxiliar que es de donde sacamos esta información. Este registro permanecerá en la BD hasta que finalmente se Publique en la Plataforma o si se eliminara el objeto antes de publicarse.

En la opción de los objetos pendientes de validar, el gestor podrá seleccionar cualquiera de ellos y según sea un gestor de un tipo u otro, podrá consultar su estructura o sus contenidos. Esas pantallas de consulta serán muy similares para los dos tipos, y tendremos estas opciones al final de la misma:

- **Validar Objeto:** El gestor ha comprobado que el objeto es válido con respecto a la estructura o contenido. Se actualiza este hecho en la tabla de la Base de Datos Auxiliar que recoge los objetos pendientes de validar.

- **Creación de Nota de Aviso:** El gestor detecta algún tipo de error en el objeto y crea una nota de aviso indicando el error que tiene que subsanar el registrador para tener un objeto correcto. Se registrará en la BD un nuevo registro en la tabla de Notas Pendientes con el indicador del objeto erróneo y la descripción del error.

En la opción de objetos ya validados pero no publicados, se podrá consultar qué objetos se encuentran en esta situación y seleccionar uno o varios de éstos y seleccionar la opción de publicación:

- **Publicación Final del Objeto:** El Objeto Virtual creado ha sido validado y ya estaría disponible para los usuarios de la Plataforma. Esta opción sólo se presentará activa si se han validado las dos características evaluables del Objeto.

Así pues, terminamos indicando que en módulo de Gestión contendrá la lógica de negocio y los componentes necesarios para dar soporte a las opciones comentadas que pueden realizar los Gestores en la Plataforma.

Estos componentes se encargarán de las actualizaciones, consultas y nuevos registros en la Base de Datos Auxiliar del sistema, así como las actualizaciones de validez de los Objetos del Repositorio con respecto a su estructura, contenido y si ha sido finalmente puesto a disposición de los usuarios que utilicen el repositorio.

2.1.3.6 Base de Datos Auxiliar de la Plataforma

Esta Base de Datos será una BD bastante sencilla, y como su nombre indica, se podría considerar auxiliar ya que nos va a permitir gestionar en ella los usuarios que tengan acceso a la Plataforma, así como también las notas de aviso de los objetos erróneos de los usuarios, los objetos pendientes de validación y publicación en la Base de Datos, y todos los datos auxiliares o temporales necesarios para el correcto fun cionamiento de la Plataforma.

No queremos profundizar demasiado en esta cuestión porque es un elemento conocido por desarrolladores y diseñadores de arquitecturas, y simplemente se resume en una base de datos que contiene distintas tablas para realizar las siguientes acciones:

- **Gestión de usuarios:** inserción, modificación de alguno de los atributos, eliminación de usuarios, consulta de perfiles, etc.

- **Gestión de notas de aviso:** creación de notas, borrado, consulta, mantenimiento.

- **Gestión de Objetos Virtuales:** creación de registro indicativo de que está pendiente el objeto, modificación de alguna de las columnas de indiquen que se haya llevado a cabo un tipo de validación, borrado, consulta de validaciones, etc. También la gestión necesaria para consultar objetos validados, publicados o no y acciones similares.

- **Elementos auxiliares y temporales:** tablas auxiliares que recojan datos temporales relacionados con los objetos y/o con la navegación de la Plataforma, etc.

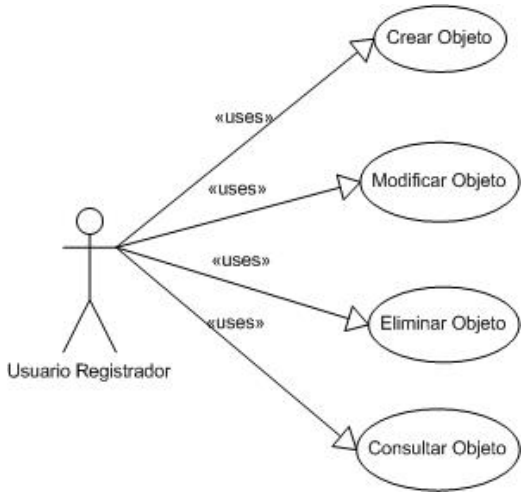
Por considerar que la Base de Datos debería ser lo más sencilla posible, se considera que podría implementarse con un gestor de bases de datos lo más sencillo y completo posible, y si puede ser, de libre distribución.

2.1.4. Análisis y Diseño de la Arquitectura PRG-eLearn

La arquitectura PRG-eLearn organiza sus diferentes funcionalides en módulos diferentes con un propósito de claridad para la mejor comprensión de su lógica de funcionamiento. Esta organización separa los componentes de la aplicación que trabajan con el registro y la gestión, respectivamente, de los contenidos.

2.1.4.1. Análisis de Requisitos de los Módulos de Registro y Gestión de PRG-eLearn

Vamos a numerar los principales requisitos necesarios para cada módulo de la Plataforma. Se detallan los requisitos que presentan los módulos, describiendo los Casos de Uso asociados a los mismos.



MÓDULO DE REGISTRO

Creación de Objetos

- Los objetos que vayan a ser creados deben corresponderse con alguna temática e-learning. No deberían registrarse objetos de cualquier tipo que no contengan algún tipo de recurso de sentido académico o e-learning.

- El usuario que crea los objetos tiene que tener permisos de Registrador puesto que si no no podría acceder al menú de creación del Objeto.

- A cada nuevo objeto creado, se le asignará un identificador automáticamente para poder crear el registro asociado en la BBDD Auxiliar.

- En la BBDD, se registrará en la tabla de Registro de Objetos los siguientes datos:

    - El identificador del objeto.

    - El identificador del propietario o autor del objeto.

    - Las validaciones realizadas sobre el objeto, que se iniciarán como pendientes. Habrá dos columnas para los dos tipos de validación.

- El registro correspondiente a este objeto permanecerá en la tabla de la BBDD para facilitar el mantenimiento de los objetos registrados en el Repositorio. Este registro sufrirá cambios por actualizaciones en sus atributos si se produce alguna modificación o similar en el objeto.

- El objeto creado será finalmente un archivo XML (*manifest*) con diferentes etiquetas correspondientes a los datos introducidos por el registrador, con el orden estructural que haya determinado el registrador y con los contenidos elegidos y asociados como recursos del objeto.

- El objeto no lo registra en el Repositorio el registrador, sino que crea el registro en la tabla de objetos registrados con los campos de validación como pendientes y se mostrará a los gestores en las listas de objetos pendientes de validar.

- El registro del objeto en el Repositorio lo realizará alguno de los gestores cuando se hayan realizado sobre él las dos validaciones contempladas y seleccione la opción de publicar.

Modificación de Objetos Registrados

- El registrador puede modificar cualquiera de los objetos que haya creado él mismo. Tanto los objetos que estén pendientes de validar y aparezcan en la BBDD Auxiliar como los objetos que ya se encuentren validados correctamente en el Repositorio y se quieran actualizar.

- Según seleccione esta opción, se mostrará la lista de objetos que ha creado y la opción para seleccionar el que se quiera modificar y el botón para empezar la modificación que crea necesaria.

- Esta lista se obtiene realizando una búsqueda en la tabla de Objetos Registrados consultando los identificadores de objeto asociados al identificador de usuario. Ese identificador de usuario se asocia cuando se realiza el registro en la BBDD del usuario Registrador.

- Tanto los objetos que estén pendientes de validación como los que estén registrados, pasarán a estado “Pendiente de Validar” y se modificará el registro correspondiente al código del objeto en la BBDD Auxiliar si el objeto ya estuviera validado por los gestores en la BBDD.

- Dicho estado simplemente es modificar los valores de las columnas de Validaciones Pendientes y ponerlas como no validadas. Se comportaría como un objeto recién creado pendiente de validación por parte de los Gestores.

Eliminación de Objetos Registrados

- El usuario obtendrá la lista de objetos creados y/o registrados por él mismo pudiendo seleccionar aquellos que quiera eliminar.

- Si se selecciona la opción de Eliminar el Objeto, se realizan algunas acciones en el Repositorio y en la Base de Datos:

    - Se elimina el objeto del Repositorio si ya hubiera sido validado y publicado.

    - Se elimina el registro de la tabla Objetos Registrados de la BBDD correspondiente al objeto registrado.

    - Se eliminan las notas asociadas al objeto si tuviera alguna.

Consulta de Objetos Registrados

- El usuario obtendrá una lista de los Objetos que haya creado y registrado en la Plataforma. Se obtienen a través de una consulta a la tabla correspondiente de la BBDD Auxiliar.

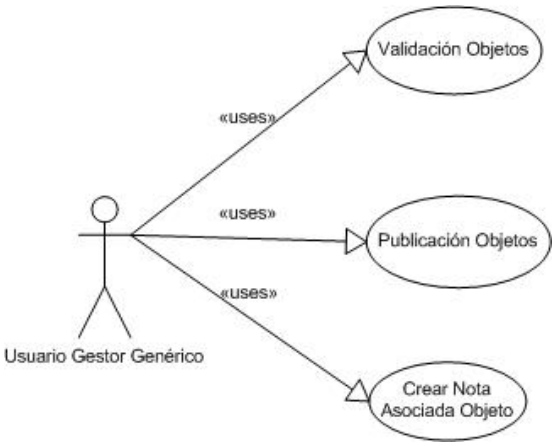
- Selecciona aquél de la lista que quiera consultar y se mostrará el objeto. La lista de objetos será más o menos la misma que la que se muestra en la Eliminación y la Modificación y podría obtenerse de igual forma, buscando en BBDD por el identificador de objeto y usuario.

- En la pantalla de consulta del objeto, además de todas las características de éste, se mostrarán todas las notas asociadas si es que tiene alguna. Esto se realiza con una simple consulta a la tabla de Notas de la BBDD buscando con el identificador del objeto.

- Puesto que ha entrado en la pantalla de consulta, se actualiza la BBDD para indicar en la tabla y columna correspondientes, que las notas han sido ya consultadas, pero no se borrarán hasta que el objeto se publique en el Repositorio de Objetos o sea eliminado.

- Una vez seleccionado y mostrado el objeto correspondiente, podremos realizar las acciones propias de Eliminación y de Modificación a través de los botones correspondientes, además de la de Consulta.

- Esta opción sería global a las otras dos acciones comentadas arriba: *Eliminación* y *Modificación*.



MÓDULO DE GESTIÓN

Al igual que para el módulo de Registro, el usuario que realizará las actividades propias del Gestor tiene que haber sido dado de alta con los permisos correspondientes a uno de los dos tipos de Gestor para poder acceder al menú de opciones del Gestor de Validación de Estructura o al de Validación de Contenidos.

Validación de Objetos Registrados pendientes de Validar

- Seleccionando esta opción, se mostrará al Gestor del tipo que sea, una lista con los objetos creados y registrados pendientes de validar de la parte correspondiente al tipo de Gestor.

- Esta lista se obtendrá buscando los objetos pendientes de validar de alguna de las dos validaciones, dando la opción de consultar el objeto si la validación que falta se corresponde con el tipo de Gestor.
    - De esta forma los gestores pueden observar qué objetos faltan por validar de alguna de las dos partes o de las dos y no tenemos que diferenciar la búsqueda de objetos en la BBDD, sino sólo permitir su validación si es de su tipo.
    - Si la única validación que falta es la suya, facilitaría el registro de ese objeto en el Repositorio lo antes posible si se realizan primero las validaciones de los objetos que están semivalidados.
- Una vez seleccionado el objeto a validar, pasaría a mostrarse su Contenido o Estructura para que el gestor pudiera consultarlo según su tipo.
- El gestor comprueba si el objeto es válido o no lo es. Dispondrá de las siguientes opciones:
    - **Validar el objeto como correcto**: Se actualiza el registro correspondiente en la BBDD Auxiliar para indicar que se ha validado el objeto.
    - **Crear Nota de Aviso**: Se pasará a otra pantalla en la que podremos matizar algún tipo de descripción de algún posible error que contenga el objeto validado por el gestor.
        - Se presupone que la creación de la nota de aviso se realiza porque el objeto presenta algún error de tipo estructural o de contenido.
        - Se crea un registro en la tabla correspondiente a las notas en la BBDD que asocie el objeto con la nota y la descripción del error que se usará para avisar al Registrador del objeto para que lo modifique.
        - Se pueden crear varias notas diferentes para un mismo objeto por el mismo gestor para dar la opción de indicar en distintas notas, varios errores reconocidos en el objeto.

Publicación de Objetos Validados en el Repositorio

- En este caso se mostrará una lista de todos los objetos que presentan sus validaciones como realizadas pero no han sido aún registrados en el Repositorio.

- En este momento, el gestor seleccionará uno o varios y pulsando el botón correspondiente al Registro se procederá a la Publicación de los objetos en el Repositorio.

- En este momento se realizan diferentes acciones en varias de las tablas de la BBDD Auxiliar como son las siguientes:

    - Se actualiza el registro correspondiente al objeto para indicar en la columna correspondiente que ya se haya en el Registro. De esta forma no se mostrará en la lista de objetos que pueden pasar a publicarse.

    - Se eliminan las notas registradas asociadas al identificador del objeto puesto que si está validado no presenta errores y no tiene sentido conservar las notas que podrían mostrarse y llevar a equívoco a la hora de consultar el objeto.

- Volvemos a repetir que esta acción de publicar la puede realizar cualquiera de los dos tipos de Gestor ya que esta opción es común y se comporta de igual forma en los dos lados.

MÓDULO DE AUTENTICACIÓN

- Este módulo actúa en el inicio de la interacción con la Plataforma ya que se encarga de controlar los usuarios que acceden a la Plataforma y comprueba si están dados de alta en ella y a qué perfil pertenecen para mostrar un interfaz u otro.

- El usuario accede a la pantalla inicial de la Aplicación y una vez que entra en ella, se lanza la pantalla de validación de usuarios.

- El módulo de Autenticación comprueba que es un usuario válido, y según su perfil, llama a un interfaz u otro. Cada interfaz ofrece las opciones correspondientes a los módulos de registro para los registradores y gestión para los gestores.

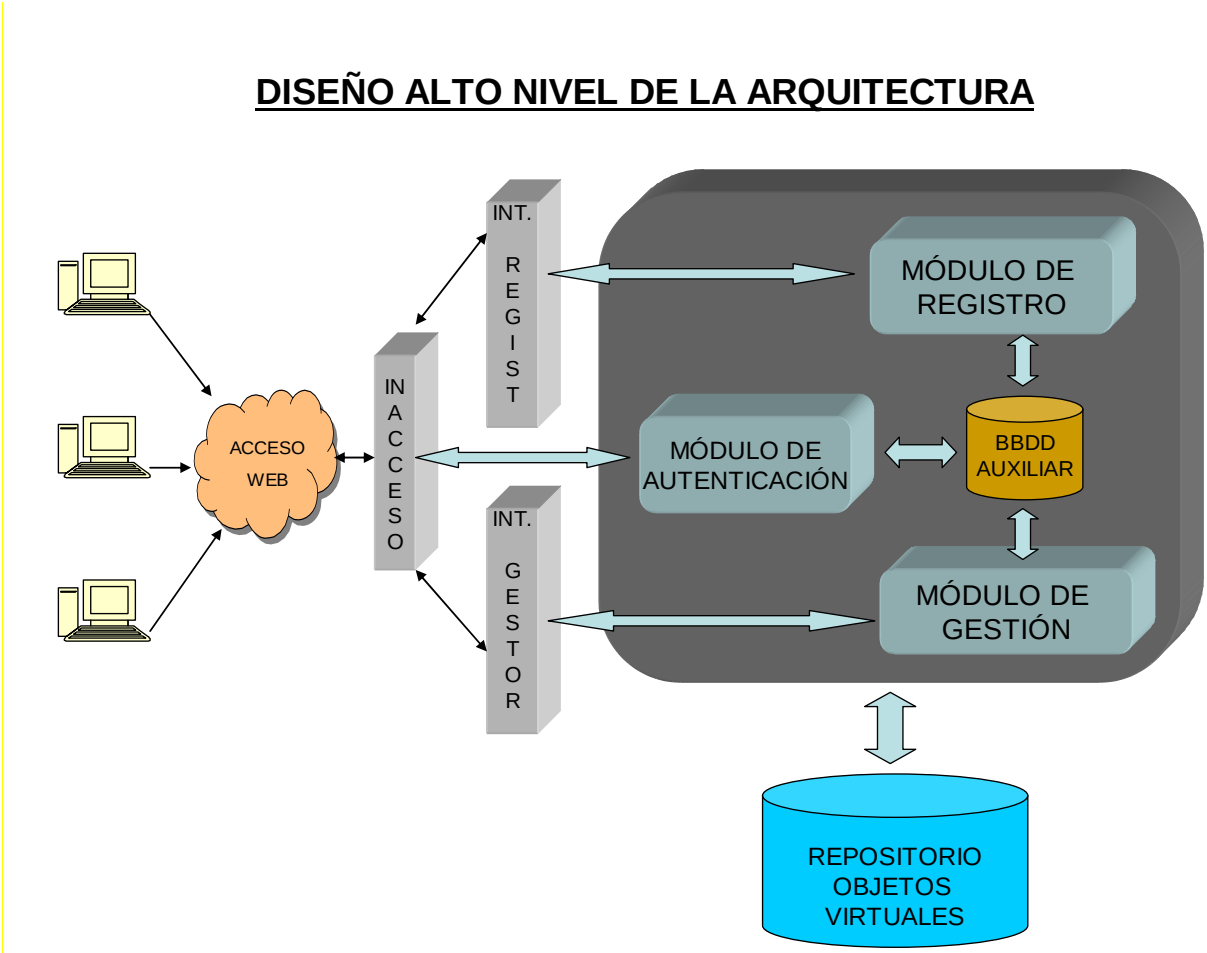
- Para su correcto funcionamiento, guardará algún tipo de variable u objeto en sesión que nos indique el usuario que interactúa y qué perfil dispone, consultando la correspondiente tabla de Usuarios de la BBDD, a partir del login y password introducidos. Se obtiene el perfil del usuario si existe, y mostrará algún tipo de mensaje de error si no está prensente en la BBDD, o falta por introducir alguno de los campos por los que consulta.

- Los usuarios tendrán que haber sido insertados en la Base de Datos mediante el Administrador de la Plataforma con su “login”, “password” y “perfil” correspondiente.

- Las inserciones, modificaciones y eliminaciones de la Base de Datos las realiza el Administrador a través del propio Gestor de la BBDD. No implementamos un interfaz correspondiente a la Gestión de Usuarios en este Diseño de la Plataforma, aunque podría ser una de las principales características para una futura evolución de la Aplicación.

2.1.4.2. Diseño de Alto Nivel de la Arquitectura PRG-eLearn

En primer lugar mostramos un Diagrama de Alto Nivel de lo que va a ser nuestra Arquitectura para el desarrollo de la Plataforma de Registro y Gestión definiendo sus principales módulos y la interacción que va a ocurrir entre todos ellos. A continuación el diagrama de la Arquitectura:



Como podemos ver en el Diagrama, tenemos tres elementos principales en la Arquitectura que serán los encargados de que el Sistema funcione y proporcione las funciones que nuestra Plataforma ofrece. Son los siguientes:

- Interfaz Gráfica de interacción con el Usuario.

- Módulo de Lógica de Negocio de la Plataforma.

- Repositorio de publicación o registro de objetos virtuales.

Nuestra arquitectura pretende implementarse con herramientas que permitan el desarrollo de la Plataforma con tecnología Orientada a Objetos debido a todas las ventajas que esta tecnología ofrece. La mejor opción que hemos decidido adoptar, es el desarrollo de la Plataforma con tecnologías J2EE utilizando toda la cantidad de recursos e implementaciones que ofrece.

Es por ello que la Plataforma podemos dividirla en estos tres módulos principales ya que adoptan perfectamente la filosofía de un proyecto de *"Tres Capas"* que podemos desarrollar con la tecnología comentada. Así pues, se puede comprobar rápidamente la relación de este modelo con las capas que propone nuestra arquitectura:

- **Capa de Interfaz de Usuario -** Diferentes interfaces gráficos de usuario que ofrecerán las opciones a cada tipo de usuario de la Plataforma.

- **Capa de Lógica de Negocio -** Módulo que engloba los tres módulos de lógica de negocio de la Plataforma encargados de la Gestión y Registro de objetos de la Plataforma, y de la Autenticación de usuarios.

- **Capa de Acceso a Datos –** Repositorio de Objetos Virtuales donde se publicarán los objetos creados por los usuarios Registrador del Sistema.

Vistas ya las capas de nuestra arquitectura, pasamos a describir los principales módulos que la componen.

INTERFAZ GRÁFICA DE LA PLATAFORMA

Podemos definirla como dividida en dos interfaces, la Interfaz de Usuario Registrador y la de Usuario Gestor, que serán las dos principales interfaces que contiene el Sistema.

Para acceder a una interfaz u otra, se accede a través de una primera pantalla de acceso que valida al usuario que accede a la Plataforma y comprueba que está validado y que según su perfil accederá a uno de los interfaces.

Hablamos de dos interfaces principales ya que las acciones que realizan los usuarios Gestor son muy similares y, por tanto, el interfaz será prácticamente el mismo exceptuando alguna de las pantallas que posea o algún detalle en las mismas.

Antes de definir los dos tipos de interfaz de la aplicación, comentar que nuestra Arquitectura se va a comportar siguiendo también otra filosofía o idea, además de la de tres capas comentada anteriormente, con respecto al tratamiento de la interfaz y los módulos de la lógica de negocio que la componen.

Nos referimos a que se va a desarrollar según el tipo **Modelo – Vista – Controlador** sobre todo si nos referimos a la parte de la Interfaz y de los módulos de la Lógica de Negocio. Comentamos su comportamiento a continuación:

- ✓ La *Vista* ofrece por pantalla al usuario las opciones de iteración con la Plataforma y recoge las opciones y acciones seleccionadas por el mismo, y también devuelve y muestra los datos de respuesta según las acciones o algún tipo de error cometido, etc.

- ✓ El *Controlador* como su nombre indica, controla las acciones que se producen en la Plataforma y pasa el control al Modelo si es necesario realizar algún tipo de operaciones o similar sobre los datos de entrada para generar una salida, o a la Vista si las acciones implican algún tipo de respuesta por pantalla tras la finalización de acciones del Modelo o para indicar errores, etc.

