REPOSITORIO DE OBJETOS DE APRENDIZAJE DISTRIBUIDO

Proyecto de Sistemas Informáticos

Autores
Jesús Javier Marcos Granjo
Alberto Ortega Valentín
Jesús Bartolomé Sanz

Profesor director
Antonio Sarasa Cabezuelo

Facultad de Informática
Universidad Complutense de Madrid
Curso 2006-2007

RESUMEN

En este proyecto de la asignatura *Sistemas Informáticos* se desarrolla un repositorio distribuido de objetos de aprendizaje, el cual se compone de varios nodos. Cada uno de ellos comparte su propio repositorio con el resto mediante una función de búsqueda.

Se han utilizado Web Services para implementar la comunicación entre los distintos nodos del sistema. Dentro de cada uno de éstos, un servidor Tomcat se encarga de gestionar la aplicación y en él se almacenan los objetos de aprendizaje de dicho nodo. Por otra parte, se utiliza un gestor de base de datos DB2 para almacenar la información concerniente a los usuarios (datos personales, permisos…), a los objetos de aprendizaje (nombre, comentarios, autor, ruta física en el servidor, metadatos, fecha de creación…) y a las taxonomías (nombre, ruta…). También se emplea este gestor para la realización de una búsqueda más eficiente sobre los metadatos XML, mediante el lenguaje XQuery.

SUMMARY

In this project of the Sistemas Informáticos subject we develop a distributed repository of learning objects, which is composed of several nodes. Each of these nodes shares his own repository with the rest of them through a search function.

We use web services to implement the communication amongst the different system nodes. A Tomcat server manages the application in each node and stores the learning objects of that particular node. A DB2 database manager is used to store the information regarding to the users (personal data, privileges), the learning objects (name, comments, author, path, metadata, creation date…) and the taxonomies (name, path…). We use this database manager to develop a more efficient search over the XML metadata too, with the XQuery language.

PALABRAS CLAVE

- Objeto de Aprendizaje
- Repositorio distribuido
- Servicio Web / Web Service
- XQuery
- Google Web Toolkit

INDICE

1.- INTRODUCCION:

En los últimos años se han desarrollado distintas iniciativas que tenían como finalidad estandarizar la generación de material educativo digital en la forma de los denominados objetos de aprendizaje. Estas entidades albergan en su interior el contenido, la estructura y la presentación del contenido. Una de las motivaciones de estas iniciativas es facilitar la reutilización de los objetos ya existentes para la generación de nuevos objetos más complejos. Es por ello que una pieza clave para conseguir este objetivo es la disponibilidad de repositorios de objetos, su publicación, visibilidad y recuperación. En torno a esta necesidad se han definido distintas aproximaciones de cómo implementar un repositorio y un protocolo de comunicación que facilite este proceso de búsqueda/recuperación. Con esta motivación de trasfondo, en el presente proyecto de Sistemas Informáticos se ha desarrollado una implementación de una red de repositorios de objetos de aprendizaje distribuidos, junto a otras herramientas que complementan la funcionalidad principal de la aplicación.

Objetos de aprendizaje

A nivel conceptual un objeto de aprendizaje consta de tres elementos: unos contenidos, unas descripciones del comportamiento del objeto, y un conjunto de metadatos que hacen referencia a los objetos. Aunque las implementaciones de un objeto de aprendizaje pueden ser muy variadas y pueden tener particularidades, todas tienen en común el implementar los objetos como unidades compuestas por un documento que describe los contenidos y su relación, y los contenidos propiamente descritos por el documento. Uno de los objetivos principales en el ámbito de los objetos de aprendizaje es conseguir la reutilización de los recursos empaquetados dentro de los mismos en contextos y aplicaciones diferentes para los que fueron diseñados inicialmente. Para ello es necesario disponer de mecanismos que faciliten la interoperabilidad entre sistemas heterogéneos. En este sentido diferentes instituciones relacionadas con el ámbito de la educación han llevado a cabo un proceso de definición de estándares sobre diversos aspectos relacionados con los objetos de aprendizaje con la doble finalidad de conseguir reutilización e interoperabilidad

Repositorios distribuidos de objetos de aprendizaje

Un repositorio digital es un almacén de recursos digitales a los que se puede acceder sin que sea necesario un conocimiento previo de la organización o la estructura de dicho almacén. Además de los componentes recopilados, se contempla un almacenamiento de metadatos que aporten información sobre dichos componentes y que son el elemento principal que permiten la recuperación de los objetos. Existen diferentes iniciativas para la creación de repositorios digitales de contenidos y la interoperabilidad entre ellos, pero cabe destacar:

- IMS DRI. (IMS Digital Repository Interoperability).La especificación facilita un esquema funcional de la arquitectura del sistema y un modelo de referencia completo para la interoperabilidad de repositorios. El modelo de referencia define ocho funciones relevantes, repartidas en dos áreas. Por un lado a nivel del repositorio y por otro lado a nivel de manejo de los recursos.

- OAI (Open Archives Initiative) es una iniciativa para desarrollar y promover estándares de interoperabilidad para la difusión de contenidos en Internet. No está específicamente orientada a los contenidos educativos sino a cualquier contenido digital. El objetivo de OAI es crear una forma simple y sencilla de intercambiar información (concretamente metadatos) entre repositorios heterogéneos que alberguen cualquier objeto que contenga metadatos asociados. Para ello OAI desarrolló el Protocolo PMH (Protocol for Metadata Harvesting) que permite el intercambio de estos metadatos entre repositorios. Este protocolo define los mecanismos para recolectar los registros de los repositorios que contienen metadatos.

- La iniciativa OKI (Open Knowledge Initiative) desarrolla y promueve especificaciones que describen cómo los componentes de un entorno software se pueden comunican con otros sistemas. Las especificaciones proporcionadas por OKI permiten la interoperabilidad e integración de sistemas, definiendo los estándares para una arquitectura orientada al servicio (Service Oriented Architecture SOA). El modelo de arquitectura planteado por OKI aplica los conceptos básicos de separación, ocultación y jerarquización en capas, para obtener los beneficios de la interoperabilidad y la integración simple.

- SQI (Simple Query Interface) es una especificación que pretende ser una capa que garantice la interoperabilidad entre redes o entornos educacionales heterogéneos. El objetivo es ser una parte del sistema que sea capaz de buscar en los distintos repositorios (heterogéneos) de objetos educativos existentes en las redes a pesar de que posean interfaces propietarias de búsqueda de cada fabricante. Para permitir la interoperabilidad entre repositorios digitales heterogéneos, es necesario tener en cuenta ciertos aspectos. Por un lado, se necesita un modelo semántico común, el cual especifique el formato de las distintas propiedades de los objetos contenidos en los repositorios. Por otro lado, la interoperabilidad está basada en protocolos comunes, los cuales definen las interacciones posibles entre los repositorios. Para ello se debe disponer de una gran variedad de protocolos para intentar cubrir un amplio espectro de repositorios.

2.- HERRAMIENTAS SOFTWARE:

Para implementar nuestra aplicación Web, hemos usado los siguientes componentes software:

2.1.- Apache Tomcat:

Es el servidor Web que nos permite mantener aplicaciones Web dinámicas, en nuestro caso, desarrolladas mediante JSPs. Internamente, el servidor Apache Tomcat compila los ficheros JSP y los transforma en servlets.

Contiene diversos directorios, entre los que destacan:

- *webapps*: es el directorio donde se almacenan las aplicaciones
- *common/lib*: directorio de librerías comunes para todas las aplicaciones. En nuestro caso, incluimos las librerías necesarias para DB2, PDF y envío de ficheros de cliente a servidor, además de las que trae por defecto la instalación.
- *work*: directorio en el que se crean los servlets que resultan de la compilación de los JSP.

2.2.- Apache Axis

Para la realización de este proyecto, nos hemos decantado por la utilización de un software basado en Java de libre distribución: Axis, de la Apache Software Foundation. Básicamente Axis está formado por un conjunto de librerías Java que nos permitirán construir procesadores SOAP para ser utilizados en la implementación de clientes de Servicios Web, pero también para desarrollar nuestros propios servicios o incluso pasarelas, gateways, que mediarán en la invocación a servicios. Para facilitar esta tarea de desarrollo, Axis proporciona algunas facilidades como son por ejemplo:

- Un sencillo servidor Web para los servicios que desarróllenos.
- Herramientas de conversión entre Java y WSDL
- Herramientas de conversión entre WSDL y la generación de stubs
- Herramientas de depuración como un monitor de conexiones TCP.

Asimismo, Axis proporciona los mecanismos necesarios para que los Servicios Web ya desarrollados puedan ser fácilmente instalados en servidores más avanzados que el que Axis proporciona, como son el Apache Tomcat (en nuestro caso, haremos uso del proporcionado por Apache Tomcat).

Para ver la creación, despliegue y utilización de los Servicios Web, (ver Apéndice B).

2.3.- IBM DB2 UDB V9.0

Es el gestor de bases de datos en el que almacenamos toda la información relacional necesaria para el mantenimiento del sistema, incluyendo los archivos de metadatos XML.

El uso de DB2 V9 ofrece numerosas ventajas, incorporadas en esta nueva versión, respecto al tratamiento de XML. Principalmente, permite:

- Guardar los metadatos XML directamente en una columna de tipo XML. La inserción, actualización y borrado de un campo XML se realiza del mismo modo que un campo de cualquier otro tipo.
- Consultar todo o parte de documentos XML
- Combinar datos XML con relacionales en una única consulta
- Publicar datos relacionales como XML bien formado
- Descomponer documentos XML en tablas relacionales.
- Añadir un esquema XSD y validar documentos XML contra dicho esquema

Sin embargo, también nos hemos encontrado con alguna limitación debido a que usamos la versión Express o gratuita de DB2, como por ejemplo, el tamaño del XML no podía ser mayor de 32Kb

Realizamos la conexión a DB2 a través de JDBC, mediante un controlador universal de tipo 4: com.ibm.db2.jcc.DB2Driver

Básicamente, las operaciones que realizamos contra la base de datos son: conexión, desconexión, commit, consulta (SELECT…), inserción de filas (INSERT…), borrado de filas (DELETE…) y actualización de filas (UPDATE…). Todas las sentencias están realizadas mediante SQL, combinado con XQuery en el caso de las consultas sobre los metadatos XML.