- ✓ El *Modelo* lo podemos definir como la Lógica de Negocio de la Aplicación necesaria para procesar los datos de entrada o las acciones, y obtener la respuesta del sistema utilizando esos datos. Devolvería el control al Controlador con la respuesta si la necesitara para que éste continuara con la ejecución del sistema.

Básicamente este es el comportamiento principal de la filosofía comentada y es el que nos va a guiar en el desarrollo de nuestra Plataforma, para obtener una aplicación lo más estructurada posible para que nos facilite su mantenimiento y futura evolución a fin de optimizarla si fuera preciso.

Así pues podemos concretar que el elemento Vista y el elemento Controlador se presentan en esta parte de la Arquitectura, es decir del interfaz, mientras que el Modelo se encontrará en la parte de la Lógica de Negocio que implementan los diferentes módulos de la Plataforma.

Con esta filosofía en mente, pasamos a ver las dos interfaces que componen este módulo de la Arquitectura:

Interfaz de usuario registrador:

El interfaz de usuario registrador proporciona las diferentes acciones que puede realizar el registrador de objetos dentro de la Plataforma a través de las pantallas que la componen, y éstas son las acciones definidas y ya comentadas en el análisis de requisitos.

Este elemento de la arquitectura, al igual que el interfaz del Gestor, recogerá las peticiones de los usuarios y las enviará al módulo correspondiente que da soporte a las opciones de la interfaz, que en este caso sería el Módulo de Registro.

Observando este comportamiento, podemos decir que la plataforma se comporta según la filosofía del **Modelo – Vista – Controlador** comentada en el punto anterior, ya que el interfaz recoge las peticiones (Vista), existe un elemento que llamará a un componente u otro de la lógica de negocio implementada en los módulos (Controlador) y este componente realizará las acciones pertinentes según la entrada que haya recibido (Modelo), devolviendo una salida si fuera necesario.

Una vez el módulo Registrador haya realizado las operaciones correspondientes a la acción introducida por el usuario, la Vista mostrará los eventos relacionados a dichas operaciones para que el usuario obtenga información de lo ocurrido y pueda seguir interactuando con la Plataforma.

En resumen, esta interfaz se encarga de ofrecer las pantallas con las opciones que puede realizar el Registrador, y proporcionar las pantallas de respuesta ante las acciones realizadas en la Plataforma.

Interfaz de usuario gestor:

La interfaz del usuario Gestor se comporta prácticamente igual que la del usuario Registrador y se basa en la misma filosofía que éste. Tendremos las pantallas que ofrecen las opciones a los usuarios y que serán similares en muchos aspectos, con las principales diferencias que presenten si son de tipo Gestor Estructural o si son de tipo Gestor de Contenidos.

Esta vez será el Módulo de Gestión de Objetos Virtuales el encargado de realizar las operaciones de la lógica de negocio asociadas a este interfaz y a los datos y eventos de entrada, para realizar las acciones pertinentes y obtener los datos de salida que se produzcan o provocar los eventos asociados.

A la Vista (Interfaz) se le pasan los datos o eventos de salida y los mostrará en las pantallas correspondientes o mostrará errores producidos asociados a las acciones comentadas.

Como podemos ver, ambos interfaces se comportan de manera muy similar y se comunican con sus respectivos módulos que recogen los datos y proporcionan la lógica de negocio asociada a los mismos.

MÓDULO DE AUTENTICACIÓN DE USUARIOS

Este Módulo se comunica con el Interfaz gráfico de usuario y actuará cuando se accede a la Plataforma en un primer acceso para realizar la validación de usuario y obtener el perfil asociado al mismo.

Este módulo tendrá un componente que será el encargado de recoger los datos introducidos como *login* y *password* del usuario, e implementará unos métodos para conectar con la Base de Datos Auxiliar que posee la Plataforma y consultar si el usuario es válido y qué perfil posee dentro de la aplicación.

Después de obtener la respuesta según estos datos, realizará las acciones pertinentes como son:

- Devolver algún tipo de error si no se encuentra el usuario introducido.
- Redirigir el control de la aplicación hacia una interfaz u otra, según perfil.

Este componente encargado de las anteriores acciones, podemos implementarlo como un Servlet o algún tipo de clase que conecte con la BBDD y reciba los datos de la interfaz, si implementamos la aplicación con tecnologías J2EE.

MÓDULO DE REGISTRO DE OBJETOS VIRTUALES

Este módulo será el encargado de dar soporte a todas las acciones que se realicen a través del Interfaz Registrador que utiliza el usuario de mismo nombre. Recogerá las peticiones de entrada que realicen los usuarios y realizará las acciones asociadas a cada una de las opciones de menú que se proponen.

Se implementarán los componentes necesarios para dar soporte a todo lo comentado utilizando las tecnologías comentadas anteriormente, y desarrollando métodos de acceso a la Base de Datos donde se modificarán registros en las tablas correspondientes, así como métodos de acceso al Repositorio donde se registrarán los objetos creados y completados con sus características y demás.

Así pues, este módulo tiene interacciones con los otros elementos de la Plataforma que a continuación comentamos y definimos las acciones que se desarrollan con respecto estos otros elementos:

- **Base de Datos Auxiliar**
  - Inserciones de nuevos registros asociados a nuevos objetos creados por el usuario.
  - Consulta de objetos virtuales ya registrados para eliminación, modificación o simple consulta.
  - Consulta de notas asociadas a los objetos para comprobar errores en los mismos.
- **Interfaz de usuario Registrador**
  - Recibe las peticiones realizadas sobre la interfaz, según las opciones de menú que éste ofrece.

- Realiza las operaciones requeridas sobre los datos de entrada para generar la salida correspondiente.

- Entrega estos datos de salida obtenidos al internaz por si tiene que mostrarlos o los errores que se hayan producido.

- **Repositorio de Objetos Virtuales**

    - Realizará todos los métodos correspondientes para hacer inserciones de nuevos objetos creados en el repositorio, así como de consulta, modificación y eliminación de objetos.

MÓDULO DE GESTIÓN DE OBJETOS VIRTUALES

Este módulo, si lo vemos a alto nivel, es similar al módulo de registro y será el encargado de dar soporte a las acciones que se realicen a través del Interfaz Gestor que utilizan los dos usuarios de este tipo. Recogerá las peticiones de entrada que realicen los usuarios y realizará las acciones asociadas a cada una de las opciones de menú que se proponen, diferenciando cada uno de los dos tipos de perfil gestor que proporciona la Plataforma y que influirá en las pantallas de la Interfaz común y en los componentes de la lógica que trata este módulo.

Como hemos comentado en el módulo anterior, en este también se implementarán los componentes necesarios y para los métodos de acceso a la BBDD a fin de realizar consultas, inserciones y lo que sea necesario, así como los métodos de consulta y publicación de objetos sobre el Repositorio, utilizando las tecnologías propuestas para ello.

Las interacciones de este módulo con los demás elementos de la Plataforma y las acciones producidas se describen a continuación:

- **Base de Datos Auxiliar**

    - Consulta de objetos registrados pendientes de Validar.

    - Inserción de notas asociadas a los objetos registrados pendientes de validar debido a algún tipo de error reconocido por el gestor del tipo que sea.

    - Modificación del registro asociado al objeto para indicar que ha sido validado de la parte correspondiente al gestor que interactúa, y/o para indicar que ya ha sido finalmente publicado en el Repositorio.

- **Interfaz de usuario Gestor**

    - Recibe las peticiones realizadas sobre la interfaz, según las opciones de menú que éste ofrece. Las opciones serán las mismas para ambos tipos de usuario Gestor.

    - Realiza las operaciones requeridas sobre los datos de entrada para generar la salida correspondiente.

    - Entrega estos datos de salida obtenidos al internaz por si tiene que mostrarlos o los errores que se hayan producido.

- **Repositorio de Objetos Virtuales**

    - Realizará todos los métodos correspondientes para realizar las validaciones y/o publicaciones sobre los objetos registrados en el Repositorio, que no es más que acceder al objeto y modificar alguno de sus Metadatos.

BASE DE DATOS AUXILIAR

Este elemento es bastante sencillo pero no por ello menos importante que los demás elementos de la Arquitectura ya que todos los módulos encargados de la Lógica de negocio de la Plataforma acceden a él para realizar diversas acciones y podríamos considerarlo el nexo de unión de todos ellos.

Esta Base de Datos se compone de varias tablas encargadas de recoger los datos con los que van a interactuar los módulos y que las principales podrían ser las siguientes:

- Una tabla para gestionar los objetos creados y registrados en la Plataforma con los atributos correspondientes de si está validado, publicado, etc.

- Tabla de los usuarios registrados en la plataforma con sus perfiles asociados.

- Tabla para gestionar las notas asociadas a los objetos que presenten algún tipo de error.

- Diferentes tablas que almacenen elementos auxiliares para el correcto funcionamiento de la Plataforma. Algunas de esas tablas pueden ser estáticas con los valores asociados a etiquetas del estándar IMS, y otras dinámicas para el almacenamiento temporal de datos asociados a los objetos que sea necesario consultar en momentos puntuales, etc.

La Base de Datos se quiere implementar con alguno de los gestores de bases de datos más conocidos, sencillos y que sean de distribución gratuita y se puedan obtener a través de la Web, un gestor de

BBDD que cumpla todos estos requisitos y se integre bien con las tecnologías con las que se va a trabajar para desarrollar la Plataforma.

REPOSITORIO DE OBJETOS VIRTUALES

El Repositorio de Objetos Virtuales es uno de los elementos principales y vitales de nuestra Plataforma ya que en él se lleva a cabo toda la acción para la que implementamos esta aplicación como son el Registro y la Gestión de Objetos.

Este elemento va a consistir básicamente en una especie de servidor o contenedor de objetos virtuales que no son más que ficheros en formato XML (denominados manifest) ajustándose a unas pautas y a unos estándares para que puedan registrarse en él:

- Deben ajustarse al estándar IMS – Content Packaging que establece unos requisitos sobre las etiquetas del fichero así como de la estructura.

- Los contenidos de los ficheros deben referirse a informaciónde e-learning ya que la Plataforma se basa en el registro de información de este tipo.

Una de las posibles implementaciones del Repositorio para el desarrollo de la plataforma, podría ser un repositorio UDDI de servicios web por la similitud que tendría la funcionalidad de la aplicación y este tipo de arquitectura. En ese caso, los objetos virtuales se tendrían que codificar en el lenguaje WSDL necesario para registrarse en este tipo de repositorio, señalando los datos y funciones del objeto virtual. Esta sería la implementación óptima para ofrecer una completa plataforma de objetos virtuales.

Otra posible implementación mucho más sencilla pero menos completa del repositorio, sería la de implementar el repositorio de alguna manera en una base de datos, insertando los objetos virtuales en la misma. Posiblemente esta opción sea mejor para el desarrollo de un prototipo de manera rápida y funcional.

Realmente todas las restricciones y pautas que tienen que tener los objetos virtuales registrados no van a ser definidas por los usuarios que registren los objetos, sino que se realizarán estas conversiones de forma transparente al usuario registrador, pero sí que será necesario que se cumplan los estándares de IMS ya que los gestores controlan que se ajusten a estos parámetros.

Los módulos de Gestion y Registro a través de los componentes implementados, desarrollan la lógica de negocio necesaria para realizar todas las actividades que vayan a realizarse sobre el Repositorio como pueden ser:

- El registro de los objetos virtuales iniciales que registra el usuario correspondiente.

- La obtención del objeto virtual para realizar consultas sobre el mismo que nos puede llevar, según sea un tipo de usuario u otro, a su eliminación o modificación o simple consulta.

- La publicación de un objeto por parte de cualquiera de los gestores, una vez se haya comprobado que las dos validaciones son correctas.

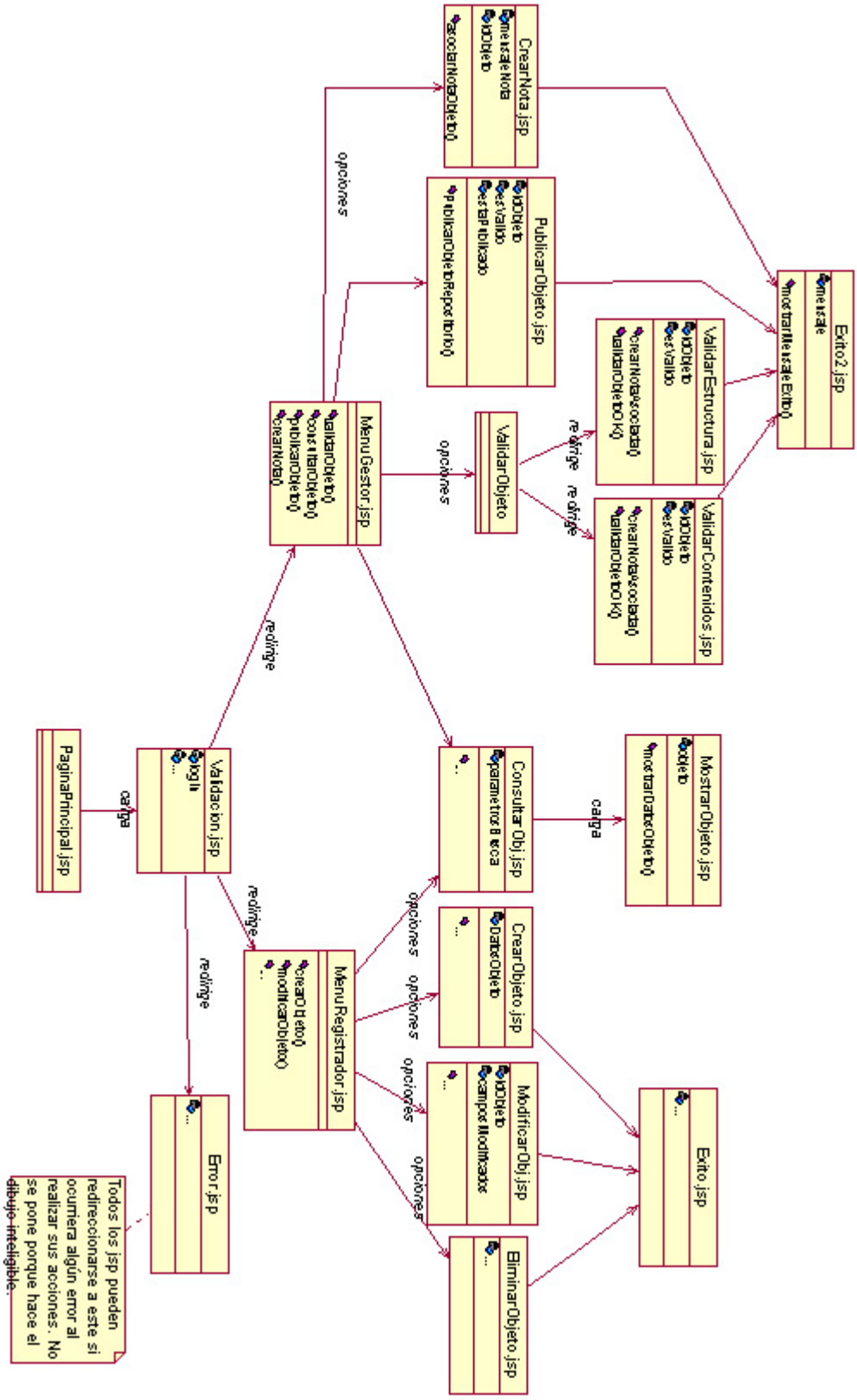
Ya que todos los elementos del Sistema van a desarrollarse utilizando tecnologías J2EE podemos intuir que la implementación del Repositorio va a desarrollarse con alguna herramienta que trabaje bien con estas tecnologías.

2.1.4.3. Diseño Detallado de la Arquitectura PRG-eLearn

Una arquitectura J2EE bien organizada mantiene aisladas capas con funcionalidades de distinto tipo. La arquitectura de struts preserva la esencia del modelo-vista-controlador. Con esta filosofía, organizaremos la aplicación de forma que se vean diferenciadas la generación de pantallas en el navegador, la lógica de negocio y la persistencia.

Si en el diagrama de alto nivel se presentaban anteriormente los módulos de forma general, con sus formas más importantes de comunicación, en el diagrama de clases siguiente partiremos de los archivos jsp, para los que debemos recalcar que siempre llevarán asociados sus archivos java tanto de tipo action como de tipo action form, que son los que realizan las funciones de negocio, mientras que los jsp sólo deben presentar datos y dar opciones al usuario.

A continuación del diagrama, veremos qué lugar tiene cada clase en el esquema modular, y detallaremos su cometido.



Mapa de navegación de la arquitectura PRG-eLearn

- **Interfaz de acceso**: formada por las pantallas de inicio. Pide el identificador de usuario y contraseña y realiza la validación del usuario en la base de datos, buscando un tipo de usuario concreto para el identificador introducido y obteniendo el tipo de perfil para conducirle a un interfaz u otro.

- **Interfaz del gestor**: formado por el menú del usuario Gestor y sus posibles caminos. Nos ofrece las opciones propias que tiene el gestor, como son:

    - **Validar objeto**: la misma idea que en publicar objeto, la base de datos sufrirá la modificación del campo "validado" para cada objeto validado. Desde aquí el usuario podrá consultar el objeto antes de validarlo definitivamente.

    - **Publicar objeto**: se listará al gestor un listado con los objetos pendientes de publicar, datos tomados de la base de datos auxiliar. Se le dará la opción de hacerlo con los que desee, actualizándose el campo "publicado" en la base de datos de los objetos que se publiquen y dándose de alta los objetos en el repositorio para poder ser ofrecidos a partir de ese momento.

    - **Crear nota**: si el gestor considera que un objeto no es correcto en algún modo, asociará una nota a éste a través de la jsp correspondiente, que buscará el objeto y asociará una nota en la base de datos a ese objeto. La nota se guardará en la base de datos auxiliar y se asociará al objeto correspondiente.

    - **Consultar objeto**: la opción en donde un gestor puede estudiar los objetos para decidir su corrección y poder validarlos o publicarlos. Se le despliega la lista de objetos a los que tenga acceso y podrá seleccionar uno, del que se presentarán su contenido inmediatamente, tras tomarse del repositorio.

- **Interfaz del registrador**: ofrecerá al registrador las opciones propias de este usuario:

    - **Consultar objeto**: se le presentará el objeto que haya escogido de entre los de la lista de objetos propiamente creados por el usuario o demás objetos presentes en el repositorio y que pueda consultar, tomando los datos de la tabla adecuada en la base de datos auxiliar.

    - **Crear objeto**: Aparecerá un menú con todos las posibles etiquetas y elementos que pueden formar parte de un objeto, siempre atendiendo a las especificaciones del estándar, con sus requisitos específicos. Una vez creado, se escribirán sus datos en la base de datos auxiliar y se depositará en el repositorio.

- **Modificar objeto**: Podrá escogerse un objeto de una lista tomada de la base de datos que contendrá los objetos propios del usuario. El objeto se tomará del repositorio, y una vez finalizada la modificación, se escribirán sus datos en la base de datos auxiliar si hubieran cambiado éstos, y se almacenaría de nuevo en el repositorio el objeto modificado, con los campos validado y publicado de nuevo como pendientes.

- **Eliminar objeto**: Podrá escoger eliminar el objeto que está consultando, o de una lista de sus propios objetos creados. Sus datos procederán a ser borrados de la base de datos y será eliminado del repositorio, así como sus notas asociadas si las tuviera.

Todos estos casos de uso podrán verse sujetos a algún error por parte del usuario, momento en el que se le dará un aviso relacionado con el error cometido desde la pantalla correspondiente.

La *lógica* de negocio de las diferentes opciones o funcionalidades de la plataforma, con las que se podrá trabajar a través de la navegación, viene manejada por un conjunto de clases que se apoyan en otras clases que recogen las respectivas operaciones tanto para el tratamiento de la base de datos como del repositorio, que ofrece la persistencia de los objetos creados mediante la plataforma.

Se ha definido un diagrama de clases con las clases más relevantes que compondrían la lógica de negocio necesaria para determinar la arquitectura de la plataforma. Este diagrama está definido a muy alto nivel y no tiene detalles concretos y definidos de las clases, y muy probablemente será diferente al diagrama que definamos para el prototipo, donde se conocerán muchos mejor los detalles de las clases a implementar, con sus atributos, métodos, etc.

Por cuestiones de claridad no se incluyen en el diagrama las clases *Action* ni *ActionForm* asociadas a la filosofía del framework Struts, y que tienen un comportamiento similar en todos sus accesos:

- El jsp muestrará las opciones que pueda seleccionar el usuario de la aplicación en ese momento.

- La clase *ActionForm* obtiene los datos asociados al jsp introducidos por el usuario, como pueden ser formularios, textos, textareas, checkbox y demás elementos.

- La clase *Action* implementa la lógica de control necesaria para ofrecer los datos que se ofrecen a partir de las opciones realizadas por el usuario en la interfaz y que hará las respectivas llamadas a las clases que implementan la lógica de negocio de la plataforma: acceso a base de datos, persistencia de objetos virtuales, etc.

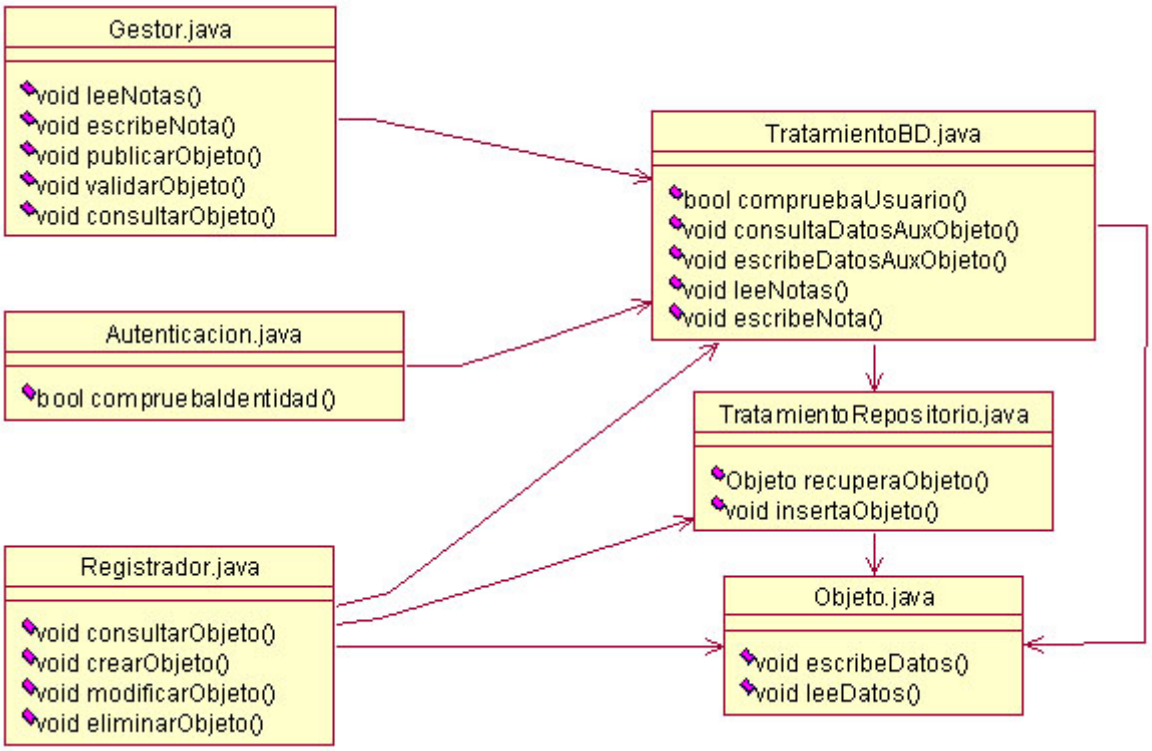


Diagrama de clases en la arquitectura PRG-eLearn

2.1.5. Diseño de la Plataforma de Simulación

2.1.5.1. Selección de Herramientas de Simulación

Vamos a comentar de forma resumida las herramientas que se van a utilizar para el desarrollo, implementación y demás de la Plataforma ya que se detallarán de forma más explícita en la parte Instalación, Configuración y Prueba de la Plataforma de Simulación.

En primer lugar, se va a trabajar con un entorno de desarrollo bastante completo y sencillo de instalar y utilizar como es netBeans 5.0, que además proporciona una fácil integración con el framework Struts ya que lo incorpora y ofrece la oportunidad de utilizarlo para respetar la filosfía tantas veces comentadas como es Modelo Vista Controlador.

Para lanzar la aplicación una vez desarrollada, utilizaremos un servidor de aplicaciones como es Tomcat 5.5.9 que viene integrado en el propio entorno de desarrollo, con las facilidades que esto conlleva, y ofrece todas las características necesarias para el correcto funcionamiento de la aplicación desarrollada.

Para el desarrollo de la Base de Datos Auxiliar de la aplicación se ha pensado en utilizar mySQL por su sencillez de manejo, por ser de libre distribución y por poseer mucha documentación asociada en la Web. Podríamos utilizar otros gestores de base de datos más completos y fiables, pero la mayoría requieren licencia, y obviamente hay que pagarla.

Para el desarrollo de un buen repositorio UDDI, se ha investigado el que ofrece Jakarta que se llama jUDDI y es de libre distribución y además hecho con Java, pero tiene bastantes inconvenientes como poca documentación asociada y pocos ejemplos de uso. Se comenta como posible implementación si realizáramos una implementación completa de la Plataforma y por tanto, es probable que no lo utilicemos para el prototipo por falta de tiempo.

Pueden existir más elementos para la simulación de la Plataforma que se comentarán más adelante en la parte del prototipo desarrollado y donde veremos los elementos de simulación más a fondo.

2.1.5.2. Diseño Técnico de la Plataforma

En este apartado nos vamos a referir a la parte hardware necesario para el correcto funcionamiento de la Plataforma y cuáles son los requisitos para su funcionamiento.

Hoy en día los equipos que poseemos todos en casa, trabajo y/o facultades son equipos bastante potentes como para encargarse de ofrecer una aplicación de forma constante a través de Web si el sistema donde se instala no está demasiado sobrecargado de aplicaciones y demás elementos que puedan residir e influir en el rendimiento de la Plataforma.

Por tanto, se recomienda la utilización de la Plataforma en un PC normal que se comporte como servidor utilizando un servidor de aplicaciones que permita el acceso a la misma por múltiples usuarios y con acceso también múltiple y simultáneo a la base de datos.

Pensamos que con un PC con las siguientes características se cumplen los requisitos para el funcionamiento de la plataforma, pensando que es posible la ampliación de sus componentes si el sistema creciera mucho y el equipo se ralentizara o lo notara. A continuación una posible configuración de un PC donde se podría instalar y ofrecer la Plataforma de Registro y Gestión implementada:

- Equipo PC con microprocesador Intel Pentium 4 ó AMD Athlon XP con 2 Ghz de velocidad o a partir de 2 Ghz.
- De 512 a1024 Mb de memoria RAM.
- Disco duro de 80 Gb de capacidad.

- Sistema operativo Windows XP ya que es con el que se va a desarrollar. Por ahora no se contempla una versión para Linux, podría ser para una futura ampliación.

- Conexión contínua a la red con una buena conexión para permitir el acceso a múltiples usuarios, por ejemplo conexión ADSL o Cable de 1 Mb.

Este sistema sería más o menos correcto para que la Plataforma funcionara sin problemas, con las posibles ampliaciones comentadas por su hubiera saturación de sistema o similar.

2.1.6. Definición del Plan de Desarrollo y Simulación de la Arquitectura PRG-eLearn

El plan de desarrollo para la segunda fase se divide en distintas tareas que describen el proceso de desarrollo y simulación:

- Análisis Detallado de la Información de eLearning.

- Análisis y Diseño Detallado de un Prototipo PRG-eLearn.

- Instalación, Configuración y Prueba de la Plataforma de Simulación.

- Definición del Plan de Pruebas Funcionales y de Rendimiento del Prototipo PRG-eLearn.

- Instalación y Configuración del Prototipo PRG-eLearn.

- Realización del Plan de Pruebas Funcionales y de Rendimiento del Prototipo PRG-eLearn.

- Análisis de Resultados y Optimización de la Arquitectura PRG-eLearn.

2.2. Fase II: Desarrollo y Simulación de la Arquitectura PRG-eLearn

La ejecución del plan de desarrollo y simulación de la arquitectura propuesta se describe en los siguientes puntos:

2.2.1. Análisis Detallado de la Información de eLearning

En este apartado entraremos a describir en detalle la estructura del paquete de nuestra plataforma y los metadatos utilizados para describirlo, de cara a la implementación de un prototipo. En cuanto a la

información de e-learning en sí misma, debemos mencionar que no nos restringimos a ningún ámbito concreto. Las únicas restricciones vienen dadas por el hecho de que en nuestra plataforma los paquetes no incluyen los contenidos en sí mismos, sino que son referenciados a través de una URL ó se refieren a recursos no digitales.

El paquete del prototipo de nuestra plataforma es una implementación parcial de la versión 1.1.4 de la especificación *IMS Content Packaging*, así como de la versión 1.2.1 de la especificación *IMS Learning Resource Meta-Data* para los metadatos. Concretamente, al usar XML, nos basamos en los documentos *IMS Content Packaging XML Binding* en su versión 1.1.4 para el paquete de contenidos y en *IMS Learning Resource Meta-Data XML Binding* en su versión 1.2.1 para los metadatos.

También tenemos en cuenta, como hemos mencionado en apartados anteriores, las recomendaciones de *IMS Digital Repositories Interoperability* y de la más recientemente desarrollada *IMS General Web Services Base Profile*, ambas en su versión 1.0.

A continuación describiremos en detalle la estructura de nuestro paquete de contenidos, los metadatos que utilizaremos en la implementación del prototipo y su uso.

2.2.1.1. Estructura del paquete de contenidos en el prototipo de PRG-eLearn

En este subapartado comentaremos la estructura de un paquete de contenidos, descrita en su archivo imsmanifest.xml, en nuestra plataforma.

Como hemos comentado anteriormente, el paquete del prototipo de nuestra plataforma es una implementación parcial de la versión 1.1.4 de la especificación *IMS Content Packaging*. Recordemos brevemente que dicha especificación de IMS describe estructuras de datos que son usadas para proporcionar interoperabilidad de contenidos web con otros sistemas. Cuando usamos XML, dichas estructuras son elementos con sus correspondientes atributos. El objetivo de la especificación es definir un conjunto estandarizado de estructuras que sirva para intercambiar contenidos, así como agregar o desagregar paquetes de contenidos. Un paquete IMS está formado por 2 elementos principales:

- El manifiesto, **imsmanifest.xml**, que es un archivo XML que describe la estructura del paquete y los contenidos reales del mismo, si bien también puede referenciar recursos accesibles mediante una URL.

- Los contenidos, es decir, los archivos físicos que componen el paquete.

El paquete incorporado en un único fichero, generalmente comprimido en formato ZIP, se llama *Package Interchange File* (PIF).

Para una descripción completa de dicha especificación, remitirse a los documentos anteriormente citados, disponibles en las direcciones web listados en la bibliografía de este documento.

En el prototipo de nuestra plataforma no realizaremos envío de paquetes que incluyan los archivos reales, sino que tendremos recursos accesibles vía web. Por otro lado, permitimos también describir recursos no digitales, como libros, apuntes, etc. Esto se traduce en las siguientes diferencias con respecto a una implementación completa de la especificación *IMS Content Packaging*:

1. El paquete de nuestra plataforma se compone únicamente del archivo **imsmanifest.xml**.

2. Un elemento **<resource>** que se refiera a un contenido digital siempre poseerá un atributo href que contendrá el punto de entrada al recurso y será una URL absoluta.

3. No existen elementos **<file>,** al no existir recursos “locales”.

2.2.1.2. Metadatos utilizados en el prototipo de PRG-eLearn

En este subapartado mencionaremos qué metadatos utilizaremos en el prototipo de nuestra plataforma, en qué secciones del manifiesto aparecerán y su uso, ya sea para describir contenidos del paquete o para propósitos especificos en nuestra plataforma.

Nuestro prototipo utilizará parte de los metadatos descritos en la versión 1.2.1 de la especificación *IMS Learning Resource Meta-Data*. Dado que usamos XML, utilizamos concretamente el documento *IMS Learning Resource Meta-Data XML Binding*. Recordemos que dicha especificación proporciona una especificación de los metadatos que pueden ser utilizados para describir objetos de aprendizaje y sus contenidos. Los documentos anteriormente citados están disponibles en las direcciones web listados en el apartado de la bibliografía de este documento.

Debemos mencionar que no utilizaremos ningún metadato que no exista en dicha especificación para propósitos específicos de nuestra plataforma. Con ello aumentamos la interoperabilidad de nuestros paquetes de contenidos.

A continuación describimos en detalle cada uno de los metadatos que incluimos en nuestro prototipo. Utilizaremos el siguiente formato:

Descripción. Una descripción breve del metadato.

Multiplicidad. Número de apariciones del elemento dentro de su elemento padre.

Elementos donde pueden aparecer. Los elementos dentro del archivo imsmanifest.xml donde puede aparecer un árbol cuya raíz es <metadata> y cuya hoja es el metadato en nuestra plataforma.

Vocabulario. Ver el apartado 2.2.1.3. "Elementos usados globalmente". En este apartado enumeramos los posibles valores del elemento <value> de un metadato. Si un metadato no utiliza un vocabulario, omitimos este apartado.

Uso. Para qué se utiliza el metadato y/o de qué manera en el contexto de nuestra plataforma. Cuando no tengamos que añadir ninguna información relevante, sino que baste la dada en el apartado "descripción", omitiremos este apartado.

Atributos y uso. Atributos opcionales u obligatorios del elemento y su uso dentro de nuestra plataforma.

Elementos hijos. Los posibles hijos del elemento en nuestra plataforma, que pueden variar según el apartado del manifiesto donde aparezca la instancia de <metadata>.

2.2.1.2.1. Elemento <lom>

Descripción. Es el elemento raíz.

Multiplicidad. 1 y sólo 1 dentro de cada instancia de un elemento <metadata> del manifiesto.

Elementos donde pueden aparecer. Secciones <manifest> y <resource>.

Uso.

Si aparece dentro de <manifest>, describe el objeto virtual en su conjunto.

Si aparece dentro de <resource>, describe un recurso concreto.

Atributos y uso. Ninguno.

Elementos hijos.

Si aparece dentro de <manifest>: <general>, <lifecycle>, <metametadata> *(que no utilizaremos)*, <technical>, <educational> y <classification>.

Si aparece dentro de <resource>: <general>, <technical> y <educational>.

2.2.1.2.2. Elemento <general>

Descripción. Información general del elemento.

Multiplicidad. 0 o 1 dentro del elemento <lom>.

Elementos donde puede aparecer. <manifest> y <resource>.

Uso. Igual que <lom>.

Atributos y uso. Ninguno.

Elementos hijos.

Si aparece dentro de <manifest>: <identifier>, <title>, <catalogentry>, <description> y <keyword>.

Si aparece dentro de <resource>: <catalogentry>.

2.2.1.2.2.1. Elemento <identifier>

Descripción. Etiqueta global que identifica el elemento donde aparece.

Multiplicidad. 0 o 1 dentro del elemento <general>.

Elementos donde puede aparecer. <manifest>.

Uso. Será un identificador único cuyo valor coincide con el devuelto por el repositorio y con el almacenado en la base de datos auxiliar.

Atributos y uso. Ninguno.

Elementos hijos. Ninguno.

2.2.1.2.2.2. Elemento <title>

Descripción. Nombre dado al objeto de aprendizaje.

Multiplicidad. 0 o 1 dentro del elemento <general>.

Elementos donde puede aparecer. <manifest>.

Atributos y uso. Ninguno.

Elementos hijos. <langstring> (puede aparecer 1 o más veces dentro del elemento <title>, con un atributo xml:lang distinto en cada caso).

2.2.1.2.2.3. Elemento <catalogentry>

Descripción. Define una entrada dentro de un catálogo.

Multiplicidad. 0 o más dentro del elemento <general>.

Elementos donde puede aparecer. <manifest> y <resource>.

Uso. Siguiendo las recomendaciones de *IMS Content Packaging Best Practice and Implementation Guide*, permitimos el uso de este metadato (junto con sus etiquetas hijas) para la búsqueda e indexado dentro del repositorio tanto de objetos completos (si aparece en <manifest>) como de recursos concretos (si aparece en <resource>).

Atributos y uso. Ninguno.

Elementos hijos. <catalog> y <entry> (en cualquier caso).

2.2.1.2.2.3.1. Elemento <catalog>

Descripción. Nombre del catálogo.

Multiplicidad. 1 y sólo 1.

Elementos donde puede aparecer. <manifest> y <resource>.

Atributos y uso. Ninguno.

Elementos hijos. Ninguno.

2.2.1.2.2.3.2. Elemento <entry>

Descripción. Valor real de la entrada dentro del catálogo.

Multiplicidad. 1 y sólo 1 dentro del elemento <catalogentry>.

Elementos donde puede aparecer. <manifest> y <resource>.

Atributos y uso. Ninguno.

Elementos hijos. <langstring> (puede aparecer 1 o más veces dentro del elemento <entry>, con un atributo xml:lang distinto en cada caso).

2.2.1.2.2.4. Elemento <description>

Descripción. Una descripción textual del objeto de aprendizaje.

Multiplicidad. 0 o más dentro del elemento <general>.

Elementos donde puede aparecer. <manifest>.

Atributos y uso. Ninguno.

Elementos hijos. <langstring> (puede aparecer 1 o más veces dentro del elemento <description>, con un atributo xml:lang distinto en cada caso).

2.2.1.2.2.5. Elemento <keyword>

Descripción. Una colección de palabras clave o frases que describen el objeto de aprendizaje.

Multiplicidad. 0 o más.

Elementos donde pueden aparecer. <manifest>.

Atributos y uso. Ninguno.

Elementos hijos. <langstring> (puede aparecer 1 o más veces dentro del elemento <keyword>, con un atributo xml:lang distinto en cada caso).

2.2.1.2.3. Elemento <lifecycle>

Descripción. Características relacionadas con el estado actual del objeto y de aquellos que han contribuido al objeto de aprendizaje.

Multiplicidad. 0 o 1 dentro del elemento <lom>.

Elementos donde pueden aparecer. <manifest>.

Atributos y uso. Ninguno.

Elementos hijos. <status> y <contribute>.

2.2.1.2.3.1. Elemento <status>.

Descripción. El estado de este objeto de aprendizaje.

Multiplicidad. 0 o 1 dentro del elemento <lifecycle>.

Elementos donde pueden aparecer. <manifest>.

Vocabulario. Draft, Final, Revised y Unavailable.

Uso. Los paquetes pueden crearse a lo largo de varias sesiones y se almacenan directamente en el registro. Además tanto su estructura como su contenido pueden estar validados o no. Utilizamos este metadato para indicar si un objeto almacenado en el repositorio está disponible o no para evitar usos que no sean su terminación por parte del registrador o su validación por parte de los gestores de estructura y contenido. Desde nuestra plataforma sólo utilizaremos 2 valores de los 4 disponibles en el vocabulario:

- *Final*: indica que un objeto está disponible, o lo que es lo mismo, el registrador lo ha terminado, el gestor de estructura ha validado que la estructura es correcta y el gestor de contenidos ha validado que el contenido es coherente. En esta situación un usuario de la plataforma puede realizar cualquier operación con el paquete que le permita su rol (registrador, gestor, etc.) y su nivel de privilegio.

- *Unavailable*: indica que un objeto no está disponible, por no estar terminado y/o no estar validado a nivel de estructura y contenido. En esta situación el objeto tiene un uso restringido. Dependiendo del estado concreto del objeto (terminado/no, validado a nivel de estructura/no, contenido validado/no), almacenado en la base de datos auxiliar de la plataforma, el objeto sólo se podrá modificar (por parte del registrador que lo creó), validar a nivel de estructura (por parte del gestor de validación de estructura) y/o validarse a nivel de contenido (por parte del gestor de validación de contenido). El paquete no estará disponible para otros usuarios de la plataforma (aunque tengan el nivel de privilegio suficiente), por lo que no podrán realizar ninguna operación sobre el paquete. Tampoco estaría disponible para usuarios finales u otros sistemas en posibles ampliaciones de PRG-eLearn.

Atributos y uso. Ninguno.

Elementos hijos. <source> y <value>.

2.2.1.2.3.2. Elemento <contribute>

Descripción. Personas u organizaciones que han contribuido al estado del objeto de aprendizaje durante su evolución.

Multiplicidad. 0 o más dentro del elemento <lifecycle>.

Elementos donde pueden aparecer. <manifest>.

Uso. Ver 2.2.1.2.3.2.1. Elemento <role>.

Atributos y uso. Ninguno.

Elementos hijos. <role> y <centity>.

2.2.1.2.3.2.1. Elemento <role>

Descripción. Tipo de contribución.

Multiplicidad. 1 y sólo 1 dentro del elemento <contribute>.

Elementos donde pueden aparecer. <manifest>.

Vocabulario. Author, Publisher Unknown, Initiator, Terminator, Validator, Editor, Graphical Designer, Technical Implementer, Content Provider, Technical Validator, Educational Validator, Script Writer, Instructional Designer.

Uso. Utilizaremos 3 valores posibles del vocabulario para identificar a los diferentes tipos de usuarios presentes en PRG-eLearn que pueden haber contribuido al estado del objeto durante su evolución:

- *Author*. Se corresponde al registrador en nuestra plataforma.

- *Technical Validator*. Se corresponde al gestor de validación de estructura en nuestra plataforma.

- *Educational Validator*. Se corresponde al gestor de contenido de objetos de nuestra plataforma.

Atributos y uso. Ninguno.

Elementos hijos. <source> y <value>.

2.2.1.2.3.2.2. Elemento <centity>

Descripción. Las personas que han contribuido al objeto de aprendizaje.

Multiplicidad. 0 o más dentro del elemento <contribute>.

Elementos donde pueden aparecer. <manifest>.

Uso. El elemento tendrá el formato que se indica a continuación. Nótese que el valor de fn es el nombre del registrador o gestor y no el identificador del usuario en nuestra plataforma. La consulta de los objetos asociados a un determinado usuario para conocer cuáles puede modificar y/o validar se realiza directamente sobre la base de datos auxiliar de PRG-eLearn.

```
<centity>
      <vcard>
            begin:vcard
            fn: Nombre del registrador o gestor
            end:vcard
      </vcard>
</centity>
```

Atributos y uso. Ninguno.

Elementos hijos. <vcard>.

2.2.1.2.4. Elemento <technical>

Descripción. Agrupa las características técnicas del elemento en que aparece.

Multiplicidad. 0 ó 1.

Elementos donde pueden aparecer. <manifest> y <resource>.

Atributos y uso. Ninguno.

Elementos hijos. <format> y <location>.

2.2.1.2.4.1. Elemento <format>

Descripción. Describe el formato de los datos de un objeto de aprendizaje.

Multiplicidad. 1 y sólo 1. Nota: en el estándar es 0 o más.

Elementos donde pueden aparecer. <manifest> y <resource>.

Uso. Definirá el formato de los datos del objeto o recurso situado en el punto de entrada descrito en <location>. El contenido de dicho elemento debería ser un tipo MIME ó "non-digital". La lista completa de tipos MIME puede encontrarse en:

http://www.iana.org/assignments/media-types/

En el caso de los recursos digitales, y dado que en PRG-eLearn los paquetes no almacenan archivos físicos, muchos de estos tipos no serán necesarios en nuestra plataforma. No obstante, dado que la lista de formatos es muy amplia y para facilitar posibles ampliaciones futuras, no restringiremos el contenido de dicho elemento.

En cuanto a los recursos no digitales, se indicarán estableciendo como contenido del elemento "non-digital".

Atributos y uso. Ninguno.

Elementos hijos. Ninguno.

2.2.1.2.4.2. Elemento <location>

Descripción. La cadena usada para acceder al objeto de aprendizaje.

Multiplicidad. 1 y sólo 1. Nota: en el estándar es 0 o más.