2.3.1.- XQuery

XQuery es un lenguaje de consultas diseñado para fuentes de datos XML, e incluido como novedad, y en exclusiva, en la versión 9 de DB2. Permite buscar, realizar cálculos, extraer valores e incluso construir nuevos documentos XML.

Este lenguaje se basa, a su vez, en XPath, el cual se usa para navegar y seleccionar fragmentos de documentos XML.

DB2 proporciona dos funciones para recuperar documentos XML mediante XQuery:

- *db2-fn:xmlcolumn(xml-column-name)*
- *db2-fn:sqlquery(select xml-column-name from table-name)*

Xquery se basa en 5 cláusulas, que se conoce como sentencia FLWOR:

- **F**OR: itera a lo largo de una secuencia, y asigna una variable a cada elemento de la secuencia

- **L**ET: asigna una variable a una secuencia

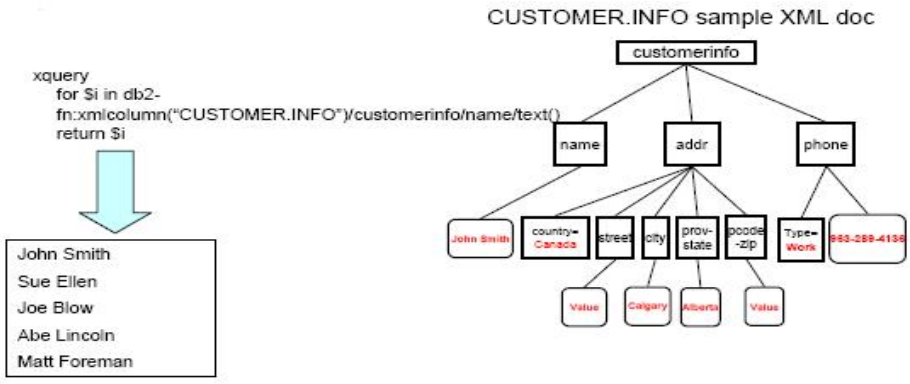
- **W**HERE: elimina coincidencias

- **O**RDER BY: reordena coincidencias

- **R**ETURN: construye los resultados de una consulta

Ejemplo:

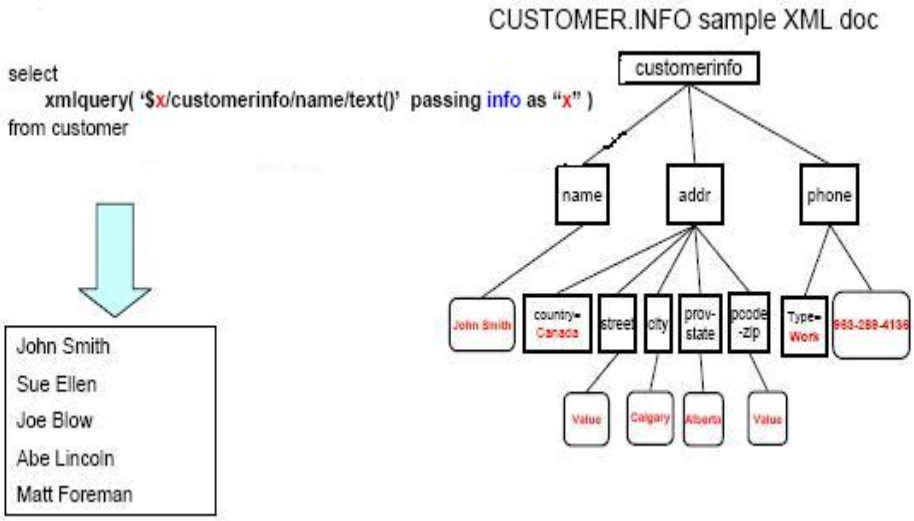
```
xquery                          <tag> 9 </tag>
     for $i in (4,3,2,1)        →        <tag> 12 </tag>
     let $x := $i * 3
 order by $x
 where $x > 6
 return <tag> { $x } </tag>
```



Sin embargo, DB2 también permite realizar consultas sobre datos XML mediante SQL.

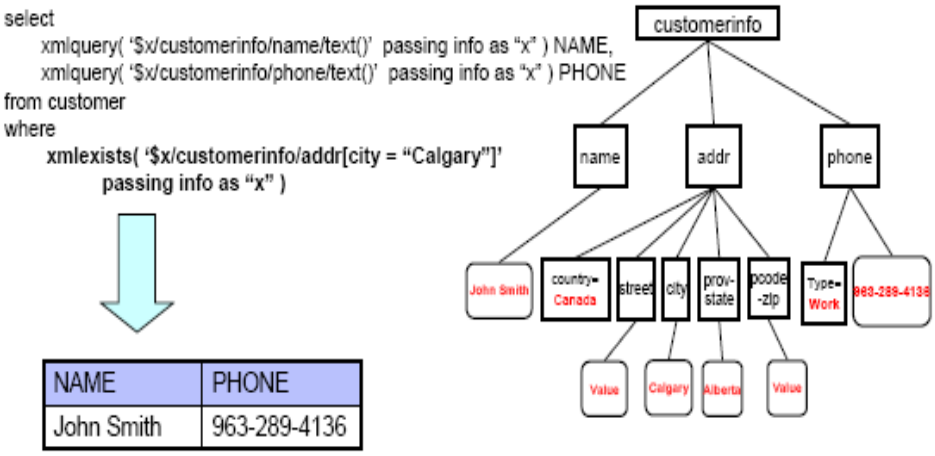
De esta forma, se comienza con una sentencia SELECT y se usa la función *xmlquery()* para ejecutar una XQuery dentro de la SELECT. Esta función es escalar y devuelve una sola secuencia XML por cada fila procesada, y siempre una (si no encuentra ninguna parte de la expresión, devuelve una secuencia vacía).

Ejemplo:



Puede usarse también la función *xmlexists()* en una cláusula WHERE para eliminar filas en base a contenidos de un documento XML

Ejemplo:



Como ya se ha mencionado previamente, podemos generar datos relacionales a partir de datos XML.

Para ello, usamos la función *xmltable()*, que mapea datos de un XML en una tabla relacional construida dinámicamente.

Ejemplo:

```
SELECT TB.COLNAME, TB.COLPHONE
FROM customer c,
XMLTable('$cu/customerinfo' passing c.info as "cu"
 COLUMNS
  "COLNAME"  CHAR(20)  PATH 'name',
  "COLPHONE" CHAR(12)  PATH 'phone'
 ) AS TB
```

COLNAME	COLPHONE
John Smith	963-289-4136
Sue Ellen	555-444-3655
Joe Blow	555-555-3687
Abe Lincoln	333-333-2879
Matt Foreman	222-222-8513

Las funciones de publicación de XML recogen datos y crean tipos de nodo XML o documentos XML completos:

- *XMLELEMENT():* crea un elemento XML
- *XMLATTRIBUTES():* usado dentro de XMLELEMENT para crear atributos

Ejemplo:

```
SELECT
    XMLELEMENT(NAME "empname", e.firstnme)
    AS "Result"
FROM employee e
```

Result
<empname>John Smith</empname>
<empname>Sue Ellen</empname>
<empname>Joe Blow</empname>
<empname>Abe Lincoln</empname>
<empname>Matt Foreman</empname>

Por último, la función *xmlagg()* agrupa elementos en lugar de producir un elemento por fila.

Ejemplo:

```
select
  xmlelement(NAME "products",
    xmlagg(
      xmlelement(NAME "pid", p.pid)
    )
  )
from product p
```

```
<products>
  <pid>100-100-100</pid>
  <pid>100-101-100</pid>
  <pid>100-102-100</pid>
  <pid>100-103-100</pid>
</products>
```

2.4.- AJAX (*Asynchronous JavaScript And XML*)

AJAX se ha convertido en una de las tecnologías más populares para la creación de aplicaciones Web dinámicas, con resultados tan interesantes como los conseguidos en Gmail.

Esta tecnología nos permite mantener una comunicación asíncrona entre el navegador y el servidor utilizando JavaScript y XML. Esto quiere decir que podemos realizar cambios en la página sin necesidad de volver cargarla.

La simple mención de JavaScript puede causar ciertas reticencias entre muchos programadores, ya sea porque las herramientas no son tan maduras como en otras plataformas, o por las dificultades que nacen de las incoherencias con los estándares de algunos navegadores.

2.5 Google Web Toolkit

Por esto disponemos GWT (Google Web Toolkit) que es un soporte de desarrollo de código libre que permite crear aplicaciones con AJAX programando desde JAVA. En cierta forma el GWT 'traduce' el código JAVA a JavaScript permitiendo crear aplicaciones dinámicas para Web compatibles con la mayoría de los navegadores de una forma sencilla.

GWT nos ahorra las complicaciones de uso de JavaScript permitiéndonos escribir nuestra interfaz en Java y obtener, una vez compilada, XHTML y JavaScript compatible con todos los navegadores.

2.5.1.- Instalación de GWT

Para la instalación del Google Web Toolkit hay que realizar los siguientes pasos:

1.- Instalación de Java SDK.

Versión superior a 1.4

2.- Descargarse el ".zip" que contiene el Google Web Toolkit.

Lo podemos encontrar en la siguiente URL
http://code.google.com/webtoolkit/

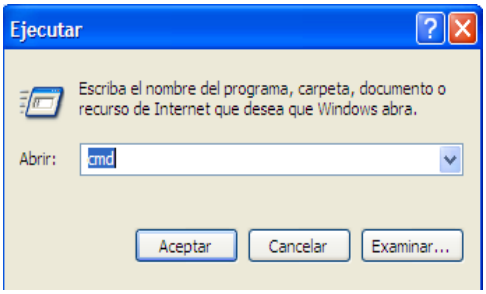
3.- Descomprimir el archivo ".zip" del Google Web Toolkit.

GWT no tiene aplicación de instalación. Todos los archivos que necesitas para ejecutar y usar GWT se encuentran en el directorio donde hemos extraído el .zip. A continuación se explica la creación de una aplicación cuyo desarrollo se llevara a cabo sobre el framework Eclipse, que podemos conseguir en http://www.eclipse.org, que es de código libre.

2.5.2.- Creación de una Aplicación GWT (con Eclipse)

GWT dispone de un comando llamado *applicationCreator* que automáticamente genera todos los archivos que se necesitará para comenzar un proyecto de GWT. También se puede generar archivos de proyecto de Eclipse y archivos config para la depuración. Si queremos generar un proyecto en Eclipse para una nueva aplicación, primero tenemos que usar el comando *projectCreator* que generar un proyecto de Eclipse para tu aplicación, sigue los siguientes pasos:

1.- Abrir una ventana de comandos. (Inicio->Ejecutar…)



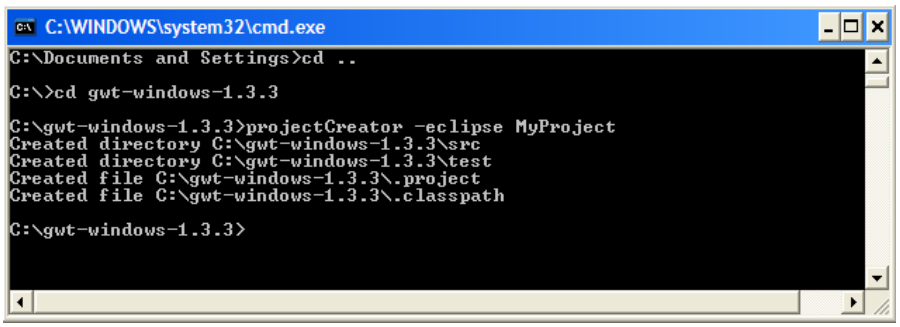
2.- Escribir lo siguiente en la línea de comandos.

projectCreator -eclipse MyProject

Donde:

-*MyProject*: Nombre que se desee dar al proyecto.

-“*-eclispe*”: Indica que queremos generar un proyecto para Eclipse.



3.- A continuación escriba la siguiente en la línea de comandos, siendo *MyProyect* el nombre que se le ha dado al proyecto en el aportado anterior:

applicationCreator -eclipse MyProject com.mycompany.client.MyApplication

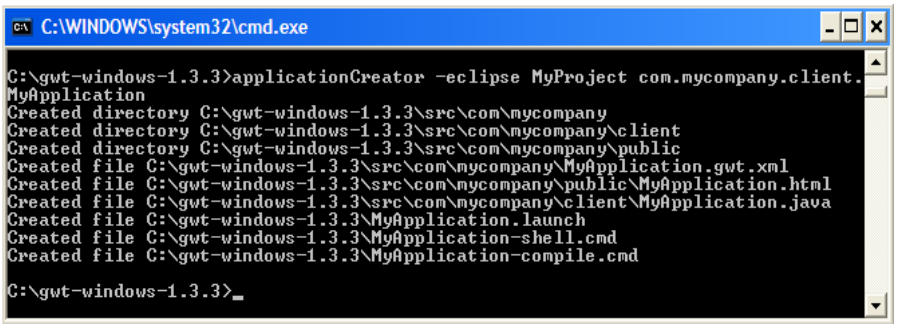
Donde:

-“*-eclispe*”: Indica que queremos generar un proyecto para Eclipse.

- *MyProject* : Nombre que se le ha dado al proyecto en el apartado anterior.

- *mycompany*: Nombre de la compañía para la que se trabaja o cualquier otro nombre.

-*MyApplication:* Nombre que queremos darle a nuestra aplicación.

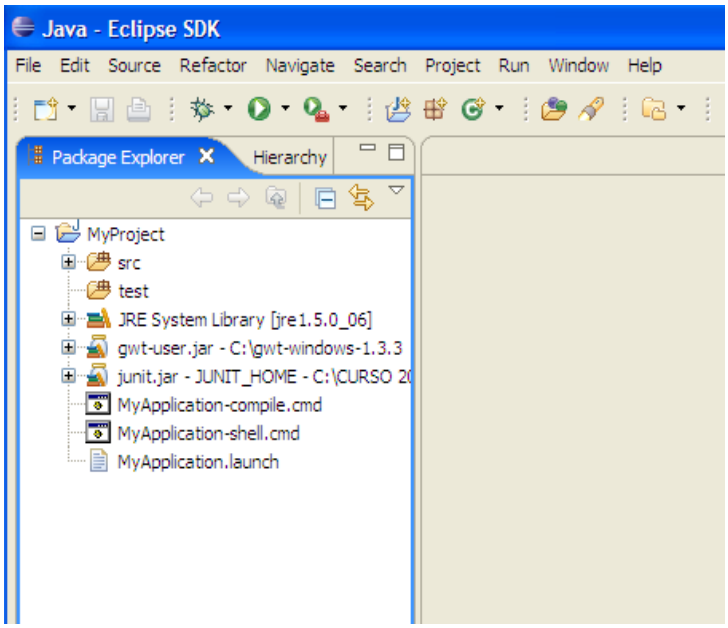


Cuando se han completados todos los pasos anteriores en el directorio del GWT deberían aparecer los scripts MyApplication-shell y MyApplication-compile así como .project, .classpath, y el archivo MyApplication.launch.

Para que los archivos generados no se mezcle y por mayor claridad, crearemos una carpeta donde guardaremos todos los archivos de este proyecto.

Para abrir el proyecto que hemos generado en el Eclipse, hay que abrir el Eclipse y seleccionar “*File/Import…*”. En la ventana que aparece elegir "Existing Projects into Workspace" e indicar el directorio en el cual tenemos

nuestro proyecto. Cuando se ha completado esto, debería aparecer el proyecto de GWT cargado en la zona de trabajo de Eclipse:



Mediante eclipse podemos probar y depurar nuestra aplicación. Para ejecutarlo hay que hacer clic en el botón verde "Run" y ya tenemos el proyecto creado ahora solo tenemos que desarrollar nuestra aplicación. También puede ejecutarse mediante el script *MyApplication-compile.*

2.5.3.- Generación de los archivos de la aplicación AJAX

Una vez que hemos desarrollado nuestra aplicación y la hemos probado con eclipse podemos pasar a generar los archivos AJAX para nuestra aplicación web. Lo archivos se regeneran simplemente ejecutando el script *MyApplication-compile*, este script "traduce" el código java que hemos generado en una aplicación AJAX y lo de deja en la carpeta "WWW" de nuestro proyecto.

3.- DESCRIPCION DE LA BASE DE DATOS

La base de datos la utilizaremos para guardar toda la información de los usuarios, objetos de aprendizaje, ….

3.1.- Tablas de la aplicación

La base de datos sobre la cual trabaja nuestra aplicación se basa en 6 tablas, todas ellas con esquema *WEB* para diferenciarlas de las tablas propias de DB2:

- Tabla de usuarios del sistema (*web.usuarios*): cada fila está unívocamente identificada por el nombre del usuario, y se compone de los siguientes campos:
  - Nombre de usuario (*usuario*)
  - Contraseña de usuario (*clave*)
  - Número de teléfono (*teléfono*)
  - Correo electrónico (*email*)
  - Tipo de usuario (*tipo*): toma los valores -1 si el usuario se ha registrado pero todavía no está confirmado por el administrador; 0 si es un usuario con privilegios comunes; 1 si es un usuario administrador; 2 si es el super administrador (usuario *admin*)
  - Idioma del usuario (*idioma*): toma los valores 0 si su perfil de idioma es el español; 1 si es inglés.

- Tabla de objetos de aprendizaje del sistema (*web.objetos*):
  - Código autogenerado que identifica unívocamente cada fila (*código*).
  - Nombre del OA (*nombre*)
  - Autor del OA (*autor*): contiene integridad referencial respecto al campo *usuario* de la tabla de usuarios.
  - Contenido del fichero imsmanifest.xml del OA (*objeto*)
  - Ruta relativa del OA dentro de la aplicación (*ruta*)
  - Fecha de envío del OA (*fechaenvio*)
  - Indicador de si el OA ha sido confirmado por el administrador – valor 1 – o no – valor 0 – (*registrado*).

- Tabla de taxonomias (*web.taxonomias*):
  - Código autogenerado que identifica unívocamente cada fila (*código*).
  - Nombre de la taxonomía (*nombre*)
  - Contenido del fichero .xml que define la taxonomía (*taxonomia*)
  - Ruta relativa de la taxonomía dentro de la aplicación (*ruta*)
  - Fecha de envío del OA (*fechaenvio*)

- Tabla de comentarios de los OA (*web.comentarios*):

  Cada fila está unívocamente identificada por el autor del comentario y el código del OA, de tal forma que un mismo usuario no pueda comentar el mismo OA más de una vez:

  - Código del OA comentado (*codigoObjetos*): contiene integridad referencial respecto al campo *código* de la tabla de objetos.
  - Fecha del comentario (*fecha*)
  - Autor del comentario (*autor*): contiene integridad referencial respecto al campo *usuario* de la tabla de usuarios.
  - Comentario (*comentario*)

- Tabla de equivalencias entre los comandos del lenguaje de búsqueda y los campos XML (*web.equivalencias_xml*):

  - Código autogenerado que identifica unívocamente cada fila (*código*).
  - Comando del lenguaje de búsqueda (*campo*).
  - Ruta LOM del campo XML (*linea_xml*)

- Tabla de palabras no tratadas (*web.palabras_no_tratadas*):

  - Palabra no tratada (*palabra*): cada valor de este campo es único

- Tabla de signos eliminados (*web.signos_eliminados*):

  - Signo eliminado (*signo*): cada valor de este campo es único

Las integridades referenciales están especificadas de tal forma que al eliminar una fila se borren también las filas de otras tablas que la referencian (sentencia *ON DELETE CASCADE*), y por tanto, no se produzca error.

Por ejemplo, si el administrador decide eliminar un OA del sistema, automáticamente el gestor DB2 se encargará de eliminar los comentarios asociados a dicho OA; lo mismo ocurre para los comentarios de un usuario, y para los OA de un usuario.

En el caso de los OA de un usuario, podría obviarse la integridad referencial para permitir que un OA sobreviva en el sistema aunque su autor haya desaparecido; sin embargo, hemos optado por la opción contraria, ya que así el sistema es más consistente, y en todo caso, el administrador puede descargar el OA antes de eliminar el autor y después volverle a enviar con su propio usuario.