Elementos donde pueden aparecer. <manifest> y <resource>.

Uso. El punto de entrada al objeto de aprendizaje ó al recurso. En el caso de los recursos no digitales, es la descripción de donde se encuentra (ver atributos y uso). Es necesario que exista coherencia entre los atributos del elemento <resource> del paquete, el contenido del elemento <format> hermano de esta instancia de <location> y los atributos y contenidos de esta instancia de <location> tal y como indicamos en la siguiente tabla:

Tipo de recurso	**IMS Content Packaging**			**IMS Learning Resource Metadata**		
	Elemento <resource>			Elemento <format>	Elemento <location>	
	¿Posee atributo href?	Valor del atributo href	Valor del atributo type (1)	Contenido del elemento	Valor del atributo type	Contenido del elemento
Digital	Sí	La URL del recurso	'webcontent' (2)	Un formato digital	'URI' (2)	La cadena de acceso al recurso. Por ejemplo, una URL.
No digital	No	-	'other' (2)	'non-digital' (2)	'TEXT' (2)	El lugar físico donde se encuentra el recurso.

Notas:

(1) el atributo type es obligatorio.

(2) los valores entre comillas son el contenido literal de un elemento o atributo, que no lleva comillas.

Atributos y uso.

Type.

Posibles valores:

URI: un recurso accesible a través de Internet con una dirección específica, como por ejemplo una URL.

TEXT: una descripción textual de donde se localiza el recurso. Utilizaremos este valor cuando el recurso sea no digital.

Elementos hijos. Ninguno.

2.2.1.2.5. Elemento <educational>

Descripción. Condiciones de uso del recurso.

Multiplicidad. 0 ó 1 dentro del elemento <lom>.

Elementos donde pueden aparecer. <manifest> y <resource>.

Atributos y uso. Ninguno.

Elementos hijos. <learningresourcetype>, <intendedenduserrole> y <context>.

2.2.1.2.5.1. Elemento <learningresourcetype>

Descripción. Tipo de recurso.

Multiplicidad. 0 ó más dentro del elemento <educational>.

Elementos donde pueden aparecer. <manifest> y <resource>.

Vocabulario. Exercise, Simulation, Questionnaire, Diagram, Figure, Graph, Index, Slide, Table, Narrative Text, Exam, Experiment, ProblemStatement, SelfAssesment.

Atributos y uso. Ninguno.

Elementos hijos. <source> y <value>.

2.2.1.2.5.2. Elemento <intendedenduserrole>

Descripción. El usuario esperado del objeto de aprendizaje.

Multiplicidad. 0 ó más dentro del elemento <educational>.

Elementos donde pueden aparecer. <manifest>.

Vocabulario. Teacher, Author, Learner, Manager.

Atributos y uso. Ninguno.

Elementos hijos. <source> y <value>.

2.2.1.2.5.3. Elemento <context>

Descripción. El entorno de aprendizaje típico donde se va a usar el objeto de aprendizaje.

Multiplicidad. 0 ó más dentro del elemento <educational>.

Elementos donde pueden aparecer. <manifest>.

Vocabulario. Primary Education, Secondary Education, Higher Education, University First Cycle, University Second Cycle, University Postgrade, Technical School First Cycle, Technical School Second Cycle, Proffesional Formation, Continous Formation, Vocational Training.

Atributos y uso. Ninguno.

Elementos hijos. <source> y <value>.

2.2.1.2.6. Elemento <classification>

Descripción. Descripción de una característica del objeto de aprendizaje mediante entradas en clasificaciones.

Multiplicidad. 0 a 2 veces, según las características que se describan (ver uso).

Elementos donde pueden aparecer. <manifest> y <resource>.

Uso. En el prototipo de nuestra plataforma daremos 2 usos muy concretos a las clasificaciones, dejando para posibles ampliaciones la posibilidad de introducir clasificaciones libremente para otros usos. Tenemos un uso distinto según al elemento del paquete donde aparece del elemento <classification> y sus descendientes. Dichos usos en nuestro prototipo son los siguientes:

Si aparece dentro del elemento <manifest>, lo utilizaremos para definir si un objeto es público o privado. Un objeto podrá definirse público o privado, generándose una estructura de la forma siguiente:

```
<classification>
	<purpose>
		<source>
			<langstring xml:lang=”x-none”>
				LOMv1.0
			</langstring>
		</source>
	<value>
		<langstring xml:lang=”x-none”>
			Accesibility Restrictions
		</langstring>
	</value>
	</purpose>
	<taxonpath>
		<source>
			<langstring>Privacidad</langstring>
		</source>
```

```
            <taxon>
                  <entry>
                        <langstring>(1)</langstring>
                  </entry>
            </taxon>
      </taxonpath>
</classification>
```

(1) toma uno de los siguientes valores: público ó privado.

Si aparece dentro del elemento <resource>, lo utilizaremos para definir el nivel de privilegio de un recurso. El nivel de privilegio va de 1 a 9, siendo 1 el menor y 9 el mayor. Los usuarios también poseen un nivel de privilegio, y la accesibilidad a un objeto se define de la forma siguiente: un usuario de nivel n puede acceder a los contenidos de nivel m, 1<=m<=n, es decir, a los contenidos con nivel de privilegio igual o menor. Al introducir el nivel de privilegio de un objeto se genera una estructura de la forma siguiente:

```
<classification>
      <purpose>
            <source>
                  <langstring xml:lang=”x-none”>
                        LOMv1.0
                  </langstring>
            </source>
            <value>
                  <langstring xml:lang=”x-none”>
                        Security Level
                  </langstring>
            </value>
      </purpose>
      <taxonpath>
            <source>
                  <langstring>
                        Nivel de privilegio
                  </langstring>
            </source>
            <taxon>
                  <entry>
                        <langstring>(1)</langstring>
                  </entry>
            </taxon>
      </taxonpath>
</classification>
```

(1) toma como valor un entero del intervalo [1,9]

Atributos y uso. Ninguno.

Elementos hijos. <purpose> y <taxonpath>.

2.2.1.2.6.1. Elemento <purpose>

Descripción. Objetivo de la clasificación.

Multiplicidad. 1 y sólo 1 dentro del elemento <classification>. Nota: en el estándar es 0 ó 1.

Elementos donde pueden aparecer. <manifest> y <resource>.

Vocabulario. Accesibility Restrictions, Security Level. Nota: el estándar prevee otras posibilidades.

Uso.

Si aparece dentro de <manifest> la clasificación se está usando para indicar si el objeto es público o privado. En este caso, el contenido del elemento <value> hijo de esta instancia de <purpose> se establece a Accesibility Restrictions.

Si aparece dentro de <resource> la clasificación se está usando para indicar el nivel de privilegio necesario para acceder a un recurso. En este caso, el contenido del elemento <value> hijo de esta instancia de <purpose> se establece a Security Level.

Atributos y uso. Ninguno.

Elementos hijos. <source> y <value>.

2.2.1.2.6.2. Elemento <taxonpath>

Descripción. Un camino taxonómico dentro de una clasificación específica.

Multiplicidad. 1 y sólo 1 dentro del elemento <classification>. Nota: en el estándar son 0 ó más.

Elementos donde pueden aparecer. <manifest> y <resource>

Atributos y uso. Ninguno.

Elementos hijos. <source> y <taxon>.

2.2.1.2.6.2.1. Elemento <source>

Descripción. El nombre de la clasificación.

Multiplicidad. 1 y sólo 1 dentro del elemento <taxonpath>. Nota: en el estándar es 0 ó 1.

Elementos donde pueden aparecer. <manifest> y <resource>.

Uso.

Si aparece dentro de <manifest> la clasificación se está usando para indicar si el objeto es público o privado. En este caso, el contenido del elemento <langstring> hijo de esta instancia de <source> se establece a Privacidad.

Si aparece dentro de <resource> la clasificación se está usando para indicar el nivel de privilegio necesario para acceder a un recurso. En este caso, el contenido del elemento <langstring> hijo de esta instancia de <source> se establece a Nivel de privilegio.

Atributos y uso. Ninguno.

Elementos hijos. <langstring>.

2.2.1.2.6.2.2. Elemento <taxon>

Descripción. Una entrada en una clasificación.

Multiplicidad. 1 y sólo 1 dentro del elemento <taxonpath>.

Elementos donde pueden aparecer. <manifest> y <resource>.

Atributos y uso. Ninguno.

Elementos hijos. <entry>.

2.2.1.2.6.2.2.1. Elemento <entry>

Descripción. El valor del nodo del árbol.

Multiplicidad. 1 y sólo 1 dentro del elemento <taxon>. Nota: en el estándar es 0 ó 1.

Elementos donde pueden aparecer. <manifest> y <resource>.

Uso.

Si aparece dentro de <manifest> la clasificación se está usando para indicar si el objeto es público o privado. En este caso, el contenido del elemento <langstring> hijo de esta instancia de <entry> se establece a uno de los valores siguientes: público ó privado.

Si aparece dentro de <resource> la clasificación se está usando para indicar el nivel de privilegio necesario para acceder a un recurso. En este caso, el contenido del elemento <langstring> hijo de esta instancia de <entry> se establece a un valor entero que debe estar situado dentro del intervalo [1,9].

Atributos y uso. Ninguno.

Elementos hijos. <langstring>.

2.2.1.3. Elementos usados globalmente

En este apartado describiremos los elementos XML que pueden aparecer como hijos de distintos metadatos. Utilizaremos el siguiente formato:

Descripción. Una descripción breve del metadato.

Atributos y uso. Atributos opcionales u obligatorios del elemento.

Elementos hijos. Los posibles hijos del elemento.

2.2.1.3.1. Elemento <langstring>

Descripción. Una frase en algún idioma humano.

Atributos y uso.

xml:lang. Representa el lenguaje de los contenidos del elemento. El valor "x-none" se requiere cuando <langstring> aparece como hijo de <source> o de <value> en un vocabulario. En nuestro prototipo consideraremos únicamente los siguientes valores posibles del atributo:

Valor del atributo	Idioma que representa
"en"	Inglés
"es"	Español
"fr"	Francés

Elementos hijos. Ninguno.

2.2.1.3.2. Vocabularios

Descripción. Podemos considerarlos tipos enumerados. El elemento donde aparece toma un valor concreto de un conjunto de valores posibles. El uso de un vocabulario dentro de un elemento no se traduce por un único elemento XML, sino por dos: <source> y <value>.

Atributos y uso. Ninguno.

Elementos hijos. <source> y <value>.

2.2.1.3.2.1. Elemento <source>

Descripción. La fuente del valor. En nuestra plataforma el contenido del elemento <langstring> hijo de este elemento es siempre LOMv1.0.

Atributos y uso. Ninguno.

Elementos hijos. <langstring>.

2.2.1.3.2.2. Elemento <value>

Descripción. El contenido del elemento <langstring> hijo de esta instancia de <value> es el valor que se toma dentro del vocabulario.

Atributos y uso. Ninguno.

Elementos hijos. <langstring>.

2.2.1.3.3. Elemento <vcard>

Descripción. Define una tarjeta electrónica virtual. Aunque el estándar permite utilizarlo en varios elementos distintos y en distintos formatos, en el prototipo de nuestra plataforma sólo aparece dentro del elemento 2.2.1.2.3.2.2. <centity> usado con un formato específico. Ver este apartado para más detalles.

Atributos y uso. Ninguno.

Elementos hijos. Ninguno.

2.2.2. Análisis y Diseño Detallado de un Prototipo PRG-eLearn.

A continuación, vamos a describir el prototipo que será desarrollado para el funcionamiento de la Plataforma de Registro y Gestión de información e-Learning.

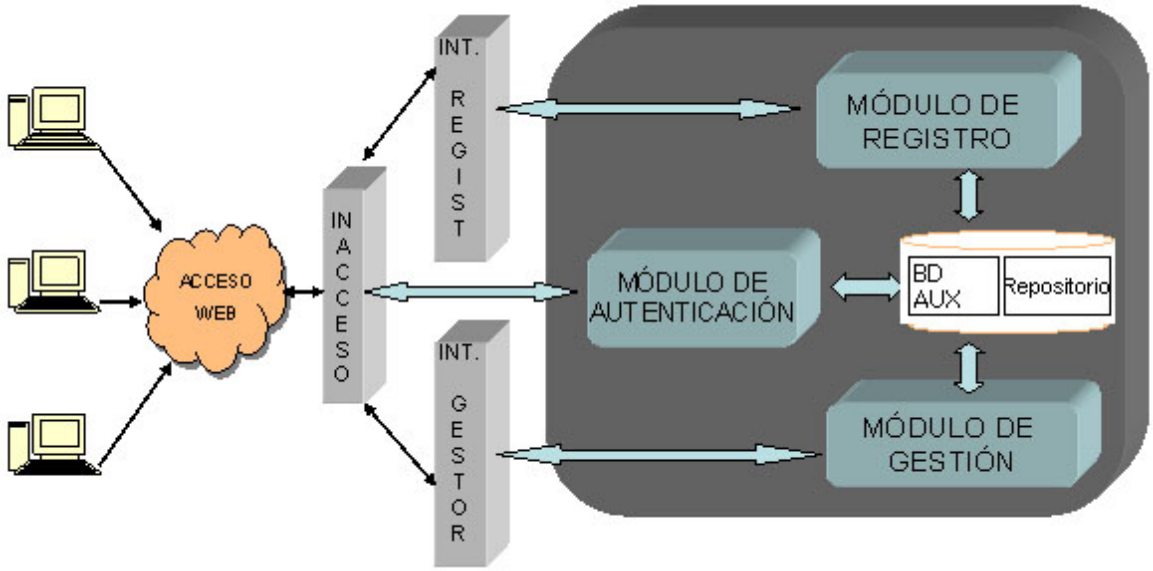
Cabe recalcar que al desarrollar simplemente un prototipo y no un desarrollo completo de la plataforma, no se cubrirán todas las funcionalidades descritas en los apartados anteriores, sino que se desarrollará una versión básica de la plataforma. Uno de los cambios más importantes en la arquitectura es la ausencia de un repositorio en sí, sino que se simula el almacenamiento de los objetos en la base de datos. Esto no altera la claridad en la muestra de las funcionalidades en el prototipo.

Así pues, en el prototipo se va a implementar la funcionalidad correspondiente a las partes del usuario Registrador y el usuario Gestor de Validación de Estructura. Dentro de estos usuarios, se desarrollarán las principales funciones de los mismos y serán descritas a continuación.

Definimos las etapas de desarrollo del prototipo que vamos a abarcar, diferenciando los diferentes módulos presentes en el análisis y diseño de la plataforma en anteriores apartados.

Para considerar los contenidos de e-learning los marcaremos con los siguientes metadatos:

- *Privado/público*. Los contenidos pueden ser creados para poder ser itulizados por quien pueda tener acceso a ellos, o bien simplemente para utilización de su creador.
- *Nivel de privilegio*. Cuando múltiples tipos de usuarios pueden acceder a un contenido, algunos contenidos pueden ser accesibles a algunos y a otros no. En nuestro caso establecemos nueve niveles de privilegio. Este nivel va desde 1 hasta 9, siendo 1 el menor y 9 el mayor. Los usuarios con nivel de privilegio *n* pueden acceder a los contenidos con nivel *n*, *n*-1, *n*-2...



Esquema de la arquitectura del prototipo PRG-eLearn

MÓDULO DE AUTENTICACIÓN

En primer lugar, se desarrollará la parte implicada en la autenticación de usuarios para ofrecer las funciones de uno u otro a la hora de acceder a la plataforma, y consultar qué privilegios pueden tener a la hora de interactuar con la misma.

Este módulo es bastante sencillo y simplemente se compondrá de una pantalla de validación de usuario para empezar a trabajar con la plataforma, y según el tipo de usuario validado, y si está presente en la base de datos, desplegará el menú correspondiente a su perfil y ya podrá empezar a utilizar la plataforma.

Este módulo, por tanto, desarrollará una pantalla de validación y por detrás la lógica de negocio necesaria para consultar si existe el usuario en la base de datos y el perfil correspondiente.

MÓDULO DEL USUARIO REGISTRADOR

Esta parte del prototipo, implementará las principales funciones asociadas al usuario Registrador que son básicamente las siguientes:

- Creación de nuevos Objetos Virtuales
- Modificación de Objetos Virtuales propios.
- Eliminación de Objetos Virtuales Propios.
- Consulta de los Objetos Virtuales creados por el usuario.
- Consulta de Notas asociadas al Objeto Virtual desarrollado.

Para cada una de estas funciones se desarrollarán sus pantallas correspondientes, con las opciones que definan cada una de las funcionalidades.

MÓDULO DEL USUARIO GESTOR DE VALIDACIÓN DE ESTRUCTURA

Este perfil de usuario desarrollará tres funcionales principales, que son las siguientes:

- Validación de Objetos que estén pendientes de Validación.
- Creación de Notas asociadas a los Objetos, para dar consejos a los usuarios registradores sobre posibles errores en los objetos.
- Publicación de los Objetos Virtuales correctos y que ya hayan sido validados por el mismo usuario gestor. En ese momento se pondrán a disposición de los futuros usuarios finales.

La parte de creación de notas asociadas a los objetos, se dispondrá a continuación de que se muestre alguno de los objetos que se están validando para facilitar la labor del gestor y se asocie automáticamente la nota descrita con el objeto correspondiente. En la parte del Registrador, se podrán consultar las notas asociadas a los objetos que haya desarrollado el usuario gestor que esté trabajando con la plataforma.

En la parte de Validación de Objetos, se podrá elegir cualquiera de los objetos que estén pendientes de validar en ese momento por alguno de los gestores del sistema. Una vez se seleccione alguno de esos objetos, se mostrará en una pantalla a continuación donde el gestor podrá consultar la validez de la

estructura que posea dicho objeto, cómo está formado, los atributos que posee, etc. En resumen, todo lo que esté relacionado con la validación de estructura y las comprobaciones que atañe a éste tipo de gestor.

La Publicación de Objetos es más sencilla, ya que solamente se ocupa de publicar los objetos que han sido creados y ya se ha comprobado la validez de su estructura, y por tanto, pueden pasar a ser utilizados por parte de los futuros usuarios de los objetos.

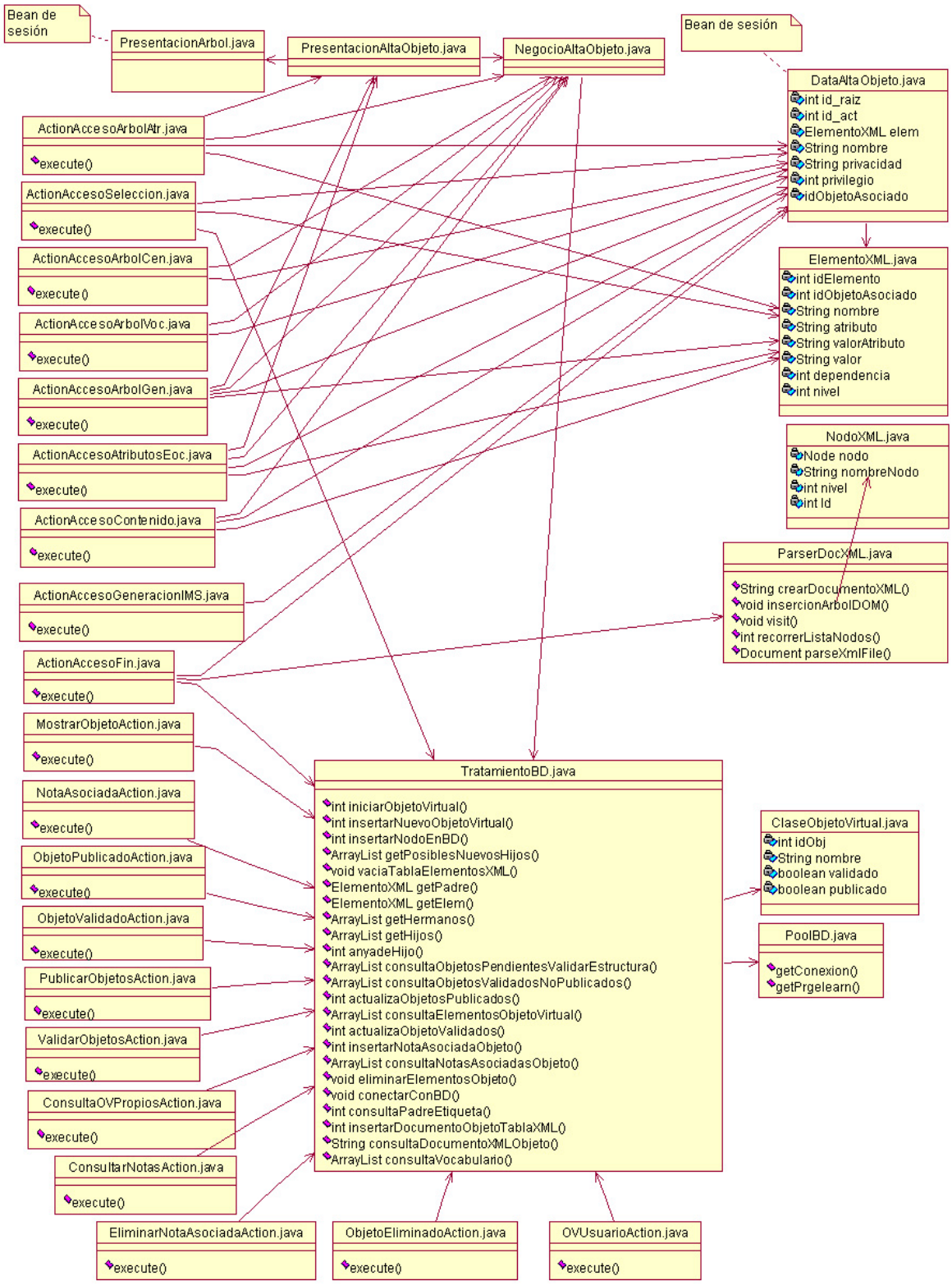


Diagrama de clases de PRG-eLearn

En la parte izquierda e inferior del diagrama, vemos las clases Java correspondientes a los *action* que llevan a cabo la lógica de negocio, tanto del registrador como del gestor. En el diagrama se han obviado los *action form* por cuestiones de simplificación.

La capa de presentación se lleva a cabo en archivos de tipo *jsp* que se apoyan en la clase PresentacionAltaObjeto.java.

El trabajo de persistencia lo llevana cabo las clases TratamientoBD.java y NegocioAltaObjeto.java.

2.2.3. Instalación, Configuración y Prueba de la Plataforma de Simulación

La plataforma de simulación para el desarrollo del prototipo que quiere ofrecerse a los usuarios que lo utilicen, se basa en tecnologías J2EE que permiten ofrecerlo como Aplicación Web accesible a todos los usuarios que tengan acceso a la red, y que facilita su desarrollo gracias al conocimiento y experiencia que poseemos de dichas tecnologías y del cual se conocen sus favorables características.

Para aprovechar estas tecnologías y poder ofrecerlo vía Web, necesitará utilizarse alguno de los Servidores de Aplicaciones que se ofrecen en el mercado para poder lanzar la aplicación y que sea accesible a los usuarios, y por tanto, comentaremos a continuación cuál se ha elegido y por qué razones.

Por último, se comenta la necesidad de contar con una base de datos con la que trabajar y controlar las diferentes acciones que se produzcan en la aplicación con respecto los diferentes elementos con los que se trabaje en la misma, como pueden ser: objetos virtuales que se creen, qué usuarios trabajan en la aplicación y qué objetos han creado, cuál es la estructura de los objetos, etc.

Para el desarrollo e implementación de todos los elementos que formen las pantallas, que recojan la lógica de negocio o la lógica de acceso a la base de datos por parte de la aplicación se ha utilizado el entorno de desarrollo “netBeans 5.0” debido a su facilidad de manejo, de instalación, de configuración y de integración con el framework Struts con el que se quería trabajar para facilitar la división de módulos que recoge la filosofía del Modelo - Vista – Controlador, y con uno de los más sencillos y completos servidores de aplicación con el que poder trabajar y proporcionar todas las características requeridas por el prototipo, que es Tomcat.

Simplemente, se obtuvo de la Web de netBeans el programa netBeans 5.0 que es de libre distribución y se instaló en los equipos de desarrollo, seleccionando trabajar con el framework Struts que viene integrado en el propio programa, y que facilita la definición de las clases Action y ActionForm y el mapeo ulitizado para el flujo de navegación de pantallas, que son la base para trabajar con la lógica de negocio independientemente de las pantallas que muestran los datos y con las que interactúa el usuario, a pesar de desconocer antes de este desarrollo el comportamiento de dicho framework.

El servidor de aplicaciones utilizado es Tomcat 5.5.9, que viene precisamente integrado en el entorno de aplicación y por tanto la aplicación se lanza de forma fácil y sencilla desde el propio entorno y no es necesario configurarlo más que para proporcionar un pool de conexiones que es el procedimiento más adecuado para que las aplicaciones Web con acceso a base de datos, se comporten de una manera correcta y fiable.

La configuración de dicho pool de conexiones, es muy sencilla para este servidor y basta con seguir una serie de pasos que se han descrito más adelante en la configuración del prototipo de la aplicación, y que incluso viene detallado en la ayuda que incorpora el entorno de desarrollo en el que se trabaja, por tanto no vamos a explayarnos más sobre él en este momento.

Por último, comentamos un elemento tan importante como los anteriores como es la base de datos con la que se trabaja en la aplicación, y donde se guardan todos los datos necesarios para la interacción con la misma. En este caso, se ha utilizado mySQL debido una vez más, a su facilidad de uso e instalación, su integración con el sistema y la abundante documentación para trabajar con ella.

Para trabajar con esta base de datos de una forma más intuitiva que la ofrecida por el modo consola que la acompaña, se trabaja con una herramienta que ofrece interacción con mySQL de forma visual y que obviamente facilita el trabajo con la base de datos que se vaya a definir y que se obtiene en la misma página Web de la que se obtiene el programa y que se llama mySQL Control Center.

Todos los elementos utilizados y comentados que forman la Plataforma de simulación son de libre distribución y de fácil acceso a través de la Web, dos de las razones más importantes por los que se han utilizado y por los que se conocían.

2.2.4. Definición del Plan de Pruebas Funcionales y de Rendimiento del Prototipo PRG-eLearn

Se pretende realizar una serie de pruebas funcionales y de rendimiento sobre la aplicación desarrollada, con las que poder comprobar el funcionamiento más o menos real del prototipo implementado comprobando su correcto funcionamiento y su comportamiento para múltiples accesos y diferentes usuarios.

Por tanto, seguiremos dos líneas principales para realizar las pruebas de la aplicación y que se resumen en los siguientes puntos:

- Pruebas del correcto funcionamiento de la aplicación probando todas las opciones que proporciona la plataforma para los diferentes tipos de usuario implementados, intentando forzar errores para comprobar su comportamiento en esos casos, y realizando las acciones propias que ofrece la aplicación.

    - Para el usuario ***Registrador:***

        - Creación de varios objetos virtuales, por parte de uno y/o varios usuarios.
        - Consulta de dichos objetos para cada usuario registrador diferente.
        - Eliminación correcta de objetos virtuales y notas de errores.
        - Modificación y consulta de los objetos.
    - Para el usuario ***Registrador:***
        - Validación de alguno o varios de los objetos virtuales creados.
        - Creación de notas asociadas a los mismos.
        - Publicación de alguno o varios de los objetos virtuales validados.
- Comportamiento del prototipo ante accesos múltiples a la aplicación de forma simultánea desde el mismo equipo o diferentes.
    - Creaciones de objetos por el registrador y validaciones de los mismos de forma prácticamente inmediata por el gestor o gestores, para comprobar un correcto acceso múltiple.
    - Creación de notas asociadas a objetos por parte del gestor o gestores, y consulta de las mismas por parte del registrador de forma simultánea para comprobar si se actualizan al instante.
    - Compatibilidad y comportamiento de la aplicación con los distintos exploradores web.
- Comprobar posibles comportamientos anómalos de la aplicación, introduciendo datos erróneos para ver su comportamiento, o dejando textos en blanco

Siguiendo este plan de pruebas de la aplicación, se intentarán arreglar los errores que puedan surgir que se espera que sean mínimos, y para ofrecer un prototipo final lo más completo y fiable posible. Más adelante se comentarán los resultados obtenidos al completar el plan de pruebas del prototipo.

2.2.5. Instalación y Configuración del Prototipo PRG-eLearn

El prototipo de Plataforma de Registro y Gestión de Información e-Learning ha sido desarrollado mediante tecnologías J2EE para poder ser ofrecido como Aplicación Web accesible para los diferentes usuarios finales de la Plataforma, ya que las nuevas tecnologías y corrientes tecnológicas, implican obviamente un óptimo alcance de las aplicaciones desarrolladas a través de la Web para todos los usuarios comentados anteriormente.

Por tanto para su mejor aprovechamiento y utilización, será ofrecida para ser accedida a través de la Web y para ello se instalará en algún Servidor de Aplicaciones Web que soporte interpretación y uso de tecnologías J2EE y que soporte la configuración de un Pool de Conexiones para el acceso e interacción de múltiples usuarios a la vez, sin producirse errores sobre la base de datos.

En este caso, se ha utilizado el entorno de desarrollo "**netBeans 5.0**" debido a su facilidad de manejo y de integración con el framework Struts, en el que se ha basado la implementación del prototipo para respetar y desarrollar respetando la filosofía del Modelo - Vista - Controlador.

El Servidor de Aplicaciones utilizado ha sido "**Tomcat 5.5.9**" debido a su fácil manejo e integración con el entorno de desarrollo utilizado, y porque ofrece las características anteriormente comentadas para un buen servidor de aplicaciones Web: pool de conexiones, interpretación y uso de aplicaciones desarrolladas con tecnologías J2EE (jsp, servlets, etc.), etc. Además ofrece una fácil integración junto al entorno de desarrollo netBeans ya que se instala con él, y por tanto, facilita mucho el trabajo del programador.

Dentro del desarrollo de la aplicación, necesitamos la presencia de diferentes librerías y archivos que son fundamentales para el correcto funcionamiento de la aplicación en el servidor, según se ha realizado el desarrollo de la misma. Son las siguientes:

- **Librerías del framework Struts (versión 1.2.7):** Librerías imprescindibles para el funcionamiento del framework Struts en el que se basa la aplicación.

- **Struts-config.xml:** Archivo de configuración de Struts, donde se guardan todas las referencias a las clases Action, ActionForm y los jsp encargados de mostrar las pantallas de la aplicación y que son llamados por los diferentes Action.

- **Librería xerces.jar:** Librería encargada del parcheo de los ficheros XML que representan los diferentes *manifest* de los objetos virtuales registrados por los usuarios.

- **Librerías mysql-connector-java:** Librería necesaria para la conexión de la aplicación con la base de datos y para poder realizar todas las operaciones necesarias sobre la misma. Es necesario que esta librería se incluya dentro de las librerías utilizadas por el servidor

de aplicaciones que despliegue la aplicación, si no no funcionará el pool de conexiones que se defina en el mismo.

Todos estos elementos tendrán que estar presentes dentro del directorio WEB-INF de la aplicación para su correcto funcionamiento y más concretamente, en sus correspondientes directorios:

- ***WEB-INF/lib:*** las librerías que necesitamos irán dentro de este directorio.
- ***WEB-INF/classes***: aquí tendremos las clases correctamente compiladas y en sus correspondientes paquetes.
- ***WEB-INF***: en la raíz de este directorio, tendremos los archivos de configuración del servidor, así como los de configuración de struts.

El prototipo de la Plataforma, trabaja de forma activa y contínua con la base de datos auxiliar que acompaña a la aplicación y que se llama "***prelearn***". Esta base de datos ha sido implementada y definida con **mySQL** ya que este gestor de bases de datos no requiere demasiados conocimientos de gestión de bases de datos, y que a pesar de tener algunas carencias, ofrece las características necesarias y suficientes para el fácil y rápido desarrollo y mantenimiento de los datos necesarios que trabajan con el prototipo desarrollado.

Además de todo lo comentado anteriormente, se necesita configurar un pool de conexiones en el servidor con el que poder trabajar de forma segura y eficaz con la base de datos. Habrá que configurar el pool en el servidor de aplicaciones (Tomcat 5.5.9 en nuestro caso) en el que lancemos la aplicación según las directrices que indique el propio servidor y que variarán según sea el servidor elegido. Para ello, se realizan varios pasos en el proyecto requeridos para la definición del pool.

Antes de todo, comentar que se ha preferido configurar un pool de conexiones propio del servidor de aplicaciones que lanza la aplicación, debido a que se ha probado a utilizar un pool de conexiones que ofrece el propio framework Struts y que se definía en su fichero de configuración principal (*struts-config.xml*), sin embargo la necesidad de obtener la configuración del pool a través de la *request* en cada una de las clases *Action* en las que quisiéramos acceder a la base de datos, no nos parecía la manera correcta de hacerlo, y por ello se ha optado a trabaajr con un pool de conexiones definido en el servidor.

CONFIGURACIÓN DEL POOL DE CONEXIONES PARA TOMCAT 5.5.9

Antes de configurar el pool, habrá que instalar y configurar el servidor de aplicaciones para que lance la aplicación correctamente, pero esto sólo como comentario, ya que no se explicará ni su instalación

ni su configuración, simplemente se detalla la instalación y configuración del prototipo de la Plataforma PRG y algunas de sus necesidades.

Se van a determinar los pasos que hay que seguir para configurar este pool dentro del entorno de desarrollo que se ha utilizado (netBeans en nuestro caso) al haber sido desarrollada la aplicación con él, y puesto que este servidor de aplicaciones se instala, integra y se lanza directamente desde dicho entorno.

Es obvio que esta configuración se hará de otra forma para diferentes servidores de aplicación la configuración de este pool, en los que se quiera lanzar la aplicación. Aquí están los pasos para configurar el pool en el Tomcat y que son fáciles de determinar:

- En primer lugar tenemos que crear un usuario para el Servidor Tomcat con permisos de "administrador" y "manager". Para ello, vamos al directorio de instalación del **Tomcat/conf** y buscamos el archivo *tomcat-users.xml.*
    - Se edita el archivo para introducir un nuevo usuario y se le asigna: *username*, *password*, y *roles (roles = "manager,admin").*
- Se rearranca el servidor y después, se accede a la consola de administración del servidor Tomcat, validándonos con los datos del nuevo usuario administrador.
- En la consola de Administración, selecciona "*Fuentes de Datos*".
- Se selecciona "*Crear nueva Fuente de Datos*".
- Definimos los valores para los campos requeridos de la Fuente de datos. Serán los siguientes campos (los rellenamos con los valores que se han utilizado para el desarrollo del prototipo):
    - **Nombre JNDI**: `jdbc/prgelearn` - ("jdbc/<bd>").
    - **URL Fuente de Datos**: `jdbc:mysql://localhost:3306/prgelearn` - ("`jdbc:mysql://<host>:<Puerto>/<nombre_bd>`")
    - **JDBC Driver Class**: `org.gjt.mm.mysql.Driver`
    - **User Name**: `root`
    - **Password**: (vacío)

- Se guarda la configuración y pulsamos “Acometer Cambios”. Ya se puede salir de la consola de Administración.

Ya se ha creado el pool de conexiones en el Servidor Tomcat, y ahora nos queda un último paso para poder ejecutar las clases encargadas del acceso a la base de datos, debemos comprobar que los siguientes archivos de configuración tienen los datos del pool, sino habrá que meterlos:

- Accedemos al archivo ***web.xml*** en el directorio WEB-INF y comprobamos/introducimos la siguiente referencia, salvando a continuación:

```
<resource-ref>
<description>Pool Conexiones Tomcat</description>
        <res-ref-name>jdbc/prgelearn</res-ref-name>
        <res-type>javax.sql.DataSource</res-type>
        <res-auth>Container</res-auth>
</resource-ref>
```

- Ahora se accede al archivo *context.xml* dentro del directorio META-INF, y comprobamos/introducimos la siguiente referencia, salvando a continuación:

```
<Context path="/prgelearn">
        <ResourceLink name="jdbc/prgelearn"
        type="javax.sql.DataSource"
        global="jdbc/prgelearn"/>
        <Logger
        ClassName="org.apache.catalina.logger.FileLogger"
        prefix="prgelearn" suffix=".log" timestamp="true"/>
</Context>
```

A partir de ahora ya tendríamos el pool de conexiones configurado y listo para funcionar y proporcionar el acceso seguro a la base de datos interactuando con la aplicación.

Finalmente, para que la aplicación empiece a estar disponible y funcionar, habrá que configurar el servidor para que coja el directorio donde instalemos la aplicación, o que él mismo (*Tomcat* en este caso) despliegue el archivo de distribución que se genera con la aplicación y que será un fichero con extensión ***war***.

2.2.6. Realización del Plan de Pruebas Funcionales y de Rendimiento del Prototipo PRG-eLearn

Se van a comentar los datos obtenidos en la realización del plan de pruebas definido en un punto anterior, comentando en qué han consistido las pruebas realizadas y cuáles han sido los resultados asociados a las mismas.

En primer lugar se ha llevado a cabo una prueba de navegación e interacción con la plataforma, accediendo a todas las opciones y pantallas de los diferentes tipos de usuarios, seleccionando cada una de sus opciones de menú y comprobando que cada una de ellas realmente producía la acción asociada a estas opciones.

Así pues en primer lugar vamos a determinar cuál ha sido la primera parte de las pruebas, y qué es lo que se ha llevado a cabo en ellas. Empezaremos por el usuario Registrador y a continuación, comentaremos la parte del Gestor. En la segunda parte de las pruebas, comentaremos los resultados obtenidos al probar la funcionalidad y rendimiento de la plataforma ante varios usuarios a la vez y desde diferentes equipos.

Antes de realizar las pruebas de cada uno de los usuarios disponibles, se ha comprobado que la validación de autenticación para entrar al prototipo es correcta, y que permite la entrada sólo a aquellos usuarios que han sido insertados previamente en la base de datos. También se ha comprobado que se comporta correctamente ante errores forzados, como son: usuarios incorrectos, falta de identificador y/o paswword, etc.

PRUEBAS REGISTRADOR

Las pruebas de este módulo se han centrado en la opción de creación de nuevos objetos virtuales, si bien se han comprobado también el resto de opciones. Hemos optado por generar un árbol completo, es decir, por introducir al menos un elemento de cada tipo. En concreto, se han comprobado los siguientes puntos:

- Corrección de la navegación. Se ha comprobado que se redireccionaba a la pantalla correcta en función del elemento seleccionado. Un elemento que sólo puede tener hijos, sin contenido, no pasa por la pantalla de introducción de contenido, otro que no puede tener atributos no pasa por la pantalla de introducción del valor de un atributo, etc.

- Corrección de la funcionalidad. Se ha comprobado que cada opción de las pantallas realizaba correctamente su cometido. Por ejemplo, que el botón de eliminar efectivamente elimina el elemento seleccionado y todos sus hijos, y que los cambios se reflejan correctamente tanto en base de datos como en pantalla.

- Corrección del control de errores. Se ha comprobado que los campos obligatorios no pueden dejarse en blanco y que en caso de que no se produzca un error vuelvan a aparecer vacíos la siguiente vez que se pase por esa pantalla.

- Corrección de otros aspectos de IMS. Además de lo comentado en "corrección de la navegación": se han comprobado otros aspectos relacionados con IMS, como los posibles hijos de un elemento o los posibles valores de un vocabulario. No obstante, conviene recordar que son los gestores los encargados de la corrección de algunos aspectos, como los posibles valores que puede tomar formato o el número máximo de hijos que tiene cada elemento.

- También se ha comprobado que se pueden consultar correctamente los objetos propios de el usuario que esté validado en sesión y que puede interactuar con ellos, mosdificándolos o eliminándolos del repositorio.

- También se ha comprobado que se muestra correctamente la nota o notas asociadas a algún objeto determinado si las hubiera, y que se pueden eliminar actualizándose correctamente en base de datos para no volver a ser mostradas.

PRUEBAS GESTOR

Una vez realizada la creación de uno o varios objetos por parte del Registrador, se ha comprobado la funcionalidad de las opciones de este tipo de usuario, siguiendo las opciones proporcionadas del menú.

- ***Validación de Objetos pendientes de Validar:*** se ha comprobado que se obtienen correctamente aquellos objetos virtuales pendientes de validar y no otros, y se validan correctamente, y que al actualizar la lista no se vuelven a mostrar.

- ***Creación de Nota Asociada a Objeto:*** asociamos cuantas notas queramos a los objetos y se guardan correctamente en base de datos para ser mostradas al usuario registrador. Si no introducimos nada, nos avisa de ello para que volvamos a introducirla si lo deseamos.

- ***Publicación de Objetos validados sin publicar:*** se comprueba que la lista muestra correctamente aquellos objetos que estén pendientes de publicar y que se publican correctamente todos aquellos seleccionados. Si no hay ninguno pendiente, se muestra el correspondiente mensaje y si no introducimos ninguno de los posibles también nos lo indicaría.

La segunda parte de las pruebas realizadas, se corresponde con la parte de accesos múltiples a la plataforma por parte de diferentes usuarios, con diferentes perfiles y de forma simultánea para comprobar el comportamiento de la aplicación.

- Se ha comprobado que se realizan correctamente la inserción y validación de objetos virtuales de forma prácticamente simultánea. Con esto nos referimos, por ejemplo, al correcto comportamiento de la aplicación nada más crear un nuevo objeto en la plataforma por parte del registrador, el gestor puede consultarlo y comprobar su estructura.

- También de esta forma, se comprueba que las notas creadas por el gestor podrán consultarse a continuación por el registrador que haya creado el objeto al que se le asocia la nota, y poder comprobar los fallos estructurales de su objeto u objetos virtuales.

- Comprobamos también, que la aplicación responde correctamente al interactuar con ella distintos tipos de perfilde usuario, con sus correspondientes opciones, así como usuarios del mismo perfil que accedan simultáneamente a la aplicación. Se ha comprobado que no hay solapamiento ni pérdida de información de cada uno de los usuarios, por tanto vemos que el pool de conexiones funciona correctamente y proporciona conexiones diferentes para múltiples usuarios.

- Estas pruebas de múltiples accesos simultáneos para comprobar la seguridad, deben realizarse abriendo diferentes exploradores o ventanas de explorador para asegurarnos de que son diferentes contextos para que el pool de conexiones funcione correctamente. Si lo probamos con el mismo explorador en diferentes pestañas, será como si accediéramos desde el mismo y se perderán los datos del usuario que estemos utilizando en una pestaña, si introducimos uno nuevo en otra. Esto no ocurre si probamos con diferentes ventanas del browser, ya que los contextos serán distintos y el pool funcionará correctamente.

En resumen, mediante los diferentes tipos de pruebas realizadas, que han sido repetidas varias veces para comprobar que todo el funcionamiento del prototipo se produce exactamente como se espera y como se define en según los requisitos, podemos afirmar que el prototipo ofrece la funcionalidad básica definida para el mismo, obteniendo resultados fiables aunque limitados a su funcionalidad.

Por supuesto hay pruebas que no se han contemplado ni realizado, debido a la falta de tiempo o por no abarcar esos puntos el prototipo, que mejorarían el comportamiento de la plataforma y que

principalmente podemos destacar sobre todos ellos, la falta de un módulo que asegure la seguridad de la aplicación pero en el que no hemos puesto especial énfasis. Esto se contemplará en posibles evoluciones y mejoras de la aplicación.

2.2.7. Análisis de Resultados y Optimización de la Arquitectura PRG-eLearn

El análisis de resultados obtenido a partir del desarrollo de la aplicación, nos ofrece un carácter optimista con respecto a la misma, ya que se cumplen las funcionalidades recogidas en la fase de análisis y diseño del prototipo y, a pesar de que no es completo y le faltarían varias funcionalidades de la arquitectura total que se piensa que puede ofrecer una Plataforma de Registro y Gestión descrita en la primera parte del análisis y diseño, sí que implementa las principales funciones de aquella.

Las pruebas reflejan la fiabilidad del prototipo para las funcionalidades que recoge y a partir de ellas se intuyen posibles optimizaciones de la arquitectura.