3.2.- Sentencias de creación de las tablas

Las sentencias de creación de las tablas son:

```
CREATE TABLE web.USUARIOS (
      usuario VARCHAR(20) NOT NULL,
      clave VARCHAR(20) NOT NULL,
      idioma SMALLINT NOT NULL,
     tipo SMALLINT NOT NULL,
     telefono DECIMAL(9,0) NOT NULL,
     email VARCHAR(30) NOT NULL,
    PRIMARY KEY(usuario)
);

CREATE TABLE web.OBJETOS (
      codigo INTEGER NOT NULL GENERATED ALWAYS AS IDENTITY (START
            WITH 0, INCREMENT BY 1),
      nombre VARCHAR(20) NOT NULL,
      autor VARCHAR(20) NOT NULL,
      objeto XML NOT NULL,
      ruta VARCHAR(200) NOT NULL,
      registrado SMALLINT NOT NULL,
      fechaenvio CHAR(10) NOT NULL,
      PRIMARY KEY(codigo)
     );


 ALTER TABLE web.OBJETOS
     ADD CONSTRAINT fk_autorOA
     FOREIGN KEY(autor) REFERENCES web.usuarios(usuario)
     ON DELETE CASCADE
    ;

CREATE TABLE web.TAXONOMIAS (
      codigo INTEGER NOT NULL GENERATED ALWAYS AS IDENTITY (START
                WITH 0, INCREMENT BY 1),
      nombre VARCHAR(20) NOT NULL,
      taxonomia XML NOT NULL,
      ruta VARCHAR(200) NOT NULL,
      fechaenvio CHAR(10) NOT NULL,
      PRIMARY KEY(codigo)
     );

CREATE TABLE web.COMENTARIOS (
      codigoObjetos INTEGER NOT NULL,
      fecha CHAR(10) NOT NULL,
      autor VARCHAR(20) NOT NULL,
      comentario VARCHAR(300) NOT NULL,
      PRIMARY KEY(autor,codigoObjetos)
     );

 ALTER TABLE web.COMENTARIOS
     ADD CONSTRAINT fk_comentarios
     FOREIGN KEY(codigoObjetos) REFERENCES web.objetos(codigo)
```

```
    ON DELETE CASCADE
    ;

ALTER TABLE web.COMENTARIOS
    ADD CONSTRAINT fk_autorComentario
    FOREIGN KEY(autor) REFERENCES web.usuarios(usuario)
    ON DELETE CASCADE
    ;


CREATE TABLE web.PALABRAS_NO_TRATADAS (
     palabra VARCHAR(50) NOT NULL,
     PRIMARY KEY(palabra)
    );

CREATE TABLE web.SIGNOS_ELIMINADOS (
     signo VARCHAR(50) NOT NULL,
     PRIMARY KEY(signo)
    );

CREATE TABLE web.EQUIVALENCIAS_XML (
     codigo INTEGER NOT NULL GENERATED ALWAYS AS IDENTITY (START
          WITH 0, INCREMENT BY 1),
     campo VARCHAR(100) NOT NULL,
     linea_xml VARCHAR(300) NOT NULL,
     PRIMARY KEY(campo)
    );
```

4.- INSTALACION DE LA APLICACION

4.1.- Prerrequisitos

Antes de proceder a la instalación de nuestra aplicación, es necesario instalar y configurar otras herramientas software necesarias para el funcionamiento de la misma:

- JSDK 1.5 o superior
- Usuario de Windows con nombre y contraseña *db2admin* con permisos de administrador
- DB2 UDB V 9.0 o superior
- Apache Tomcat versión 5.0 o superior
  En este caso es necesario, además, añadir las librerías jar de DB2 en la carpeta *Tomcat/common/lib,* así como la librería de PDF *itext-1.4.7.jar* y las librerías *commons-io-1.3.1.jar, commons-fileupload-1.2.jar* necesarias para el envío de ficheros al servidor.
- Axis

A continuación incluimos una guía de instalación/configuración de DB2 y Tomcat. Dicha guía usa DB2 UDB 9.0 y Tomcat 5.0, aunque con una versión superior de Tomcat el proceso sería similar. También incluimos una guía de instalación de Axis.

4.2.- Instalación de Apache Tomcat y conexión con DB2

Los pasos a seguir son los siguientes:

1. Instalación de Tomcat 5.0 utilizando el asistente de instalación para Windows.
2. Para instalar Tomcat 5.0 en el instalador poner la dirección del JDK (no apuntando al JRE).
3. Los archivos con las páginas web tienen que instalarse en la carpeta *Tomcat 5.0/webapps*
4. Para poder activar y desactivar el Tomcat se tiene que seleccionar la opción: Monitor Tomcat a la que se puede acceder a través de Inicio de Windows. Aparecerá un icono en la parte inferior derecha del Escritorio.
5. Para usar Tomcat la dirección de la página web es *http://127.0.0.1:8080*.
6. Para conectar DB2 es necesario coger los archivos *c:/ibm/sqllib/java/db2java.zip, c:/ibm/sqllib/java/db2jcc.jar,*

c:/ibm/sqllib/java/db2jcc_license_cu.jar y *c:/ibm/sqllib/java/sqlj.zip* y pegarlos en la carpeta *Tomcat 5.0/common/lib*; una vez pegados, renombrar los .zip para que sean .jar

7. Para que la conexión funcione correctamente es necesario que DB2 este iniciado.

8. Para ver que funciona correctamente utilizar el archivo *pruebadb.jsp (ver Apéndice A)*, colocándolo en *Tomcat 5.0/webapps/root* e invocándolo desde Internet como *http://127.0.0.1:8080/pruebabd.jsp.* Para que funcione correctamente tener en cuenta que habrá que cambiar el nombre de la base de datos, el usuario y la contraseña.

4.3.- Instalación del entorno Apache Axis

Los pasos de instalación son los siguientes:

1.- Descargar e instalar Apache Tomcat. En nuestro caso hemos elegido la versión Apache Tomcat 5.0.28 Server, que se puede descargar en la página oficial de Apache Tomcat (http://jakarta.apache.org/tomcat/)

2.- Descargar Axis, versión 1.4. Hemos elegido dicha versión porque parece que es la más estable. Para descargarlo, podemos acceder a (http://mirrors.hostalia.com/apache/ws/axis/1_4/axis-bin-1_4.zip).

3.- Copiar la carpeta axis-1_4\webapps\axis al directorio \Tomcat\webapps

4.- Copiar los ficheros adjuntos en este tutorial (class.bat y wsdl.bat) a \AXIS-1_3

```
SET AXIS_HOME=C:\apps\axis-1_3
SET
CLASSPATH=%CLASSPATH%;%AXIS_HOME%\lib\axis.jar;%AXIS_HOME%\lib\axisant.
jar;%AXIS_HOME%\lib\axis-schema.jar
SET CLASSPATH=%CLASSPATH%;%AXIS_HOME%\lib\commons-discovery-
0.2.jar;%AXIS_HOME%\lib\commons-logging-
1.0.4.jar;%AXIS_HOME%\lib\jaxrpc.jar
SET CLASSPATH=%CLASSPATH%;%AXIS_HOME%\lib\log4j-
1.2.8.jar;%AXIS_HOME%\lib\saaj.jar;%AXIS_HOME%\lib\wsdl4j-1.5.1.jar
```

class.bat

```
SET AXIS_HOME=C:\apps\axis-1_3
SET
CLASSPATH=%CLASSPATH%;%AXIS_HOME%\lib\axis.jar;%AXIS_HOME%\lib\axisant.
jar;%AXIS_HOME%\lib\axis-schema.jar
SET CLASSPATH=%CLASSPATH%;%AXIS_HOME%\lib\commons-discovery-
0.2.jar;%AXIS_HOME%\lib\commons-logging-
1.0.4.jar;%AXIS_HOME%\lib\jaxrpc.jar
SET CLASSPATH=%CLASSPATH%;%AXIS_HOME%\lib\log4j-
1.2.8.jar;%AXIS_HOME%\lib\saaj.jar;%AXIS_HOME%\lib\wsdl4j-1.5.1.jar
java org.apache.axis.wsdl.WSDL2Java
http://localhost:8080/axis/Calculator.jws?wsdl
```

wsdl.bat

5.- Modifica en ambos ficheros de texto (class.bat y wsdl.bat) la variable de entorno AXIS_HOME para que apunte a tu directorio AXIS.

6.- Descargar dos librerías necesarias para que funcione correctamente Axis:

- mail.jar, que se puede descargar desde
  http://java.sun.com/products/javamail/
- activation.jar, que se puede descargar desde
  http://java.sun.com/products/javabeans/jaf/downloads/index.html

7.- Copiar estas dos librerías a TOMCAT-HOME/ common/lib

Es recomendable, como en todo entorno basado en Java, para poder utilizar desde nuestros programas las funcionalidades ofrecidas por las librerías de Axis, incluir en el classpath los ficheros jar de dichas librerías. En concreto será necesario añadir al classpath los siguientes ficheros:

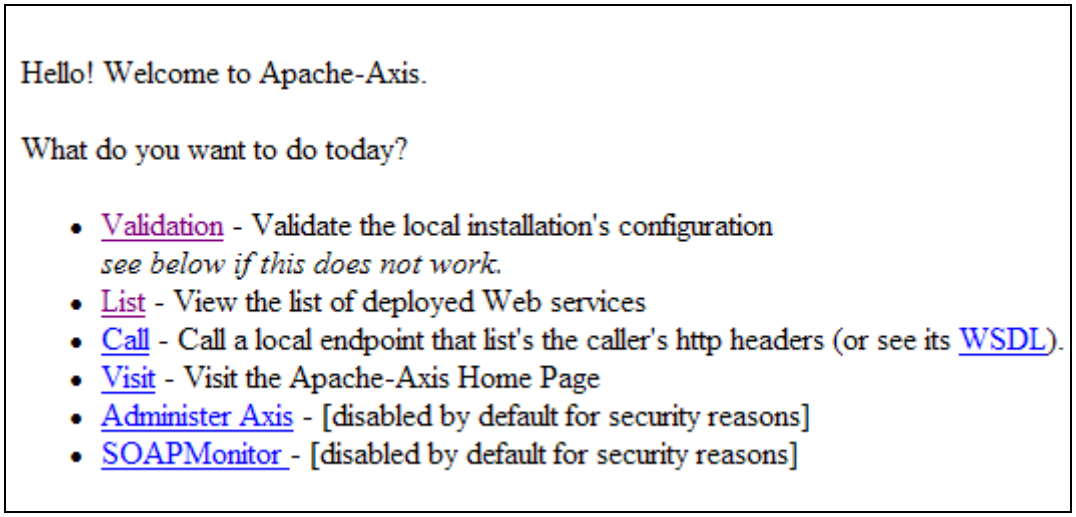
- los ficheros jar que se encuentran en el directorio lib de la distribución de Axis: axis-ant.jar, commons-logging-1.0.4.jar, axis.jar, jaxrpc.jar, saaj.jar, commons-discovery-0.2.jar, log4j-1.2.8.jar y wsdl4j-1.5.1.jar
- el fichero mail.jar
- el fichero activation.jar

Recuerde que para añadir los ficheros jar al classpath, puede modificar la variable de entorno CLASSPATH o simplemente utilizar la opción –classpath de los comandos java y javac.