En primer lugar, se comprueba que el repositorio ha sido implementado de forma funcional y rápida, pero obviamente sería mejor utilizar como repositorio un repositorio UDDI registrando objetos virtuales implementados como web services para ofrecer acceso global de estos objetos a los diferentes futuros usuarios de los objetos y de la plataforma.

Una optimización de la arquitectura también importante, sería definir y desarrollar un módulo de seguridad de la plataforma, para evitar comportamientos anómalos en ella como accesos a la plataforma desde distintos equipos con el mismo usuario, o accesos directos a alguna de las pantallas, etc.

3. Líneas de Evolución de PRG-eLearn

Por formar parte del amplio mundo del e-learning y de la normalización en el uso de sus contenidos, las posibles evoluciones del sistema PRG-eLearn quedan abiertas a ampliaciones y mejoras. Enfocaremos este apartado de dos puntos de vista: las posibilidades de mejora del propio sistema y las posibilidades de ampliación para su integración en entornos de e-learning.

Partiendo de lo ya desarrollado en este proyecto, quedaría la opción de mejorarlo en diversos aspectos. Del lado visual, el prototipo se desarrolló teniendo presente en todo momento que su objetivo era presentar funcionalidades básicas de forma clara. Se han dejado de lado las posibilidades de mejorar su calidad visual para centrarse en dotar de opciones que den buena idea de dónde queremos llegar. En este aspecto podría mejorarse su manejo para hacerlo más cómodo al usuario a la vez que agradable a la vista, con formas más sofisticadas de formar los árboles de contenidos, bien dando más protagonismo al ratón para arrastrar nuevos componentes al árbol, o también manteniendo ayudas contextuales en todo momento para dar soporte al usuario registrador o gestor.

Podría realizarse también, el desarrollo de un modelo de seguridad más funcional que las simples comprobaciones que realizamos ahora mismo en el prototipo que aseguren un comportamiento correcto de la aplicación, y que están más orientados para una verdadera aplicación ofrecida a través de la Web a los posibles múltiples usuarios. También sería recomendable desarrollar un módulo encargado de la gestión de los usuarios que vayan acceder a la plataforma de forma visual con altas bajas, etc. y no mediante acceso directo a la base de datos por parte del administrador o gestor de la misma.

Por otra parte, el trabajo que hace el prototipo para no trabaja verdaderamente la recuperación de contenidos con Web Services, sino que los almacena en un espacio de la base de datos con el objeto de simplemente mostrar la forma de su funcionamiento de cara al usuario. Aquí entraría la adición de un repositorio propio para el trabajo con Web Services, como puede ser UDDI, en el que sí podrían recuperarse contenidos con este tipo de servicios.

En el plano de la integración con otros sistemas, quedaría siempre la posibilidad de poder ampliar la plataforma para aprovechar los contenidos que genera y que éstos puedan ser utilizados por usuarios finales. Estos usuarios podrían ser administrados quizá por alguno de los perfiles que maneja ya la herramienta o aparecería un nuevo tipo de usuario administrador que hiciera lo propio con los tipos de usuario que manejan la herramienta. Para poder recuperar contenidos creados con la propia herramienta, se haría necesaria su ampliación o cuanto menos su integración a plataformas ya desarrolladas que la utilizaran como una funcionalidad más para recuperar contenidos de un repositorio en el proceso de construcción de sus propios contenidos.

4. Conclusiones

PRG-eLearn en su conjunto aporta una idea quizá poco explorada del manejo de contenidos de e-learning en cuanto a la necesidad de un repositorio no específicamente relacionado con una plataforma concreta, sino más bien disponible en teoría para poder ser utilizado con cualquier plataforma cuyos contenidos observen el estándar IMS.

Por otra parte, esta propuesta es consecuencia de la clara necesidad de normalización de los contenidos de aprendizaje, para conseguir el objetivo de poder compartir estos contenidos sin necesidad de grandes transformaciones ni de repetir trabajo ya hecho, con el consiguiente ahorro de esfuerzo a la hora de construir estos contenidos.

A modo ya más personal, la experiencia del proyecto nos ha enseñado a valorar mucho más cada uno de los pasos que deben darse para llegar a buen puerto en un proyecto de proporciones ya apreciables en comparación con lo que habíamos hecho hasta ahora en la Universidad.

La planificación de cada tarea del proceso de desarrollo es de una importancia crítica, y es posible que nuestra inexperiencia nos haya llevado no tan lejos como podríamos desear, y pudiera haberse conseguido con un grupo más experto. En otro orden de cosas, el trabajo con e-learning es siempre bastante interesante, teniendo en cuenta el entorno universitario donde hemos desarrollado este proyecto, y nos ha permitido conocer un poco más esta disciplina que tiene tanto futuro.

No podemos olvidar tampoco, los nuevos conocimientos adquiridos y la experiencia en los mismos, como es el hecho de haber aprendido bastante en el terreno de la programación con respecto a trabajar con Struts, acceso a bases de datos con pool de conexiones y demás, y conocimientos mayores acerca de Web Services y repositorios UDDI aunque simplemente en con respecto a la parte teórica de esta tecnología.

5. Bibliografía

CORDRA:

- http://cordra.net/information/publications/

Chasqui:

- Alfredo Fernández-Valmayor, Mercedes Guinea, Mariano Jiménez, Antonio Navarro, Antonio Sarasa: "*Objetos virtuales: una aproximación a la construcción de objetos de aprendizaje en Arqueología*"

OKI:

- http://okiproject.org/filemgmt-data/files/ArchitectureProcess.pdf
- http://www.ontapsolutions.com/articles/oki/20050421-oki_impllayering.shtml

Reload:

- http://www.reload.ac.uk/

WebCT:

- http://www.webct.com/

Estándar IMS

- IMS Content Packaging v 1.1.4
- IMS Learning Resource Meta-Data v1.2.1
- IMS Content Packaging XML Binding v 1.1.4
- IMS Learning Resource Meta-Data XML Binding 1.2.1

Base de Datos y Pool de Conexiones:

- http://www.mysql.com/
- http://tomcat.apache.org/tomcat-5.0-doc/jndi-datasource-examples-howto.html

FrameWork Struts:

- http://struts.apache.org/
- http://www.netbeans.org/

Repositorio UDDI y Web Services:

- http://ws.apache.org/juddi/
- http://www-128.ibm.com/developerworks/webservices

6. Anexo: Código de programación más relevante

En este apartado vamos a mostrar el código de las principales clases del Prototipo desarrollado, sin entrar a mostrar las diferentes clases *Action* ni *ActionForm* que son muchas y tienen un comportamiento bastante similar. Estas clases podrán comprobarse una vez entregado todo el código del prototipo de la Plataforma.

A continuación, mostramos las clases principales encargadas de implementar la lógica de negocio que recoge la funcionalidad del Sistema y que se intuye que tienen más relevancia que las demás que componen el conjunto del Proyecto: *TratamientoBD, PoolBD, ParserDocXML, PresentacionAltaObjeto.*

TratamientoBD.java

```
package tratamientoXML_BD;

import general.*;
import java.sql.*;
import java.util.ArrayList;

/* Clase encargada de implementar los métodos relacionados con el acceso
 * a base de datos: consultas, inserciones, eliminaciones, actualizaciones...
 */
public class TratamientoBD {


    public TratamientoBD() {
    }

    /* Procedimiento que inserta el elemento raíza del objeto virtual
     * y devuelve el identificador que se le asocia de forma automática.
     */
    public static int iniciarObjetoVirtual(String nombreNuevoObjeto) throws Exception {
        ElementoXML elemPrincipal = new ElementoXML();
        Statement stmt = null;
        ResultSet rs = null;
        int id = -1;
        Connection con = null;

        try {
            con = PoolBD.getConexion();

            stmt = con.createStatement(java.sql.ResultSet.TYPE_FORWARD_ONLY,
                    java.sql.ResultSet.CONCUR_UPDATABLE);

            int idObjeto = insertarNuevoObjetoVirtual(nombreNuevoObjeto);

            stmt.executeUpdate(
```

```
                        "INSERT INTO elementos_objeto(nombre_etiqueta, idObjeto, dependencia,
nivel) " + " VALUES('lom', "+ idObjeto +", 0, 1)",  Statement.RETURN_GENERATED_KEYS);

                    //Nos devuelve el ultimo indice que acabamos de insertar en la fila
                    // autoincrementable
                    rs = stmt.getGeneratedKeys();

                    if (rs.next()) {
                        id = rs.getInt(1);
                    }

                } catch( Exception e) {
                    try {
                        con.rollback();
                    } catch (SQLException ex) {
                        ex.printStackTrace();
                    }
                    e.printStackTrace();
                } finally {
                    try {
                        if (rs != null)
                            rs.close();
                        if (con != null)
                            con.close();
                    } catch (Exception ex) {
                        throw new Exception(ex.getMessage());
                    }
                }

                return id;
            }

            /**
             * Inserción de un nuevo objeto virtual en la base de datos.
             */
            public static int insertarNuevoObjetoVirtual(String nombreObjeto) throws Exception {
                ElementoXML elemPrincipal = new ElementoXML();
                Statement stmt = null;
                ResultSet rs = null;
                int id = -1;
                Connection con = null;

                try {
                    con = PoolBD.getConexion();

                    stmt = con.createStatement(java.sql.ResultSet.TYPE_FORWARD_ONLY,
                            java.sql.ResultSet.CONCUR_UPDATABLE);

                    stmt.executeUpdate("INSERT INTO objetos_virtuales(Nombre_Objeto) " +
                            " VALUES('" + nombreObjeto + "')",  Statement.RETURN_GENERATED_KEYS);

                    rs = stmt.getGeneratedKeys();
```

```
                if (rs.next()) {
                    id = rs.getInt(1);
                }

            } catch(Exception e) {
                try {
                    con.rollback();
                } catch (SQLException ex) {
                    ex.printStackTrace();
                }
                e.printStackTrace();
            } finally {
                try {
                    if (rs != null)
                        rs.close();
                    if (con != null)
                        con.close();
                } catch (Exception ex) {
                    throw new Exception(ex.getMessage());
                }
            }

            return id;
        }

        /**
         * Inserta un nodo con sus datos en la tabla de elementos del objeto en BD.
         */
        public static int insertarNodoEnBD(String etiqueta, String atributoEtiqueta, String
valorAtributo,
                String valor, int dependencia, int nivel, int idObjetoAsociado) throws Exception {

            int identificador = -1;
            Statement stmt = null;
            ResultSet rs = null;
            Connection con = null;

            if (atributoEtiqueta == null) atributoEtiqueta = "";
            if (valor == null) valor = "";
            if (valorAtributo == null) valorAtributo = "";

            try {
                con = PoolBD.getConexion();
                stmt = con.createStatement(java.sql.ResultSet.TYPE_FORWARD_ONLY,
                        java.sql.ResultSet.CONCUR_UPDATABLE);

                String consulta = "INSERT INTO elementos_objeto (nombre_etiqueta,
atributoEtiqueta, valorAtributo, valor, idObjeto, nivel, dependencia) " +
"VALUES('" + etiqueta + "','" + atributoEtiqueta + "','" + valorAtributo + "','" + valor + "','"
+ idObjetoAsociado + "','" + nivel + "','" + String.valueOf(dependencia) + "')";
                stmt.executeUpdate(consulta, Statement.RETURN_GENERATED_KEYS);
```

```
            rs = stmt.getGeneratedKeys();
            if (rs.next()) {
                identificador = rs.getInt(1);
            }

        } catch( Exception e ) {
            try {
                con.rollback();
            } catch (SQLException ex) {
                ex.printStackTrace();
            }
            e.printStackTrace();
        } finally {
            try {
                if (rs != null)
                    rs.close();
                if (con != null)
                    con.close();
            } catch (Exception ex) {
                throw new Exception(ex.getMessage());
            }
        }

        return identificador;
    }

    public static void deleteNodo(int id) throws Exception {

        Statement stmt = null;
        ResultSet rs = null;
        Connection con = null;

        try {
            con = PoolBD.getConexion();
            stmt = con.createStatement(java.sql.ResultSet.TYPE_FORWARD_ONLY,
                    java.sql.ResultSet.CONCUR_UPDATABLE);

            String consulta = "DELETE FROM elementos_objeto WHERE id=" + id;

            stmt.executeUpdate(consulta);
        } catch( Exception e ) {
            try {
                con.rollback();
            } catch (SQLException ex) {
                ex.printStackTrace();
            }
            e.printStackTrace();
        } finally {
            try {
                if (rs != null)
                    rs.close();
                if (con != null)
```

```
                con.close();
            } catch (Exception ex) {
                throw new Exception(ex.getMessage());
            }
        }
    }

    /* Elimina los elementos de la tabla elementos_objeto.
     */
    protected static void vaciaTablaElementosXML() throws Exception {
        Statement stmt = null;
        Connection con = null;

        try {
            con = PoolBD.getConexion();

            stmt = con.createStatement();
            stmt.executeUpdate("TRUNCATE TABLE elementos_objeto");
            stmt = con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                    ResultSet.CONCUR_READ_ONLY);

        } catch( Exception e ) {
            try {
                con.rollback();
            } catch (SQLException ex) {
                ex.printStackTrace();
            }
            e.printStackTrace();
        } finally {
            try {
                if (con !=null) {
                    con.close();
                }
            } catch (Exception ex) {
                throw new Exception(ex.getMessage());
            }
        }
    }

    /* Obtiene el padre de un elemento del árbol buscando por su identificador
     * en la tabla de elementos de objeto de BD.
     */
    public static ElementoXML getPadre(int id) throws Exception {
        ElementoXML padre = null;
        ResultSet rs = null;
        Statement stmt = null;
        Connection con = null;

        try {
            con = PoolBD.getConexion();

            stmt = con.createStatement();
```

```
                String consulta = "SELECT e2.* FROM elementos_objeto as e1, elementos_objeto as e2
" + " WHERE e1.id = " + id + " and e1.dependencia=e2.id ";

                rs = stmt.executeQuery(consulta);

                if (rs.next()){
                    padre = new ElementoXML();

                    padre.setIdElemento(rs.getInt("id"));
                    padre.setAtributo(rs.getString("atributoEtiqueta"));
                    padre.setNombre(rs.getString("nombre_etiqueta"));
                    padre.setValor(rs.getString("valor"));
                    padre.setValorAtributo(rs.getString("valorAtributo"));
                    padre.setIdObjetoAsociado(rs.getInt("idObjeto"));
                    padre.setDependencia(rs.getInt("dependencia"));
                    padre.setNivel(rs.getInt("nivel"));
                }
            } catch( Exception e ) {
                try {
                    con.rollback();
                } catch (SQLException ex) {
                    ex.printStackTrace();
                }
                e.printStackTrace();
            } finally {
                try {
                    if (rs != null)
                        rs.close();
                    if (con != null)
                        con.close();
                } catch (Exception ex) {
                    throw new Exception(ex.getMessage());
                }
            }

            return padre;
        }

        /**
         * Obtiene el elemento del árbol buscando por su identificador
         * en la tabla de elementos de objeto de BD.
         */
        public static ElementoXML getElem(int id) throws Exception {
            ElementoXML elem = null;
            ResultSet rs = null;
            Statement stmt = null;
            Connection con = null;

            try {
                con = PoolBD.getConexion();

                stmt = con.createStatement();
```

```
        rs = stmt.executeQuery("SELECT * FROM elementos_objeto " +
                "WHERE id='"+String.valueOf(id)+"' ORDER BY id");

        if (rs.next()){
            elem = new ElementoXML();

            elem.setIdElemento(rs.getInt("id"));
            elem.setAtributo(rs.getString("atributoEtiqueta"));
            elem.setNombre(rs.getString("nombre_etiqueta"));
            elem.setValor(rs.getString("valor"));
            elem.setValorAtributo(rs.getString("valorAtributo"));
            elem.setIdObjetoAsociado(rs.getInt("idObjeto"));
            elem.setNivel(rs.getInt("nivel"));
            elem.setDependencia(rs.getInt("dependencia"));
        }

    } catch( Exception e ) {
        try {
            con.rollback();
        } catch (SQLException ex) {
            ex.printStackTrace();
        }
        e.printStackTrace();
    } finally {
        try {
            if (rs != null)
                rs.close();
            if (con != null)
                con.close();
        } catch (Exception ex) {
            throw new Exception(ex.getMessage());
        }
    }

    return elem;
}

/**
 * Devuelve una lista con los hermanos de un determinado elemento (idHermano).
 * Si no tiene hermanos se delvoverá una lista de 0 elementos.
 */
public static ArrayList getHermanos(int idHermano) throws Exception {
    ElementoXML elemAct;
    ArrayList lista = new ArrayList();
    ResultSet rs = null;
    Statement stmt = null;
    Connection con = null;

    try {
        con = PoolBD.getConexion();

        stmt = con.createStatement();
```

```
                String consulta = "SELECT e2.* FROM elementos_objeto as e1, elementos_objeto as e2
" + "WHERE e1.id=" + idHermano + " AND e1.dependencia=e2.dependencia";
                rs = stmt.executeQuery(consulta);

                while(rs.next()){
                    int idElemento = rs.getInt("id");

                    if (idElemento != idHermano){
                        elemAct = new ElementoXML();

                        elemAct.setIdElemento(idElemento);
                        elemAct.setAtributo(rs.getString("atributoEtiqueta"));
                        elemAct.setNombre(rs.getString("nombre_etiqueta"));
                        elemAct.setValor(rs.getString("valor"));
                        elemAct.setValorAtributo(rs.getString("valorAtributo"));
                        elemAct.setIdObjetoAsociado(rs.getInt("idObjeto"));
                        elemAct.setNivel(rs.getInt("nivel"));
                        elemAct.setDependencia(rs.getInt("dependencia"));

                        lista.add(elemAct);
                    }
                }

            } catch( Exception e ) {
                try {
                    con.rollback();
                } catch (SQLException ex) {
                    ex.printStackTrace();
                }
                e.printStackTrace();
            } finally {
                try {
                    if (rs != null)
                        rs.close();
                    if (con != null)
                        con.close();
                } catch (Exception ex) {
                    throw new Exception(ex.getMessage());
                }
            }

            return lista;
        }

        /**
         * Obtiene los hijos que tiene un elemento del árbol buscando
         * por su identificador en BD.
         */
        public static ArrayList getHijos(int idPadre) throws Exception {
            ElementoXML elemAct;
            ArrayList lista = new ArrayList();
```

```
    int idHijo;
    ResultSet rs = null;
    Statement stmt = null;
    Connection con = null;

    try {
        con = PoolBD.getConexion();

        stmt = con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                ResultSet.CONCUR_READ_ONLY);

        rs = stmt.executeQuery("SELECT * FROM elementos_objeto " +
                "WHERE dependencia='"+idPadre+"' ORDER BY id");

        while(rs.next()){
            elemAct = new ElementoXML();

            elemAct.setIdElemento(rs.getInt("id"));
            elemAct.setAtributo(rs.getString("atributoEtiqueta"));
            elemAct.setNombre(rs.getString("nombre_etiqueta"));
            elemAct.setValor(rs.getString("valor"));
            elemAct.setValorAtributo(rs.getString("valorAtributo"));
            elemAct.setIdObjetoAsociado(rs.getInt("idObjeto"));
            elemAct.setNivel(rs.getInt("nivel"));
            elemAct.setDependencia(rs.getInt("dependencia"));

            lista.add(elemAct);
        }

    } catch( Exception e ) {
        try {
            con.rollback();
        } catch (SQLException ex) {
            ex.printStackTrace();
        }
        e.printStackTrace();
    } finally {
        try {
            if (rs != null)
                rs.close();
            if (con != null)
                con.close();
        } catch (Exception ex) {
            throw new Exception(ex.getMessage());
        }
    }

    return lista;
}

/** AÑADE HIJO **/
```

```
            public static int anyadeHijo(String etiqueta, int idPadre) throws Exception {
                int idHijo = -1;
                ElementoXML elem;
                try {
                    elem = getPadre(idPadre);

                    if (elem!=null){
                        int nivel = elem.getNivel()+1;
                        int idObjetoAsociado = elem.getIdObjetoAsociado();

                        idHijo = insertarNodoEnBD(etiqueta, "NULL", "NULL", "NULL", idPadre, nivel,
idObjetoAsociado);
                    }
                }catch (Exception ex) {
                    throw new Exception(ex.getMessage());
                }

                return idHijo;
            }

            public static int anyadeHijo(String etiqueta, String atributoEtiqueta, String
valorAtributo, String valor, int idPadre) throws Exception {

                int idHijo = -1;
                ElementoXML elem;
                try {
                    elem = getPadre(idPadre);

                    if (elem!=null){
                        int nivel = elem.getNivel()+1;
                        int idObjetoAsociado = elem.getIdObjetoAsociado();

                        idHijo = insertarNodoEnBD(etiqueta, atributoEtiqueta, valorAtributo, valor,
idPadre, nivel, idObjetoAsociado);
                    }
                } catch (Exception ex) {
                    throw new Exception(ex.getMessage());
                }


                return idHijo;
            }

            /**
             * Añade un elemento hijo a un nodo del árbol va formando los elementos
             * del objeto virtual que se está creando.
             */
            public static int anyadeHijo(String etiqueta, String valor, int idPadre) throws Exception
{
                int idHijo = -1;
                ElementoXML elem;
                try {
```

```
                elem = getPadre(idPadre);

                if (elem!=null){
                    int nivel = elem.getNivel()+1;
                    int idObjetoAsociado = elem.getIdObjetoAsociado();

                    idHijo = insertarNodoEnBD(etiqueta, "NULL", "NULL", valor, idPadre, nivel,
idObjetoAsociado);
                }
            } catch (Exception ex) {
                throw new Exception(ex.getMessage());
            }


            return idHijo;
        }


        /**
         * Añade un elemento hijo a un nodo del árbol va formando los elementos
         * del objeto virtual que se está creando.
         */
        public static int anyadeHijo(ElementoXML elemento, int idPadre) throws Exception {
            int idHijo = -1;
            try {

                idHijo = insertarNodoEnBD(elemento.getNombre(), elemento.getAtributo(),
elemento.getValorAtributo(), elemento.getValor(), idPadre, elemento.getNivel(),
elemento.getIdObjetoAsociado());
            } catch (Exception ex) {
                throw new Exception(ex.getMessage());
            }

            return idHijo;
        }

        /**
         * Obtiene los posibles hijos que puede tener un determinado elemento
         * según el estándar IMS consultando en la BD.
         */
        public static ArrayList getPosiblesHijos(int id) throws Exception {
            ArrayList lista = new ArrayList();
            ResultSet rs = null;
            java.sql.Statement stmt = null;
            String etq_hijo = null;
            Connection con = null;

            try {
                con = PoolBD.getConexion();

                stmt = con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                        ResultSet.CONCUR_READ_ONLY);
```

```
        String consulta = "SELECT e1.etiqueta_hijo FROM posibles_hijos as e1,  " +
                "elementos_objeto as e2 WHERE e1.etiqueta = e2.nombre_etiqueta " +
                "AND e2.id='" + id + "'";

        rs = stmt.executeQuery(consulta);

        while(rs.next()){
            etq_hijo = rs.getString("etiqueta_hijo");
            lista.add(etq_hijo);
        }

    } catch (Exception ex) {
        try {
            con.rollback();
        } catch (SQLException e) {
            e.printStackTrace();
        }
        ex.printStackTrace();
    }

    finally {
        try {
            if (stmt != null)
                stmt.close();
            if (con != null)
                con.close();
        } catch (Exception ex) {
            throw new Exception(ex.getMessage());
        }
    }

    return lista;
}

/**
 * Obtiene los elementos de la tabla vocabulario de BD.
 */
public static ArrayList getElementosVocabulario() throws Exception {
    ArrayList lista = new ArrayList();
    ResultSet rs = null;
    Statement stmt = null;
    String elemento = null;
    Connection con = null;

    try {
        con = PoolBD.getConexion();

        stmt = con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                ResultSet.CONCUR_READ_ONLY);

        rs = stmt.executeQuery("SELECT * FROM vocabulario;");
```

```
        while(rs.next()){
            elemento = rs.getString("elemento");
            lista.add(elemento);
        }

    }  catch (Exception ex) {
        try {
            con.rollback();
        } catch (SQLException e) {
            e.printStackTrace();
        }
        ex.printStackTrace();
    }

    finally {
        try {
            if (stmt != null)
                stmt.close();
            if (con != null)
                con.close();
        } catch (Exception ex) {
            throw new Exception(ex.getMessage());
        }
    }

    return lista;
}

/**
 * Obtiene los objetos pendientes de validar la estructura.
 */
public static ArrayList consultaObjetosPendientesValidarEstructura() throws Exception{
    ArrayList objetosPendientes = null;
    ResultSet rs = null;
    PreparedStatement ps = null;
    Connection con = null;

    try {
        con = PoolBD.getConexion();

        ps = con.prepareStatement("SELECT * FROM objetos_virtuales " +
                "WHERE Validado_Estructura='0' ORDER BY ID_Objeto");
        rs = ps.executeQuery();

        objetosPendientes = new ArrayList();

        while(rs.next()){
            ClaseObjetoVirtual ob = new ClaseObjetoVirtual();
            ob.setIdObjeto(rs.getInt("ID_Objeto"));
            ob.setNombre(rs.getString("Nombre_Objeto"));
            ob.setValidado(rs.getBoolean("Validado_Estructura"));
```

```
                ob.setPublicado(rs.getBoolean("Publicado"));

                objetosPendientes.add(ob);
            }
        } catch (Exception ex) {
            try {
                con.rollback();
            } catch (SQLException e) {
                e.printStackTrace();
            }
            ex.printStackTrace();
        }

        finally {
            try {
                if (ps != null)
                    ps.close();
                if (con != null)
                    con.close();
            } catch (Exception ex) {
                throw new Exception(ex.getMessage());
            }
        }

        return objetosPendientes;
    }

    /**
     * Obtiene los objetos virtuales creados por un determinado usuario.
     * Devolverá una lista con los que tenga o vacía si no ha creado ninguno aún.
     */
    public static ArrayList consultaOVPertenecientesUsuario(String idUsuario) throws
Exception{
        ArrayList listaOV = null;
        ResultSet rs = null;
        PreparedStatement ps = null;
        Connection con = null;

        try {
            con = PoolBD.getConexion();

            ps = con.prepareStatement("SELECT DISTINCT e1.IdObjeto , e2.Nombre_Objeto, " +
                    "e2.Validado_Estructura, e2.Publicado " +
                    "FROM objeto_asociado_usuario as e1, objetos_virtuales as e2 " +
                    "WHERE e1.IdObjeto=e2.ID_Objeto AND e1.IdUsuario='" + idUsuario + "' " +
                    "ORDER BY e1.IdObjeto");
            rs = ps.executeQuery();

            listaOV = new ArrayList();

            while(rs.next()){
                ClaseObjetoVirtual ob = new ClaseObjetoVirtual();
```

```
            ob.setIdObjeto(rs.getInt("IdObjeto"));
            ob.setNombre(rs.getString("Nombre_Objeto"));
            ob.setPublicado(rs.getBoolean("Publicado"));
            ob.setValidado(rs.getBoolean("Validado_Estructura"));

            listaOV.add(ob);
        }
    } catch (Exception ex) {
        try {
            con.rollback();
        } catch (SQLException e) {
            e.printStackTrace();
        }
        ex.printStackTrace();
    }

    finally {
        try {
            if (ps != null)
                ps.close();
            if (con != null)
                con.close();
        } catch (Exception ex) {
            throw new Exception(ex.getMessage());
        }
    }
    return listaOV;
}

/**
 * Obtiene los objetos virtuales validados que aún no hayan sido publicados.
 * Devolverá una lista con los que haya sin publicar o vacía e.o.c.
 */
public static ArrayList consultaObjetosValidadosNoPublicados() throws Exception{
    ArrayList objetosValNoPublicados = null;
    ResultSet rs = null;
    PreparedStatement ps = null;
    Connection con = null;

    try {
        con = PoolBD.getConexion();

        ps = con.prepareStatement("SELECT * FROM objetos_virtuales " +
                "WHERE Validado_Estructura='1' and Publicado='0' " +
                "ORDER BY ID_Objeto");
        rs = ps.executeQuery();

        objetosValNoPublicados = new ArrayList();

        while(rs.next()){
            ClaseObjetoVirtual ob = new ClaseObjetoVirtual();
            ob.setIdObjeto(rs.getInt("ID_Objeto"));
```

```
                ob.setNombre(rs.getString("Nombre_Objeto"));
                ob.setValidado(rs.getBoolean("Validado_Estructura"));
                ob.setPublicado(rs.getBoolean("Publicado"));

                objetosValNoPublicados.add(ob);
            }

        } catch (Exception ex) {
            try {
                con.rollback();
            } catch (SQLException e) {
                e.printStackTrace();
            }
            ex.printStackTrace();
        } finally {
            try {
                if (ps != null)
                    ps.close();
                if (con != null)
                    con.close();
            } catch (Exception ex) {
                throw new Exception(ex.getMessage());
            }
        }

        return objetosValNoPublicados;
    }

    /**
     * Actualiza los objetos que hayan sido publicados actualizando el campo correspondiente
     * en la tabla de objetos virtuales.
     * Nos pasan una lista con los objetos publicados como parámetro o vacía e.o.c.
     */
    public static int actualizaObjetosPublicados(ArrayList lista) throws Exception{
        PreparedStatement ps = null;
        int resultado = 0;
        Connection con = null;

        //Comprobamos que haya objetos en la lista
        if (lista.size()>0) {

            try {
                con = PoolBD.getConexion();

                //Hacemos el UPDATE del primer elemento que haya seguro y miramos si hubiera
más
                String consultaSQL = "UPDATE objetos_virtuales SET Publicado='1' " +
                        "WHERE ID_Objeto='" + (String) lista.get(0) + "'";

                for (int i=1;i<lista.size();i++){
                    consultaSQL = consultaSQL + " OR ID_Objeto='" + (String) lista.get(i) +
"'";
```

```
            }

            ps = con.prepareStatement(consultaSQL);
            resultado = ps.executeUpdate();

        } catch (Exception ex) {
            try {
                con.rollback();
            } catch (SQLException e) {
                e.printStackTrace();
            }
            ex.printStackTrace();
        } finally {
            try {
                if (ps != null)
                    ps.close();
                if (con != null)
                    con.close();
            } catch (Exception ex) {
                throw new Exception(ex.getMessage());
            }
        }

    }
    return resultado;
}

/**
 * Consulta de los elementos del objeto virtual con identificador id.
 * Devuelve una lista de objetos ElementoXML, con los datos de cada elemento.
 */
public static ArrayList consultaElementosObjetoVirtual(int id) throws Exception{
    ArrayList elemObjeto = null;
    ResultSet rs = null;
    PreparedStatement ps = null;
    Connection con = null;

    try {
        con = PoolBD.getConexion();

        ps = con.prepareStatement("SELECT * FROM elementos_objeto " +
                "WHERE idObjeto='" + id + "' ORDER BY id");
        rs = ps.executeQuery();

        elemObjeto = new ArrayList();

        while(rs.next()){
            ElementoXML elem = new ElementoXML();

            elem.setIdElemento(rs.getInt("id"));
            elem.setNombre(rs.getString("nombre_etiqueta"));
            elem.setAtributo(rs.getString("atributoEtiqueta"));
```

```
            elem.setValorAtributo(rs.getString("valorAtributo"));
            elem.setValor(rs.getString("valor"));
            elem.setDependencia(rs.getInt("dependencia"));
            elem.setIdObjetoAsociado(id);
            elem.setNivel(rs.getInt("nivel"));

            elemObjeto.add(elem);
        }
    } catch (Exception ex) {
        try {
            con.rollback();
        } catch (SQLException e) {
            e.printStackTrace();
        }
        ex.printStackTrace();
    } finally {
        try {
            if (ps != null)
                ps.close();
            if (con != null)
                con.close();
        } catch (Exception ex) {
            throw new Exception(ex.getMessage());
        }
    }

    return elemObjeto;
}

/**
 * Actualiza el campo de validación del objeto virtual que se le pasa
 * como objeto validado por el gestor con el valor que se indica.
 */
public static int actualizaObjetoValidados(int id, int valor) throws Exception{
    PreparedStatement ps = null;
    int resultado = 0;
    Connection con = null;

    try {
        con = PoolBD.getConexion();

        //Hacemos el UPDATE del primer elemento que haya seguro y miramos si hubiera más
        String consultaSQL = "UPDATE objetos_virtuales SET Validado_Estructura='" +
                valor + "' WHERE ID_Objeto='" + id + "'";

        ps = con.prepareStatement(consultaSQL);
        resultado = ps.executeUpdate();

    } catch (Exception ex) {
        try {
            con.rollback();
        } catch (SQLException e) {
```

```
            e.printStackTrace();
        }
        ex.printStackTrace();
    } finally {
        try {
            if (ps != null)
                ps.close();
            if (con != null)
                con.close();
        } catch (Exception ex) {
            throw new Exception(ex.getMessage());
        }
    }

    return resultado;
}

/**
 * Actualiza el campo de publicación del objeto virtual con identificador id,
 * con el valor que se indica.
 */
public static int actualizaObjetoPublicado(int id, int valor) throws Exception{
    PreparedStatement ps = null;
    int resultado = 0;
    Connection con = null;

    try {
        con = PoolBD.getConexion();

        //Hacemos el UPDATE del primer elemento que haya seguro y miramos si hubiera más
        String consultaSQL = "UPDATE objetos_virtuales SET Publicado='" +
                valor + "' WHERE ID_Objeto='" + id + "'";

        ps = con.prepareStatement(consultaSQL);
        resultado = ps.executeUpdate();

    } catch (Exception ex) {
        try {
            con.rollback();
        } catch (SQLException e) {
            e.printStackTrace();
        }
        ex.printStackTrace();
    } finally {
        try {
            if (ps != null)
                ps.close();
            if (con != null)
                con.close();
        } catch (Exception ex) {
            throw new Exception(ex.getMessage());
        }
```

```
    }

    return resultado;
}

/**
 * Inserta una nota asociada al objeto virtual que ha introducido el gestor
 * para indicar algún posible error de estructura del objeto.
 */
public static int insertarNotaAsociadaObjeto(int idObjeto, String nota) throws Exception{
    PreparedStatement ps = null;
    int resultado = 0;
    Connection con = null;

    try {
        con = PoolBD.getConexion();

        //Hacemos el UPDATE del primer elemento que haya seguro y miramos si hubiera más
        String consultaSQL = "INSERT INTO notas_objeto(IdObjetoAsociado, Descripcion) " +
                "VALUES ('" + idObjeto + "','" + nota + "')";

        ps = con.prepareStatement(consultaSQL);
        resultado = ps.executeUpdate();

    } catch (Exception ex) {
        try {
            con.rollback();
        } catch (SQLException e) {
            e.printStackTrace();
        }
        ex.printStackTrace();
    } finally {
        try {
            if (ps != null)
                ps.close();
            if (con != null)
                con.close();
        } catch (Exception ex) {
            throw new Exception(ex.getMessage());
        }
    }

    return resultado;
}

/**
 * Devuelve la lista de notas asociada a un objeto.
 * Si no hay ninguna nota, se devolverá una lista de tamaño 0.
 */
public static ArrayList consultaNotasAsociadasObjeto(int id) throws Exception{
    ArrayList listaNotas = null;
    ResultSet rs = null;
```

```
    PreparedStatement ps = null;
    Connection con = null;

    try {
        con = PoolBD.getConexion();

        ps = con.prepareStatement("SELECT * FROM notas_objeto " +
                "WHERE idObjetoAsociado='" + id + "' ORDER BY IDNotas");
        rs = ps.executeQuery();

        listaNotas = new ArrayList();

        while(rs.next()){
            String nota = rs.getString("Descripcion");
            listaNotas.add(nota);
        }

    } catch (Exception ex) {
        try {
            con.rollback();
        } catch (SQLException e) {
            e.printStackTrace();
        }
        ex.printStackTrace();
    } finally {
        try {
            if (ps != null)
                ps.close();
            if (con != null)
                con.close();
        } catch (Exception ex) {
            throw new Exception(ex.getMessage());
        }
    }

    return listaNotas;
}

/**
 * Eliminar todos los elementos relacionados con el objeto virtual que se le pasa
 * para no saturar la tabla de elementos, en la creación de objetos.
 */
public static void eliminarElementosObjeto(int idObjeto) throws Exception{
    PreparedStatement ps = null;
    int resultado = 0;
    Connection con = null;

    try {
        con = PoolBD.getConexion();

        String consultaSQL = "DELETE FROM elementos_objeto " +
                "WHERE idObjeto='" + idObjeto + "'";
```

```
            ps = con.prepareStatement(consultaSQL);
            resultado = ps.executeUpdate();

        } catch (Exception ex) {
            try {
                con.rollback();
            } catch (SQLException e) {
                e.printStackTrace();
            }
            ex.printStackTrace();
        } finally {
            try {
                if (ps != null)
                    ps.close();
                if (con != null)
                    con.close();
            } catch (Exception ex) {
                throw new Exception(ex.getMessage());
            }
        }
    }

    /**
     * Elimina el objeto virtual asociado a un usuario de la tabla de
     * objetos virtuales.
     */
    public static void eliminarObjetoVirtual(int idObjeto) throws Exception{
        PreparedStatement ps = null;
        int resultado = 0;
        Connection con = null;

        try {
            con = PoolBD.getConexion();

            String consultaSQL = "DELETE FROM objetos_virtuales " +
                    "WHERE ID_Objeto='" + idObjeto + "'";

            ps = con.prepareStatement(consultaSQL);
            resultado = ps.executeUpdate();

        } catch (Exception ex) {
            try {
                con.rollback();
            } catch (SQLException e) {
                e.printStackTrace();
            }
            ex.printStackTrace();
        } finally {
            try {
                if (ps != null)
                    ps.close();
```

```
            if (con != null)
                con.close();
        } catch (Exception ex) {
            throw new Exception(ex.getMessage());
        }
    }
}

/**
 * Elimina las notas asociadas a un objeto virtual de algún usuario de la BD.
 */
public static void eliminarNotasASociadasObjeto(int idObjeto) throws Exception{
    PreparedStatement ps = null;
    int resultado = 0;
    Connection con = null;

    try {
        con = PoolBD.getConexion();

        String consultaSQL = "DELETE FROM notas_objeto " +
                "WHERE IdObjetoAsociado='" + idObjeto + "'";

        ps = con.prepareStatement(consultaSQL);
        resultado = ps.executeUpdate();

    } catch (Exception ex) {
        try {
            con.rollback();
        } catch (SQLException e) {
            e.printStackTrace();
        }
        ex.printStackTrace();
    } finally {
        try {
            if (ps != null)
                ps.close();
            if (con != null)
                con.close();
        } catch (Exception ex) {
            throw new Exception(ex.getMessage());
        }
    }
}

/**
 * Elimina la referencia de la tabla de objetos asociados a usuario que se haya
 * eliminado de la BD.
 */
public static void eliminarObjetoAsociadoUsuario(int idObjeto) throws Exception{
    PreparedStatement ps = null;
    int resultado = 0;
    Connection con = null;
```

```
            try {
                con = PoolBD.getConexion();

                String consultaSQL = "DELETE FROM objeto_asociado_usuario " +
                        "WHERE IdObjeto='" + idObjeto + "'";

                ps = con.prepareStatement(consultaSQL);
                resultado = ps.executeUpdate();

            } catch (Exception ex) {
                try {
                    con.rollback();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
                ex.printStackTrace();
            } finally {
                try {
                    if (ps != null)
                        ps.close();
                    if (con != null)
                        con.close();
                } catch (Exception ex) {
                    throw new Exception(ex.getMessage());
                }
            }
        }

        /**
         * Consulta el elemento padre de un elemento del árbol a partir del identificador
         * del objeto virtual para asegurarme una correcta construcción del árbol.
         */
        public static int consultaPadreEtiqueta(String nombreEtiqueta, int idObjeto) throws
Exception{
            ResultSet rs = null;
            PreparedStatement ps = null;
            int idPadre = -1;
            Connection con = null;

            try {
                con = PoolBD.getConexion();

                String consulta = "SELECT MAX(id) FROM elementos_objeto WHERE nombre_etiqueta ='"+
                        nombreEtiqueta + "' AND  idObjeto='" + idObjeto + "'";

                ps = con.prepareStatement(consulta);
                rs = ps.executeQuery();

                if (rs.next())
                    idPadre = rs.getInt(1);
```

```
    } catch (Exception ex) {
        try {
            con.rollback();
        } catch (SQLException e) {
            e.printStackTrace();
        }
        ex.printStackTrace();
    } finally {
        try {
            if (ps != null)
                ps.close();
            if (con != null)
                con.close();
        } catch (Exception ex) {
            throw new Exception(ex.getMessage());
        }
    }

    return idPadre;
}

/**
 * Consulta cuál es el elemento raíz de un árbol a partir del identificador
 * del objeto virtual al que pertenezca.
 */
public static int consultaIdRaizObjetoVirtual(int idObjeto) throws Exception{
    ResultSet rs = null;
    PreparedStatement ps = null;
    int idPadre = -1;
    Connection con = null;

    try {
        con = PoolBD.getConexion();

        String consulta = "SELECT MIN(id) FROM elementos_objeto " +
                "WHERE idObjeto='" + idObjeto + "'";

        ps = con.prepareStatement(consulta);
        rs = ps.executeQuery();

        if (rs.next())
            idPadre = rs.getInt(1);

    } catch (Exception ex) {
        try {
            con.rollback();
        } catch (SQLException e) {
            e.printStackTrace();
        }
        ex.printStackTrace();
    } finally {
        try {
```

```
                    if (ps != null)
                        ps.close();
                    if (con != null)
                        con.close();
                } catch (Exception ex) {
                    throw new Exception(ex.getMessage());
                }
            }

            return idPadre;
        }

        /**
         * Inserta el manifest o documento XML generado a partir del árbol en la
         * correspondiente tabla de BD como si fuera un repositorio de manifest
         * de objetos virtuales.
         */
        public static int insertarDocumentoObjetoTablaXML(int idObjeto, String documento) throws
Exception{
            PreparedStatement ps = null;
            int resultado = 0;
            Connection con = null;

            try {
                con = PoolBD.getConexion();
                //Insertamos el documento manifest para el objeto con ese identificador
                String consultaSQL = "INSERT INTO tabla_auxiliar_documentos(IdObjeto,
Documento_Asociado) " + "VALUES ('" + idObjeto + "', '" + documento + "')";

                ps = con.prepareStatement(consultaSQL);
                resultado = ps.executeUpdate();

            } catch (Exception ex) {
                try {
                    con.rollback();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
                ex.printStackTrace();
            } finally {
                try {
                    if (ps != null)
                        ps.close();
                    if (con != null)
                        con.close();
                } catch (Exception ex) {
                    throw new Exception(ex.getMessage());
                }
            }

            return resultado;
        }
```

```
            /**
             * Actualiza el manifest o documento XML generado a partir del árbol en la
             * correspondiente tabla de BD como si fuera un repositorio de manifest
             * de objetos virtuales.
             */
            public static void actualizarDocumentoObjetoTablaXML(int idObjeto, String documento)
throws Exception{
                PreparedStatement ps = null;
                Connection con = null;

                try {
                    con = PoolBD.getConexion();
                    //Hacemos el UPDATE del documento para el objeto con ese identificador
                    String consultaSQL = "UPDATE tabla_auxiliar_documentos SET Documento_Asociado = '"
+ documento + "' WHERE IdObjeto='" + idObjeto + "'";

                    ps = con.prepareStatement(consultaSQL);
                    ps.executeUpdate();

                } catch (Exception ex) {
                    try {
                        con.rollback();
                    } catch (SQLException e) {
                        e.printStackTrace();
                    }
                    ex.printStackTrace();
                } finally {
                    try {
                        if (ps != null)
                            ps.close();
                        if (con != null)
                            con.close();
                    } catch (Exception ex) {
                        throw new Exception(ex.getMessage());
                    }
                }
            }

            /**
             * Inserta el identificador de objeto asociado al usuario que lo ha creado en
             * la tabla correspondiente en BD.
             */
            public static int insertarObjetoAsociadoUsuario(String idUsuario, int idObjeto) throws
Exception{
                PreparedStatement ps = null;
                int resultado = 0;
                Connection con = null;

                try {
                    con = PoolBD.getConexion();
                    //Hacemos el UPDATE del documento para el objeto con ese identificador
```

```
        String consultaSQL = "INSERT INTO objeto_asociado_usuario(IdUsuario, IdObjeto) " +
                "VALUES ('" + idUsuario + "', '" + idObjeto + "')";

        ps = con.prepareStatement(consultaSQL);
        resultado = ps.executeUpdate();

    } catch (Exception ex) {
        try {
            con.rollback();
        } catch (SQLException e) {
            e.printStackTrace();
        }
        ex.printStackTrace();
    } finally {
        try {
            if (ps != null)
                ps.close();
            if (con != null)
                con.close();
        } catch (Exception ex) {
            throw new Exception(ex.getMessage());
        }
    }

    return resultado;
}

/**
 * Obtiene el documento XML o manifest asociado al objeto virtual que se consulta.
 */
public static String consultaDocumentoXMLObjeto(int idObjeto) throws Exception{
    ResultSet rs = null;
    PreparedStatement ps = null;
    String docXML = null;
    Connection con = null;

    try {
        con = PoolBD.getConexion();

        String consulta = "SELECT Documento_Asociado FROM tabla_auxiliar_documentos " +
                "WHERE IdObjeto ='" + idObjeto + "'";

        ps = con.prepareStatement(consulta);
        rs = ps.executeQuery();

        if (rs.next())
            docXML = rs.getString(1);

    } catch (Exception ex) {
        try {
            con.rollback();
        } catch (SQLException e) {
```

```
            e.printStackTrace();
        }
        ex.printStackTrace();
    } finally {
        try {
            if (ps != null)
                ps.close();
            if (con != null)
                con.close();
        } catch (Exception ex) {
            throw new Exception(ex.getMessage());
        }
    }

    return docXML;
}

/**
 * Obtiene una lista con los elementos de la tabla vocabulario relacionados
 * con el elemento vocabulario por el que consultamos.
 * Será una lista vacía en otro caso.
 */
public static ArrayList consultaVocabulario(String seleccion) throws Exception{
    ResultSet rs = null;
    PreparedStatement ps = null;
    ArrayList listaValores = null;
    Connection con = null;

    try {
        con = PoolBD.getConexion();

        String consulta = "SELECT valor FROM vocabulario " +
                "WHERE elemento ='" + seleccion + "'";

        ps = con.prepareStatement(consulta);
        rs = ps.executeQuery();

        listaValores = new ArrayList();
        while (rs.next()){
            listaValores.add(rs.getString("valor"));
        }

    } catch (Exception ex) {
        try {
            con.rollback();
        } catch (SQLException e) {
            e.printStackTrace();
        }
        ex.printStackTrace();
    } finally {
        try {
            if (ps != null)
```

```
                    ps.close();
                if (con != null)
                    con.close();
            } catch (Exception ex) {
                throw new Exception(ex.getMessage());
            }
        }

        return listaValores;
    }

    /**
     * Obtiene una lista con los objetos virtuales asociados a un usuario
     * que tengan alguna nota asociada por el gestor.
     * Será una lista vacía en otro caso.
     */
    public static ArrayList consultaObjetosUsuarioConNotasAsociadas(String idUsuario) throws
Exception{
        ResultSet rs = null;
        PreparedStatement ps = null;
        ArrayList listaObjetosAsociados = new ArrayList();
        Connection con = null;

        try {
            con = PoolBD.getConexion();

            String consulta = "SELECT DISTINCT e1.IdObjetoAsociado, e3.Nombre_Objeto " +
                    "FROM notas_objeto e1, objeto_asociado_usuario as e2, " +
                    "objetos_virtuales as e3  WHERE e2.IdUsuario ='" + idUsuario + "' " +
                    "AND e1.IdObjetoAsociado=e3.ID_Objeto AND e1.IdObjetoAsociado=e2.IdObjeto"
+
                    " ORDER BY e1.IdObjetoAsociado";

            ps = con.prepareStatement(consulta);
            rs = ps.executeQuery();

            while (rs.next()){
                ElementoXML elem = new ElementoXML();

                elem.setIdObjetoAsociado(rs.getInt(1));
                elem.setNombre(rs.getString(2));

                listaObjetosAsociados.add(elem);
            }

        } catch (Exception ex) {
            try {
                con.rollback();
            } catch (SQLException e) {
                e.printStackTrace();
            }
            ex.printStackTrace();
```

```
            } finally {
                try {
                    if (ps != null)
                        ps.close();
                    if (con != null)
                        con.close();
                } catch (Exception ex) {
                    throw new Exception(ex.getMessage());
                }
            }

            return listaObjetosAsociados;
        }

        /* Método encargado de eliminar todos los datos asociados a un objeto
         * virtual de la base de datos, cuando se elimina el objeto.
         */
        public static void eliminarDatosObjetoVirtual(final int idObjeto) throws Exception {
            try {
                //Primero eliminamos el objeto de la tabla de objetos virtuales
                eliminarObjetoVirtual(idObjeto);
                //Eliminamos también sus notas asociadas, si las hubiera
                //así como los elementos en la tabla de elementos.
                eliminarNotasASociadasObjeto(idObjeto);
                eliminarElementosObjeto(idObjeto);
                //Eliminamos por ultimo, el objeto asociado al usuario en
                //la tabla de objeto_asociado_usuario
                eliminarObjetoAsociadoUsuario(idObjeto);
            } catch (Exception ex) {
                throw new Exception(ex.getMessage());
            }

        }

        /**
         * Obtiene el perfil del usuario por el que consultamos a la tabla
    correspodiente de BD a partir de su identificador y su password.
         * Devolvera null e.o.c.
         */
        public static String getTipoUsuario(String user, String pass) throws Exception{
            String perfilUsuario = null;
            ResultSet rs = null;
            PreparedStatement ps = null;
            Connection con = null;

            try {
                con = PoolBD.getConexion();

                ps = con.prepareStatement("SELECT perfil FROM tabautent WHERE idUsuario='" + user
+ "'" + " AND password='" + pass + "'");
                rs = ps.executeQuery();
```

```
                if (rs.next()){
                    perfilUsuario = rs.getString("perfil");
                }

            } catch (Exception ex) {
                try {
                    con.rollback();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
                ex.printStackTrace();
            } finally {
                try {
                    if (ps != null)
                        ps.close();
                    if (con != null)
                        con.close();
                } catch (Exception ex) {
                    throw new Exception(ex.getMessage());
                }
            }
            return perfilUsuario;
        }
    }
```