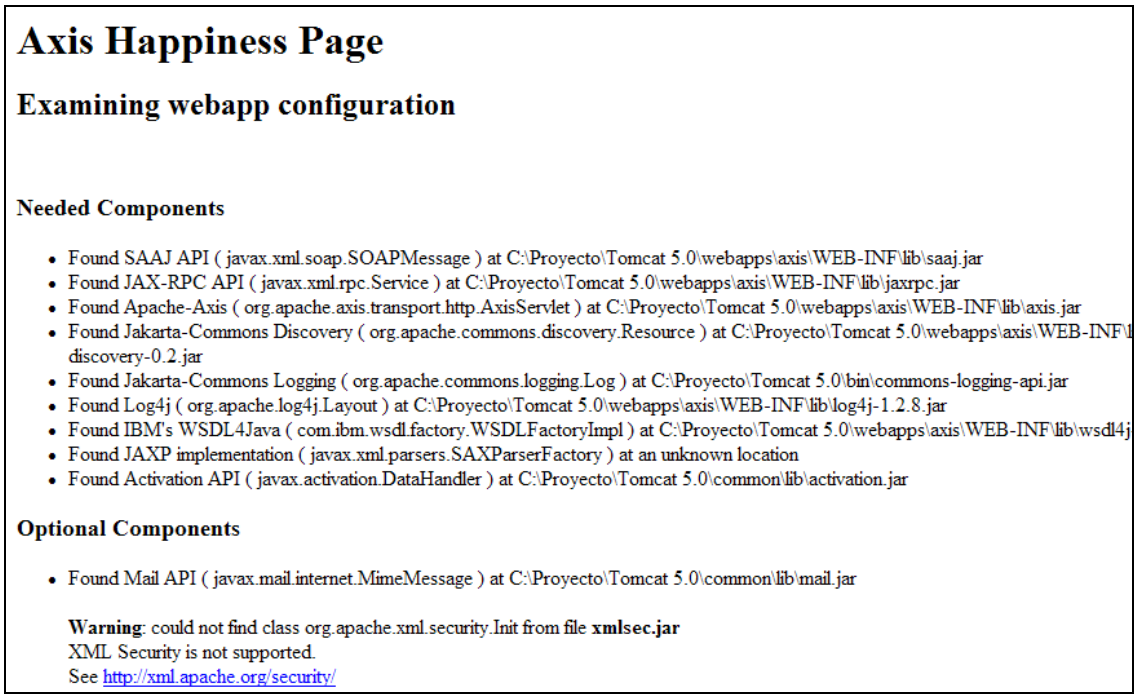
Para comprobar que Axis está correctamente instalado, realizamos la siguiente prueba:

1.- Arranca el Tomcat, ya sea desde el monitor o desde configure Tomcat (Start Tomcat)

2.- Conéctate desde el navegador a http://localhost:8080/axis



3.- Entra a Validation y comprueba que no hay errores (pueden parecer warning, pero no son de relevancia).



Axis Happiness Page

Examining webapp configuration

Needed Components

- Found SAAJ API (javax.xml.soap.SOAPMessage) at C:\Proyecto\Tomcat 5.0\webapps\axis\WEB-INF\lib\saaj.jar
- Found JAX-RPC API (javax.xml.rpc.Service) at C:\Proyecto\Tomcat 5.0\webapps\axis\WEB-INF\lib\jaxrpc.jar
- Found Apache-Axis (org.apache.axis.transport.http.AxisServlet) at C:\Proyecto\Tomcat 5.0\webapps\axis\WEB-INF\lib\axis.jar
- Found Jakarta-Commons Discovery (org.apache.commons.discovery.Resource) at C:\Proyecto\Tomcat 5.0\webapps\axis\WEB-INF\l
  discovery-0.2.jar
- Found Jakarta-Commons Logging (org.apache.commons.logging.Log) at C:\Proyecto\Tomcat 5.0\bin\commons-logging-api.jar
- Found Log4j (org.apache.log4j.Layout) at C:\Proyecto\Tomcat 5.0\webapps\axis\WEB-INF\lib\log4j-1.2.8.jar
- Found IBM's WSDL4Java (com.ibm.wsdl.factory.WSDLFactoryImpl) at C:\Proyecto\Tomcat 5.0\webapps\axis\WEB-INF\lib\wsdl4j
- Found JAXP implementation (javax.xml.parsers.SAXParserFactory) at an unknown location
- Found Activation API (javax.activation.DataHandler) at C:\Proyecto\Tomcat 5.0\common\lib\activation.jar

Optional Components

- Found Mail API (javax.mail.internet.MimeMessage) at C:\Proyecto\Tomcat 5.0\common\lib\mail.jar

Warning: could not find class org.apache.xml.security.Init from file **xmlsec.jar**
XML Security is not supported.
See http://xml.apache.org/security/

4.4.- Procedimiento de instalación

Una vez cumplidos todos los prerrequisitos, los pasos a seguir son muy sencillos:

- Con el servidor Tomcat parado, copiar la carpeta *elearning* en el directorio de Tomcat *webapps*

- Crear una base de datos en DB2 de nombre *elearn* que trabaje con XML: *CREATE DATABASE elearn USING CODESET UTF-8 TERRITORY es_ES COLLATE USING SYSTEM*

- Arrancar el servidor Tomcat y ejecutar en el navegador el jsp *instalacion/instalar.html* (*http://localhost:8080/elearning/instalacion/instalar.html*)

Introducir un nombre de usuario, contraseña, teléfono y dirección de correo electrónico para el super-administrador y pulsar *Instalar*.

5.- ESPECIFICACION: MANUAL DE USUARIO

Para acceder a la aplicación, basta con ejecutar en su navegador: *http://localhost:8080/elearning/login.html*, sustituyendo *localhost* por la dirección IP o DNS del servidor en caso de ejecutarse desde una máquina cliente distinta a la del servidor.

5.1.- Usuario ordinario

La primera vez que acceda a la aplicación, deberá registrarse como nuevo usuario, pulsando en *Registrar nuevo usuario.* En la pantalla de registro, deberá introducir un nombre de usuario y clave con los que se conectará al sistema, una confirmación de clave (obviamente igual al campo de clave), un teléfono de contacto (fijo o móvil) y una dirección de correo electrónico. Además, puede seleccionar un idioma para su perfil, que por defecto es el español. Cuando pulse en *Registrar*, se le presentará una pantalla con los datos que ha proporcionado, donde podrá *Cancelar* para modificar los datos o *Confirmar* el registro.

Una vez que el administrador haya confirmado su registro de nuevo usuario, usted podrá entrar en la aplicación, proporcionando el nombre y clave de usuario y pulsando en *Enviar.*

La pantalla inicial de la aplicación muestra un texto de bienvenida y la fecha de última actualización. En el marco izquierdo aparecen varias opciones sobre las que puede pulsar el usuario:

- Gestión de OA

  - Crear OA:

    Esta funcionalidad ha sido integrada a partir de una aplicación Web independiente de nuestro proyecto (Editor de objetos de aprendizaje online). Por ello, no entramos a explicar dicho apartado.

  - Enviar OA:

    En esta pantalla cualquier usuario puede enviar un OA guardado en su propio ordenador. Para ello debe introducir un nombre para ese objeto (procure dar un nombre identificativo, puesto que el sistema no realiza comprobación alguna sobre dicho nombre) y seleccionar el OA en cuestión en su máquina, mediante el botón *Examinar…*

    También se le permite seleccionar un esquema de validación sobre el cual desea validar el archivo de metadatos de su OA; si no desea realizar validación, seleccione la opción *(Ninguno)*, que aparece marcada por defecto.

Al pulsar en *Enviar* se registrará el nuevo OA en el sistema, siempre que el archivo de metadatos sea correcto, y se guardará una copia en su espacio de usuario. Este OA no aparecerá en ninguna búsqueda hasta que el administrador confirme su registro, lo cual puede comprobar en la sección *OA Enviados*.

- Búsqueda

  - Buscar:

    Al acceder a esta sección, se le presenta al usuario la pantalla de búsqueda rápida o simple. Aquí basta con escribir las palabras o expresiones a buscar, seleccionar el tipo de búsqueda mediante *Nodo Local* (buscar sólo en el servidor local) o *Nodo Global* (buscar en todos los servidores) y pulsar en *Buscar*. Si se escriben varias palabras, el resultado estará compuesto de todos los OA cuyos metadatos contengan alguna de las palabras especificadas. No se tienen en cuenta mayúsculas y minúsculas.

    La otra alternativa es pulsar en *Búsqueda avanzada*. En la pantalla que se presenta a continuación, el usuario puede escribir las palabras a buscar en el campo de texto de la parte superior, seleccionar el tipo de búsqueda y especificar el valor de todos o alguno de los campos que aparecen, para acabar clickeando en *Buscar*. El resultado en este caso será como el de la búsqueda simple, pero filtrando los resultados de acuerdo a las restricciones especificadas, teniéndose que cumplir todas ellas (es decir, se hace una lógica *AND* de las restricciones seleccionadas)

    Se puede realizar una búsqueda avanzada únicamente con el campo de búsqueda rápida, mediante un lenguaje que se ha implementado y cuya idea es similar al lenguaje que utiliza el buscador de *Google* (consultar Apéndice). Todo lo que permite la búsqueda avanzada está soportado también por la búsqueda simple, mediante el lenguaje de búsqueda apropiado. De igual forma, se pueden utilizar a la vez la búsqueda avanzada y la búsqueda simple, pudiendo ampliar así las condiciones de las restricciones sobre los resultados; en este caso, se haría una lógica *OR* de las distintas restricciones aplicadas sobre un mismo campo, y a la vez una lógica *AND* sobre las restricciones de distintos campos (como indica en el párrafo anterior)

    La pantalla de presentación de resultados es la misma, independientemente de si se usa una búsqueda u otra: nombre del OA y autor del mismo. Sobre cada uno de los OA encontrados se permiten las siguientes acciones:

- Vista previa:

  Muestra un resumen de los metadatos del OA.

  Permite la opción de volver, imprimir la vista en un archivo *PDF* (se guarda una copia en el espacio del usuario, pero éste puede guardarlo también en su propia máquina) y comentar el OA; en éste caso, aparece otra pantalla donde el usuario puede escribir sus comentarios y pulsar en *Guardar*, o cancelar pulsando en *Atrás.* Se debe tener en cuenta que un usuario no puede escribir más de un comentario sobre el mismo OA.

- Leer comentarios

  Presenta todos los comentarios asociados a ese OA, junto con el autor y la fecha de los mismos.

  Se puede crear un *PDF* con el contenido de la pantalla de la misma forma que la vista previa, o volver a la pantalla de búsqueda.

- Descargar

  Permite descargar en la máquina cliente el OA seleccionado

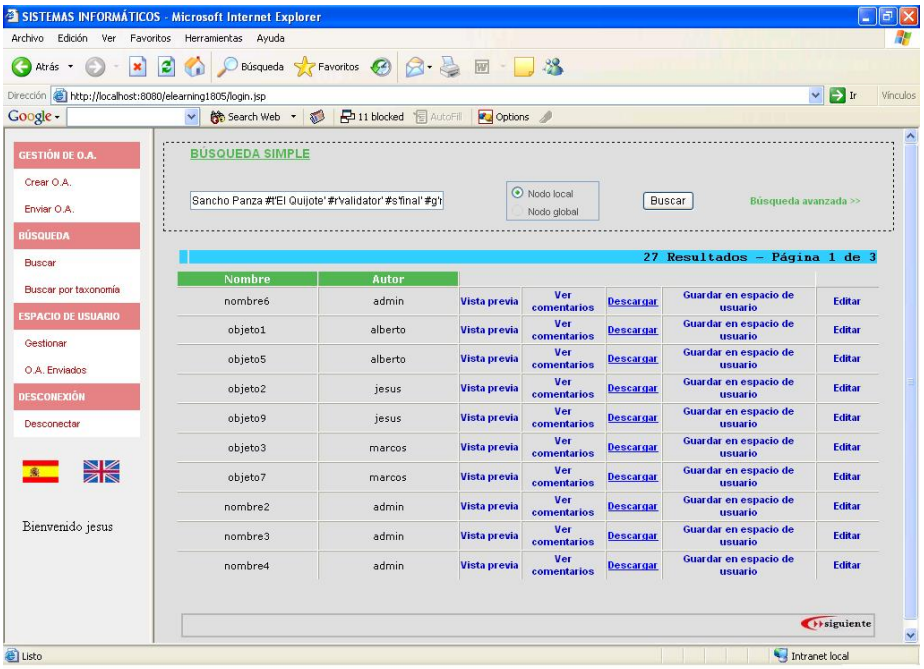
- Guardar en espacio de usuario

  Almacena una copia en el espacio del usuario

- Editar

  Permite editar el OA. Esta funcionalidad ha sido integrada a partir de una aplicación Web independiente de nuestro proyecto (Editor de objetos de aprendizaje online). Por ello, no entramos a explicar dicho apartado.

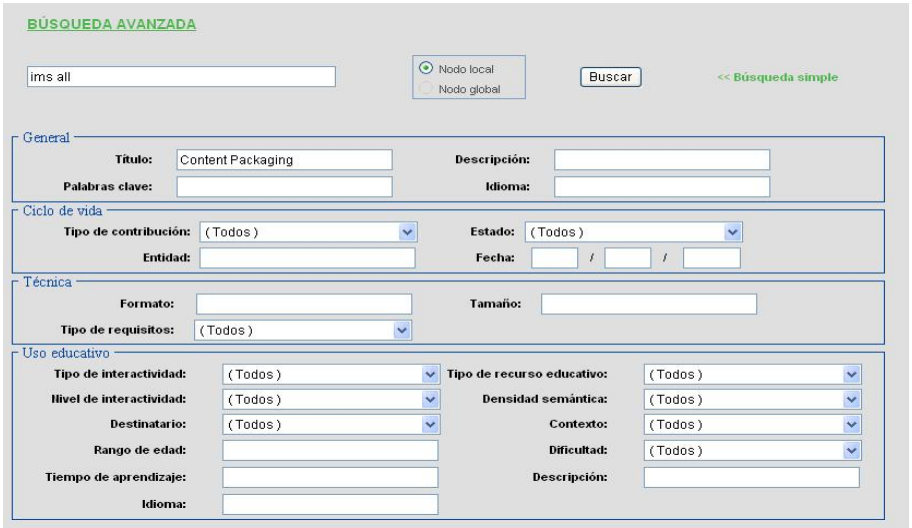
En la parte superior, se detalla el número total de resultados de la búsqueda. Si se producen más de 10 resultados, se realiza paginación de los mismos; es decir, se muestran en distintas páginas, de 10 en 10, informando en cada momento del número de página sobre el que se encuentra y permitiendo moverse adelante y atrás sobre la paginación.



Ejemplo:

Tenemos un único OA en el sistema, cuyo título es *IMS Content Packaging Sample - All Elements.*

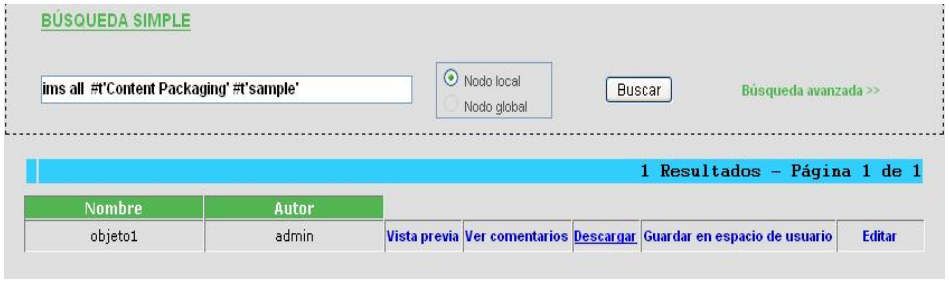
Realizamos una búsqueda avanzada en el Nodo Local, en la que buscamos las palabras *ims*, *all* y restringimos los resultados a aquellos que tengan en el título la expresión *Content Packaging*



La pantalla de resultados es la siguiente:

Como se puede observar, la restricción del título se ha convertido en el parámetro *#t 'Content Packaging'*, y no se ha presentado paginación puesto que sólo hay un resultado.

Podemos añadir más restricciones a la búsqueda simple mediante el lenguaje de búsqueda. Por ejemplo, añadimos otra restricción para que el título contenga la palabra *simple* y buscamos:



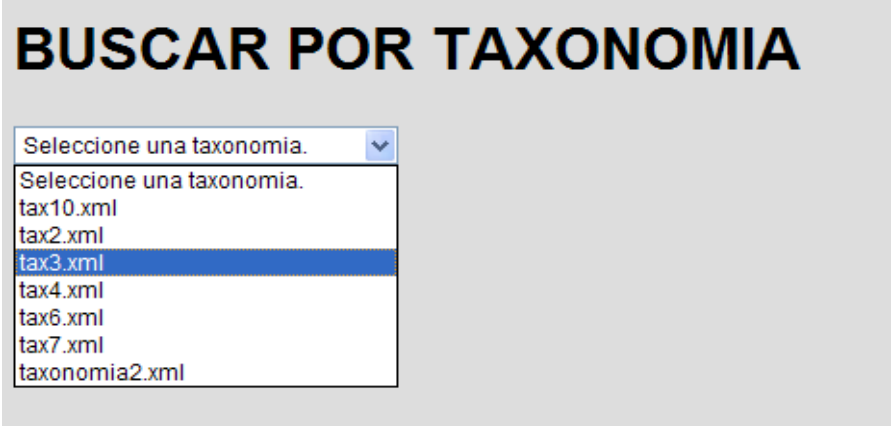
Podemos ver que el resultado es el mismo, ya que lo que ha hecho la aplicación es: buscar los OA que contengan las palabras *ims*, *all* y además su título contenga la expresión *Content Packaging* o la expresión *simple* (al menos una de las dos).

- Buscar por taxonomía:

Esta búsqueda se realiza sobre la categoría Clasificación del estándar LOM. Esta categoría puede contener varias clasificaciones para un mismo objeto. Cada clasificación se hace basándose en una taxonomía definida previamente. Estas taxonomías esta sujetas a un estándar perteneciente a IMS llamado IMS VDEX, que describe cómo se deben describir las taxonomías y otros métodos de clasificación. Cada nodo de la red tendrá un conjunto de taxonomías que el administrador del nodo se encargará de ir añadiendo o eliminando. Para realizar una búsqueda por taxonomía primero se selecciona la taxonomía por la cual se quiere realizar dicha búsqueda. Las taxonomías que contenga el nodo al que pertenece el usuario de la aplicación se

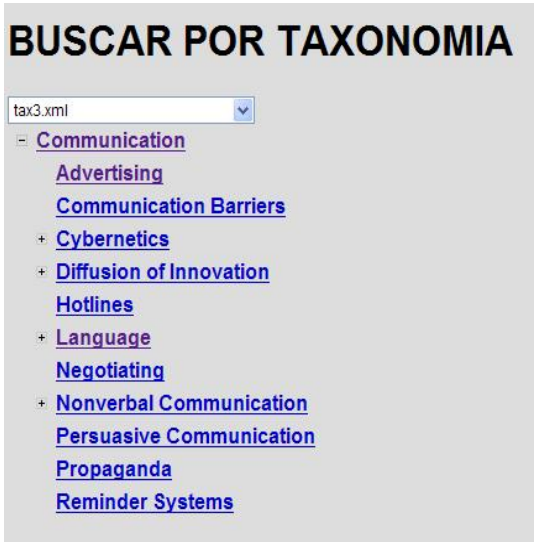
cargarán en un desplegable, el usuario seleccionará la que desee y se mostrará el árbol correspondiente a la taxonomía seleccionada.

Una vez seleccionada la taxonomía se muestra el árbol correspondiente, y el usuario deberá seleccionar uno nodo del árbol. Esto lanza una llamada de búsqueda, la cual se realizará sobre la categoría Clasificación de todos los objetos que contenga el nodo del usuario de la aplicación (esta búsqueda no se realiza sobre todos los nodos) comprobando que la ruta en el árbol del nodo pulsado se encuentra en esa categoría del LOM y que la taxonomía por la que se realiza la búsqueda es por la que esta clasificado el objeto.