PoolBD.java

```
package tratamientoXML_BD;

import java.sql.Connection;
import java.util.Properties;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.sql.DataSource;

/* Clase encargada de obtener la conexión a base de datos a través de
 * un pool de conexiones definido anteriormente en el servidor de aplicaciones.
 */
final public class PoolBD {

    private static DataSource ds = null;
    private static String JNDI_CONNECTION = "java:comp/env/jdbc/prgelearn";

    /** Creates a new instance of PoolBD */
    public PoolBD() {
        super();
    }

    /**
     * Obtenemos una conexión válida del pool de conexiones.
     * Si ya hemos obtenido el objeto DataSource, no necesitamos
     * volver a llamarlo, nos dará directamente una nueva conexión.
```

```
     */
    public static Connection getConexion(){
        Connection con = null;

        try {
            if (ds == null) {
                getPrgelearn();
                con = ds.getConnection();
                con.setAutoCommit(true);
            } else  {
                con = ds.getConnection();
                con.setAutoCommit(true);
            }
        }catch (Exception e){
            System.out.println(e.getMessage());
            e.printStackTrace();
        }

        return con;
    }

    /*
     * Obtenemos el objeto dataSource del pool de conexiones, para
     * obtener con él una conexión válida.
     */
    private static void getPrgelearn() throws javax.naming.NamingException {
        Context ctx = new InitialContext();
        ds = (DataSource) ctx.lookup(JNDI_CONNECTION);
    }
}
```

ParserDocXML.java

```
package tratamientoXML_BD;

import java.io.ByteArrayInputStream;
import java.io.InputStream;
import org.apache.xerces.dom.DocumentImpl;
import tratamientoXML_BD.NodoXML;
import general.ElementoXML;
import tratamientoXML_BD.TratamientoBD;
import org.apache.xerces.parsers.SAXParser;
import org.apache.xml.serialize.OutputFormat;
import org.apache.xml.serialize.XMLSerializer;
import org.xml.sax.SAXException;
import org.xml.sax.Attributes;
import org.xml.sax.ContentHandler;
import org.xml.sax.helpers.DefaultHandler;
import org.w3c.dom.*;

import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
```

```
import java.util.HashMap;
import java.util.ArrayList;
import java.io.IOException;
import java.io.File;
import java.io.StringWriter;

/* Clase encargada de realizar los parseos de los datos y documentos XML o manifest asociados a los
objetos virtuales que se creen o con los que se trabaje en la aplicación.
 */

public class ParserDocXML extends DefaultHandler {

    /*
     * Método que crea el documento de XML a partir de los datos presentes en la tabla
     * de elementos de la base de datos del identificador
     */
    public static String crearDocumentoXML(int idObjeto) throws Exception{
        DocumentImpl doc = new DocumentImpl();
        StringWriter s = null;
        Element raiz = null, aux = null;
        String atributo, valorAtributo, valor;
        HashMap tablaNodosArbol = new HashMap();
        String resultadoXML = null;

        //Recuperamos la lista de elementos correspondientes al Objeto Virtual
        ArrayList listaElementos = TratamientoBD.consultaElementosObjetoVirtual(idObjeto);

        for (int i=0;i<listaElementos.size();i++){
            ElementoXML elem = (ElementoXML) listaElementos.get(i);

            if (elem.getDependencia() == 0){
                //Obtenemos el elemento RAIZ desde el que cuelgan los demás
                raiz = doc.createElement("lom");
                tablaNodosArbol.put("" + elem.getIdElemento(), raiz);
            } else {
                Element actual = doc.createElement(elem.getNombre());

                //Obtenemos los atributos y demás
                atributo = elem.getAtributo().trim();
                valorAtributo = elem.getValorAtributo().trim();
                valor = elem.getValor().trim();

                //Insertamos los valores y atributos si son distintos de vacío
                if (!atributo.equals("")){
                    actual.setAttribute(atributo, valorAtributo);
                }
                if (!valor.equals("")){
                    actual.appendChild(doc.createTextNode(valor));
                }

                //Obtenemos el padre del elemento y le añadimos este nuevo hijo
                aux = (Element) tablaNodosArbol.get(""+elem.getDependencia());
```

```
            aux.appendChild(actual);

            //Insertamos el nodo en la tabla para tener referencia a los nodos
            //con el identificador correspondiente
            tablaNodosArbol.put("" + elem.getIdElemento(), actual);

        }
    }
    //FIN del Documento
    doc.appendChild(raiz);

    //Creamos el documento XML que hemos obtenido y que tenemos en el doc
    OutputFormat formato  = new OutputFormat(doc, "UTF-8", true);
    s = new StringWriter();
    XMLSerializer ser = new XMLSerializer(s, formato);

    try {
        ser.serialize(doc);
        resultadoXML = s.toString();
    } catch (IOException e) {
        throw new Exception(e.getMessage());
    }

    return resultadoXML;
}

/**
 * Lee el documento XML del fichero que se le pasa y lo mete sus elementos en la
 * base de datos en la tabla correspondiente junto a su identificador.
 * @param fichero
 */
public static void insercionArbolDOM(String fichero, int idObjeto) throws Exception{
    //Obtiene el documento XML
    Document doc = parseXmlFile(fichero, true);

    ArrayList listaNodos = new ArrayList();
    //Recorremos el árbol y metemos los nodos en una lista
    visit(doc, 0, listaNodos);
    //Recorremos los nodos para insertarlos en la BD
    int idAux = recorrerListaNodos(listaNodos, idObjeto);

    //Insertamos el documento XML correspondiente en la tabla de objetos
    StringWriter s = new StringWriter();
    OutputFormat formato  = new OutputFormat(doc, "UTF-8", true);
    XMLSerializer ser = new XMLSerializer(s, formato);

    try {
        ser.serialize(doc);
        TratamientoBD.insertarDocumentoObjetoTablaXML(idObjeto, s.toString());
    } catch (IOException e) {
        throw new Exception(e.getMessage());
    }
```

```
}

/**
 * Método encargado de recorrer los nodos del árbol de forma recursiva
 * y a su vez va generando una lista de nodos visitados.
 */
protected static void visit(Node node, int nivel, ArrayList listaNodos) {
    NodeList list = node.getChildNodes();

    for (int i=0; i<list.getLength(); i++) {
        Node childNode = list.item(i);
        String etiqueta = childNode.getNodeName();

        //Creamos un nuevo objeto nodo, con los datos del nodo visitado
        NodoXML nodo = new NodoXML();
        nodo.setNivel(nivel);
        nodo.setNodo(childNode);
        nodo.setNombreNodo(etiqueta);

        //Añadimos el nodo a la vista
        listaNodos.add(nodo);

        visit(childNode, nivel + 1, listaNodos);
    }
}

/**
 * Recorre la lista de nodos que hemos obtenido al visitar los nodos del árbol
 * y va insertando cada nodo en base de datos parseando así el documento XML,
 * con sus correspondientes elementos en la tabla de elementos del objeto virtual.
 */
protected static int recorrerListaNodos(ArrayList listaNodos, int idObjeto) throws Exception{
    NodoXML nodo = null;
    int numAtributos, idActual = 0, objetoInsertado = -1;
    String etiquetaAnt = null, valor = null, atributo = null, valorAtributo = null;
    Node childNode = null;
    ElementoXML elemPadre = null;

    //Recorremos la lista de nodos
    for (int i=0;i<listaNodos.size();i++){
        nodo = (NodoXML) listaNodos.get(i);
        childNode = nodo.getNodo();
        numAtributos = 0;

        valor = childNode.getNodeValue();

        if (childNode.getAttributes() != null){
            numAtributos = childNode.getAttributes().getLength();
        }

        if (numAtributos>0){
```

```
                //Cogemos el nodo atributo (supongo que solo hay uno)
                Node at = childNode.getAttributes().item(0);
                //Nombre del atributo
                atributo = at.getNodeName();
                //Lista de un elemento
                NodeList listaAtrib = at.getChildNodes();
                valorAtributo = listaAtrib.item(0).getNodeValue();
            }

            //Significa que es un nodo valor
            if (nodo.getNombreNodo().equals("#text")){
                //Determinamos si es el primer nodo del documento
                if (etiquetaAnt.equals("lom")){

                    //Insertamos la etiqueta raiz "lom" con sus datos correspondientes
                    idActual = TratamientoBD.insertarNodoEnBD("lom", "", "", "", 0, 1, idObjeto);

                    //Obtenemos el identificador del nuevo objeto virtual insertado
                    ElementoXML elem = TratamientoBD.getElem(idActual);
                    objetoInsertado = elem.getIdObjetoAsociado();
                } else if (!etiquetaAnt.equals("#text")) {
                    //Subimos dos niveles porque siempre hay un nodo #text en medio
                    Node aux = (Node) nodo.getNodo().getParentNode().getParentNode();

                    elemPadre = TratamientoBD.getElem(idActual);

                    //Obtenemos el identificador del padre para tener la correcta dependencia
                    int padre = TratamientoBD.consultaPadreEtiqueta(aux.getNodeName(),
elemPadre.getIdObjetoAsociado());

                    //Insertamos la etiqueta en la base de datos con todos los datos correspondientes
                    idActual = TratamientoBD.insertarNodoEnBD(etiquetaAnt, atributo,
                            valorAtributo, valor, padre, nodo.getNivel(),
elemPadre.getIdObjetoAsociado());

                    //Reseteo los valores para no repetir en una insercion
                    atributo = null;
                    valorAtributo = null;
                    valor = null;

                }
            }

            etiquetaAnt = nodo.getNombreNodo();
        }

        return objetoInsertado;
    }

    /**
     * Parsea el archivo XML y devuelve un documento DOM.
     * @param fichero
```

```
     * @param validar
     * @return Document
     */
    protected static Document parseXmlFile(String docXML, boolean validar) {
        Document doc = null;

        try {
            DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
            //Si queremos validar el documento con el DTD, descomentamos esta linea
            //factory.setValidating(validar);

            // Crea el constructor y parsea el fichero
            InputStream is = new ByteArrayInputStream(docXML.getBytes());
            doc = factory.newDocumentBuilder().parse(is);

        } catch (SAXException e) {
            e.printStackTrace();
        } catch (ParserConfigurationException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return doc;
    }
}
```

PresentacionAltaObjeto.java

```
package general;

import javax.sql.DataSource;

import java.util.List;
import java.util.ArrayList;

import beans.PresentacionArbol;

public class PresentacionAltaObjeto {

    public String getEspacios(int nivel) {

        String espacios = "";

        for (int i = 0; i < nivel; i++) {
            espacios = espacios + " " + " " + " ";
        }

        return espacios;
    }

    public String getContenidoLink(ElementoXML elem) {

        StringBuffer sb = null;
```

```
    String nombre = null, atributo = null, valorAtributo = null;

    sb = new StringBuffer();

    if (elem != null) {
        nombre = elem.getNombre();
        atributo = elem.getAtributo();
        valorAtributo = elem.getValorAtributo();

        if (nombre != null && !nombre.equals(""))
            sb.append(nombre);
        if (atributo != null && !atributo.equals(""))
            sb.append(" " + atributo + "=");
        if (valorAtributo != null && !valorAtributo.equals(""))
            sb.append("\"" + valorAtributo + "\"");
    }

    return sb.toString();
}

public String getContenidoElem(ElementoXML elem) {

    StringBuffer sb = null;
    String nombre = null;

    sb = new StringBuffer();

    if (elem != null) {
        nombre = elem.getNombre();

        if (nombre != null && !nombre.equals(""))
            sb.append(nombre);
    }

    return sb.toString();
}

public String getValorElem(ElementoXML elem) {

    StringBuffer sb = null;
    String valor = null;

    sb = new StringBuffer();

    if (elem != null) {
        valor = elem.getValor();

        if (valor != null && !valor.equals(""))
            sb.append(valor);
    }

    return sb.toString();
```

```
    }

    public String getCierreElem(ElementoXML elem) {

        StringBuffer sb = null;
        String nombre = null;

        sb = new StringBuffer();

        if (elem != null) {
            nombre = elem.getNombre();

            if (nombre != null && !nombre.equals(""))
                sb.append("&lt;/" + nombre + "&gt;");
        }

        return sb.toString();
    }

    /* El siguiente método devuelve un List de objetos PresentacionArbol con información  sobre cómo
representar cada parte del árbol de elementos ims y las cadenas a  mostrar id es el identificador del
objeto ElementoXML raíz
     */

    public List getArbol(int id) {

        ArrayList elementos = null;
        ElementoXML elem = null, padre = null;
        int i = -1, n = -1, nivel = -1, dependencia = -1, nivel_act = -1;
        String nombre = null, atributo = null, valorAtributo = null;
        String espacios = null, strId = null;
        boolean ultimo = false, tiene = false, fin = false;
        NegocioAltaObjeto neg = null;
        List lista_pos = null, lista_res = null;
        PresentacionArbol pa = null;
        String contenidoLink = null, valorElem = null;
        String cierreElem = null, cierrePadre = null;

        neg = new NegocioAltaObjeto();
        lista_res = new ArrayList();

        // tomar los elementos del árbol

            elementos = neg.getArbol(id);
            if (elementos != null)
                n = elementos.size();
            else
                n = 0;

            // añadir los objetos de presentación del árbol

            for (i = 0; i < n; i++) {
```

```
// tomar el elemento y sus datos

elem = (ElementoXML)elementos.get(i);
id = elem.getIdElemento();

nivel = elem.getNivel();
nombre = elem.getNombre();
atributo = elem.getAtributo();
valorAtributo = elem.getValorAtributo();
dependencia = elem.getDependencia();
ultimo = elem.getUltimo();
tiene = elem.getTiene();

// añadir espacios y conversión

espacios = getEspacios(nivel);
strId = Integer.toString(id);

//paramMap.put("strId",strId);
//request.setAttribute("elemento", paramMap);

// tomar cadenas para presentación

contenidoLink = getContenidoLink(elem);
valorElem = getValorElem(elem);

// añadir objetos de presentación del elemento actual

pa = new PresentacionArbol();
pa.setElem("LABEL");
pa.setCadena(espacios);
lista_res.add(pa);

lista_pos = neg.getPosiblesHijos(id);
pa = new PresentacionArbol();
if (lista_pos != null) {
    pa.setElem("LINK");
    pa.setStrId(strId);
}
else {
    pa.setElem("LABEL");
}
pa.setCadena("&lt;" + contenidoLink + "&gt;");
lista_res.add(pa);

pa = new PresentacionArbol();
pa.setElem("LABEL");
pa.setCadena(valorElem);
lista_res.add(pa);

// insertar cierres
```

```
                if (!tiene) {
                    cierreElem = getCierreElem(elem);
                    pa = new PresentacionArbol();
                    pa.setElem("LABEL");
                    pa.setCadena(cierreElem + "<br>");
                    lista_res.add(pa);
                    //out.println(cierreElem + "<br>");

                    fin = false;
                    nivel_act = nivel;
                    while (ultimo && !fin) {
                        padre = neg.buscarElemArbol(dependencia, elementos);
                        if (padre != null) {
                            nivel_act--;
                            cierrePadre = getCierreElem(padre);
                            espacios = getEspacios(nivel_act);

                            pa = new PresentacionArbol();
                            pa.setElem("LABEL");
                            pa.setCadena(espacios + cierrePadre + "<br>");
                            lista_res.add(pa);

                            //out.println(espacios + cierrePadre + "<br>");
                            dependencia = padre.getDependencia();
                            ultimo = padre.getUltimo();
                        }
                        else {
                            fin = true;
                        }
                    }
                }
                else {
                    pa = new PresentacionArbol();
                    pa.setElem("LABEL");
                    pa.setCadena("<br>");
                    lista_res.add(pa);
                    //out.println("<br>");
                }
            }  // for

            return lista_res;
        }
    }
```