Selección de la taxonomía por la que se quiere realizar la búsqueda

Por ejemplo, si seleccionamos la taxonomía "tax3.xml" se mostrará el siguiente árbol:



Arbol de la taxonomía mesh.xml

Si este árbol seleccionamos el nodo “Advertising” se buscará en los objetos la ruta “Comunnication/Advertising” y solo se devolverá como resultado aquellos objetos que este clasificados con la taxonomía “tax3” y tenga dicha ruta en la categoría clasificación del LOM.

Los resultados son los siguientes:

Resultado búsqueda por taxonomía

Como se puede observar en la figura anterior, la pantalla de resultados de la búsqueda por taxonomía es idéntica a la pantalla de la búsqueda expuesta anteriormente, por lo que no volveremos a explicar sus funcionalidades.

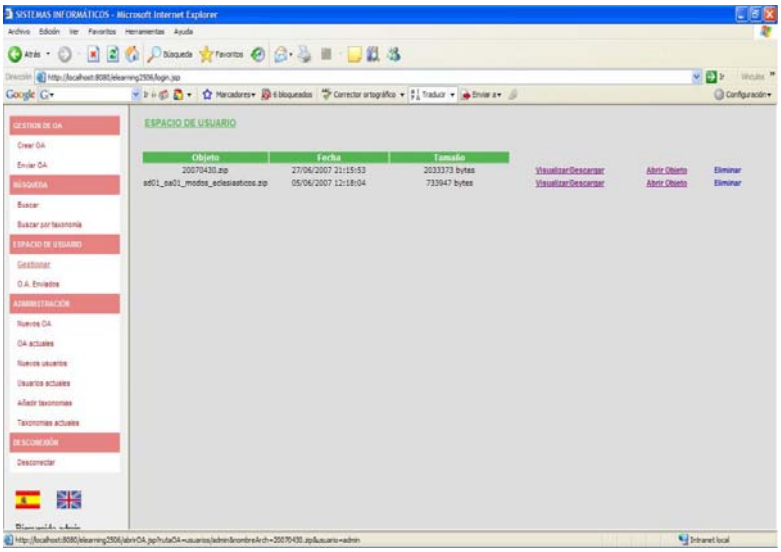
- Espacio de usuario

  - Gestionar

Al pulsar en esta opción, se mostrará información sobre los archivos presentes en su espacio de usuario. En concreto, podrá ver el nombre y extensión del archivo, la fecha de última modificación y el tamaño en bytes.

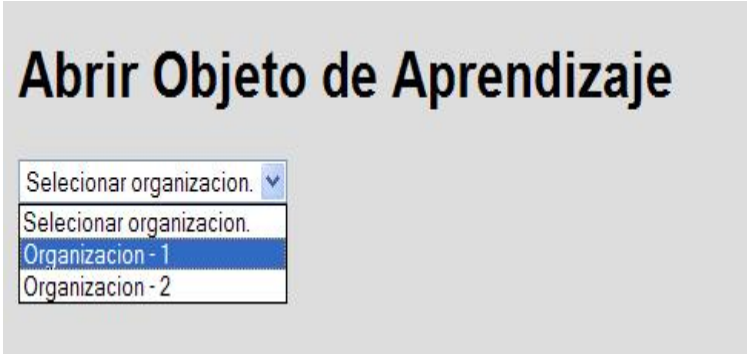
Además, se le permite visualizar el archivo si tiene formato de texto (pdf, doc, txt…) y/o descargarle si es de otro tipo (zip…), abrir el OA si el archivo es un OA (.zip) y eliminarle del espacio de usuario sea cual sea el tipo de archivo.

- AbrirObjeto



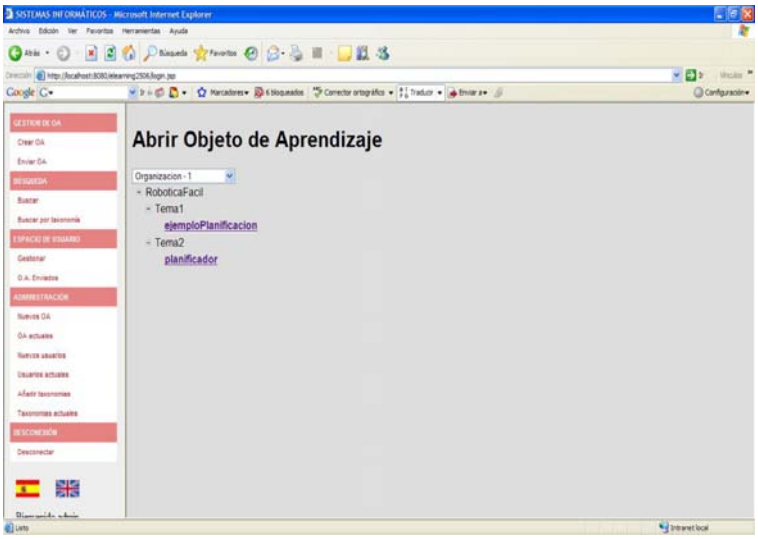
Gestión de espacio usuario.

Para la visualización de un objeto de aprendizaje se genera un árbol que muestra la organización de dicho objeto. Esta organización se obtiene del fichero imsmanifest.xml, que pertenece al estándar LOM, en el que se encuentra los metadatos del objeto, en este fichero puede venir definidas varias organizaciones para el objeto que se quiere visualizar, se cargan todas las organizaciones en un desplegable para que el usuario pueda seleccionar la organización del objeto que quiere mostrar, y una vez seleccionada una organización se mostrará el árbol que corresponde a dicha organización.



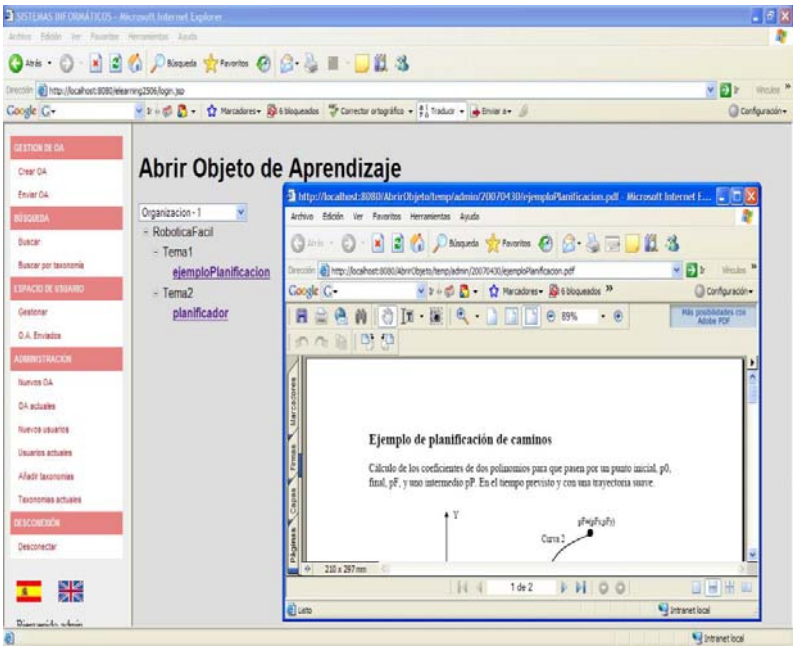
Selección organización.

Una vez seleccionada una organización de la que se ofrecen en el desplegable se mostrará el árbol que corresponde a dicha organización.



Seleccionada la organización, se muestra el árbol de contenidos.

Explorando a través del árbol se pueden encontrar los archivos que contiene el objeto y solamente pinchando sobre ellos se podrá visualizar el archivo que contenga el nodo seleccionado. Los nodos del árbol que aparezcan como un link contienen los archivos comentados anteriormente, estos archivos pueden ser una pagina html, un .pdf, un doc, una imagen o cualquier tipo de archivo que pueda visualizarse. En estos nodos se encuentra el contenido del objeto (lo que se quiere visualizar). Los demás nodos solo sirven para clasificar el contenido del objeto en temas y subtemas.



Visualización de un archivo de un objeto.

- OA Enviados

  En esta sección el usuario podrá consultar todos los OA que ha enviado desde que se registró por primera vez: el nombre del OA, la fecha en la que lo envió y un indicador de si dicho OA ha sido aprobado ya por el administrador.

o Desconexión

- Desconectar

  Aquí puede cerrar su sesión de una forma segura

o Cambio de idioma

En cualquier momento el usuario puede cambiar el idioma en el que se le mostrará la aplicación, pulsando sobre los iconos situados en la parte inferior del marco izquierdo. Este cambio de idioma se le aplicará siempre a partir de ese momento, hasta que vuelva a cambiarlo.

5.2.- Super-administrador

Este usuario se crea al instalar la aplicación, y no se permite su eliminación ni otorgar sus permisos a otros usuarios.

Tiene todas las opciones que se han explicado para un usuario ordinario, y además, algunas funciones propias de su nivel que aparecen en la zona de *Administración*. Cada nodo de la red tendrá su propio administrador con dichas funciones:

- Nuevos OA

  Muestra la lista de todos los OA que se han enviado y aún no se han confirmado. Para cada uno de ellos se puede ver el nombre y autor del OA, la fecha de envío y el teléfono y correo electrónico del autor. Asimismo, la aplicación permite *descargar* el OA en el cliente donde se conecte el administrador, *registrar*/validar el OA para que pase a formar parte de las búsquedas y/o *rechazar* el OA para eliminarlo del sistema.

- OA actuales

  Muestra la misma información que el apartado de *Nuevos OA*, con la diferencia de que aparecen los OA que ya han sido validados por el administrador en algún momento.

  Se permite también *descargar* el OA y *eliminar* lo del sistema.

- Nuevos usuarios

  Muestra la lista de los usuarios que se han registrado en la aplicación, pero aún no han sido validados por el administrador, junto con sus teléfonos y direcciones de correo electrónico.

  El administrador puede, con cada uno de ellos, *dar de alta* en el sistema o *rechazar* para eliminarle del sistema

- Usuarios actuales

  Muestra la lista de los usuarios ya validados por el administrador, junto con su teléfono, dirección de correo electrónico y tipo de usuario (usuario común o administrador). En esta pantalla no aparece el super-administrador, ya que éste es el único usuario que no se puede eliminar.

  Cada uno de ellos se puede eliminar del sistema, por lo que es muy importante tener en cuenta que se elimina también su espacio de usuario, sus comentarios sobre OA y los OA enviados por él.

- Añadir taxonomías

  En esta pantalla el administrador puede añadir nuevas taxonomías de búsqueda en el sistema, estas taxonomías la tendrá el administrador en su propio ordenador o cualquier dispositivo de almacenamiento. Para ello debe introducir un nombre para la taxonomía (el sistema comprobara si existe otra taxonomía guardada con ese mismo nombre y le avisará en caso afirmativo) y seleccionar el archivo (el sistema también comprobara si el archivo seleccionado ya está dentro de él) que describe la taxonomía en cuestión en su máquina, mediante el botón *Examinar…*

  Al pulsar en *Enviar* la taxonomía pasará a formar parte del sistema. La nueva taxonomía aparecerá en la pantalla *Taxonomías actuales* que se describe a continuación y en la lista de taxonomía de la pantalla de *Buscar Por Taxonomía* descrita anteriormente.

- Taxonomías actuales

  Muestra la lista de las taxonomías disponibles en el sistema mediante las cuales podemos realizar búsquedas de OA.

  Además, se permite para cada una de ellas *visualizar* el contenido del fichero ".xml" que la describe y *eliminar* dicho fichero del sistema y por tanto la taxonomía.

6.- IMPLEMENTACION

6.1.- Arquitectura de la aplicación

La aplicación está constituida de varios archivos *.html* y *.jsp* presentes en el directorio raíz, así como otras carpetas:

- *archivos*

  Es el directorio físico donde se almacenan los OA del sistema. Por cada OA que se envía, se crea una carpeta con un nombre único (la concatenación del nombre del OA con la fecha de envío en milisegundos) que alberga dicho OA.

- *estilos*

  Contiene las dos plantillas de estilos utilizadas, una para la pantalla de Login (*estilosLogin.css*) y otra para el resto de ficheros *.html* y *.jsp* (*estilos.css*)

- *idiomas*

  Aquí se encuentran los ficheros de inicialización de idiomas: *english.ini* y *spanish.ini*

- *imágenes*

  Contiene los iconos e imágenes utilizadas en las distintas pantallas de la aplicación: idiomas, pdf…

- *instalacion*

  En este directorio se encuentran los archivos encargados de la instalación de la aplicación: *instalar.html* e *instalar.jsp*

- *scripts*

  Contiene los archivos de Javascript que se utilizan en los *.html* y *.jsp* de la aplicación:

  - *scriptAux.js*

    Define funciones auxiliares que se utilizan en varios de los demás archivos Javascript: quitar espacios en un String, comprobar si una fecha es correcta, comprobar si un String contiene un número real…

  - *scriptBuscar.js*

Define las funciones que se utilizan en las pantallas de búsqueda simple y búsqueda avanzada.

- *scriptBuscarPorTax.js*

  Define las funciones que se utilizan para la búsqueda por taxonomía.

- *scriptNuevosOA.js*

  Define las funciones utilizadas en las pantallas de envío de OA, consulta de nuevos OA y consulta de OA actuales.

- *scriptNuevasTaxonomias.js*

  Define las funciones utilizadas en las pantallas de añadir taxonomía y taxonomías actuales.

- *scriptNuevosUsuarios.js*

  Define las funciones utilizadas para dar de alta, rechazar y eliminar un usuario por parte del administrador

- *scriptRegistrar.js*

  Define las funciones utilizadas en la pantalla de registro de nuevo usuario, para comprobar que se han rellenado todos los campos del formulario, se ha introducido un número de teléfono y una dirección de correo electrónico válidos y los campos *clave* y *confirmación de clave* coinciden.

- *scriptUserspace.js*

  Define las funciones utilizadas en la pantalla de gestión del espacio de usuario

- *scriptVistaPrevia.js*

  Define las funciones utilizadas en las pantallas de vista previa, escribir comentarios y leer comentarios de un OA

- *taxonomias*

  Contiene los archivos de taxonomías utilizados en la búsqueda por taxonomía.

- *temporal*

Este directorio se utiliza para determinadas acciones que requieren la creación y eliminación de archivos temporales.

- *usuarios*

Aquí es donde se crean los espacios de usuario, que básicamente son directorios cuyo nombre es el nombre del usuario. Al instalar la aplicación, se crea el espacio de usuario del super-administrador, y no se permite eliminarle.

- *WEB-INF/classes/com*

Contiene las clases Java con las que Google Web Toolkit (GWT) ha generado los archivos *.html* , *.js* y todo lo necesario para generar una aplicación AJAX mediante GWT.

- *WEB-INF/classes/si*

Contiene las clases Java que se utilizan en los *.jsp* de la aplicación, y que forman parte del paquete *si.*

Por otra parte, podemos agrupar algunos *.jsp* y *.html* según la funcionalidad que desempeñan:

- Registro de nuevo usuario: *registro.html*, *confirmacionRegistro.jsp* y *registro.html*
- Login: *login.html* y *login.jsp*
- Página principal: *ilearningN.jsp*, que se compone a su vez de 2 marcos: *marcoLateral.jsp* y *marcoMain.jsp*
- Envio de OA: *enviarOA.jsp* y *enviarOAFin.jsp*
- Búsqueda de OA: *buscarSimple.jsp*, *buscarAvanzado.jsp*.
  - Vista previa: *vistaPrevia.jsp*, *vistaPreviaPDF.jsp*
  - Ver comentarios: *verComentarios.jsp*, *verComentariosPDF.jsp*
  - Escribir comentarios: *escribirComentarios.jsp*, *guardarComentarios.jsp*
  - Guardar en espacio de usuario: *guardar.jsp*
- Búsqueda por taxonomía: *buscarPorTaxonomia.jsp*
  - Aplicación web dinámica AJAX generada con GWT:
    - *0C7EC1E68F568C73D2D5FB49DE3BC966.cache.html*
    - *0C7EC1E68F568C73D2D5FB49DE3BC966.cache.xml*
    - *5FFFBE5A1152B8A6FCD4C61312DF7242.cache.html*
    - *5FFFBE5A1152B8A6FCD4C61312DF7242.cache.xml*
    - *8DA72A2E98FC765761F890FC1826F5AD.cache.html*
    - *8DA72A2E98FC765761F890FC1826F5AD.cache.xml*
    - *12376E87AD3239011DB7369C44128559.cache.html*
    - *12376E87AD3239011DB7369C44128559.cache.xml*
    - *A7A547015359B34F669624427A41757F.cache.html*
    - *A7A547015359B34F669624427A41757F.cache.xml*
    - *com.ArbolXML.ArbolXML.nocache.html*
    - *ArbolXML.jsp*

        - *gwt.js*
        - *history.html*
    - Guardar en espacio de usuario: *guardar.jsp*

- Gestionar espacio de usuario: *userspace.jsp, borrarUserspace.jsp, abrirOA.jsp*
    - Aplicación web dinámica AJAX generada con GWT para “AbrirObjeto”:
        - *1BF639D9FE944E822709D1C83A23D2ED.cache.html*
        - *1BF639D9FE944E822709D1C83A23D2ED.cache.xml*
        - *0337F5AA243B802F2018BBE8F9437DBB.cache.html*
        - *0337F5AA243B802F2018BBE8F9437DBB.cache.xml*
        - *B6EE9C8FAB06C6745B31F1AB4EA2BD3A.cache.html*
        - *B6EE9C8FAB06C6745B31F1AB4EA2BD3A.cache.xml*
        - *9691F178B00FADAF54AFFE2F15970746.cache.html*
        - *9691F178B00FADAF54AFFE2F15970746.cache.xml*
        - *ECFF35DD534A0214835C129D77A489F3.cache.html*
        - *ECFF35DD534A0214835C129D77A489F3.cache.xml*
        - *com.AbrirOA.AbrirObjeto.nocache.html*
        - *AbrirObjeto.jsp*
        - *gwt.js*
        - *history.html*

- OA Enviados: *enviadosOA.jsp*
- Nuevos OA: *nuevosOA.jsp, registrarOA.jsp*
- OA actuales: *actualesOA.jsp, registrarOA.jsp*
- Nuevos usuarios: *adminNuevos.jsp, altaUsuario.jsp*
- Usuarios actuales: *adminUsuarios.jsp, altaUsuario.jsp*

- Añadir taxonomías: *anadirTaxonomia.jsp, anadirTaxonomiaFin.jsp*
- Taxonomías actuales: *taxonomíasActuales.jsp, eliminarTaxonomia.jsp*
- Desconectar: *logout.jsp*
- Cambio de idioma: *cambioIdioma.jsp*

6.2.- Clases Java: paquete si

El paquete *si* se compone de las siguientes clases:

- *AtrVal.java*

  Consta de una variable String y un array de Strings, y sirve para guardar cada campo especificado en una búsqueda de OA con los valores dados para él.

- *Auxiliar.java*

  Implementa funciones auxiliares cuya funcionalidad no encaja en las demás clases de este paquete:
  - *formatearError*: da formato a la cadena de error producida en el acceso a base de datos.
  - *modificarXML*: devuelve el contenido del archivo de metadatos de un OA con algunas modificaciones para poder utilizarse debidamente en la base de datos.

- *Buscador.java*

  Implementa todas las funciones necesarias para poder realizar la búsqueda, tanto la concatenación como el tratamiento de campos como la búsqueda en sí.

- *CampVal.java*

  Consta de dos variables String, y sirve para guardar cada campo seleccionado del formulario de búsqueda avanzada con el valor especificado para él.

- *ConjuntoResults.java*

  Representa un elemento resultante de una búsqueda de OA, formado por nombre del OA, autor del OA, código de OA y ruta física del OA.

- *Db.java*

  Comprende las funciones necesarias para el acceso y modificación de la base de datos a bajo nivel: conexión, desconexión, ejecución de sentencias de selección, inserción, actualización y borrado de filas y sentencias de creación y eliminación de tablas.

  Esta clase ofrece métodos de manejo de la base de datos a las clases *Buscador.java* y *SentenciasSQL.java* y al fichero de instalación *instalar.jsp*.

- *Idioma.java*

  Lee el fichero de inicialización correspondiente y almacena las frases en un array estático, de tal forma que se carga una sola vez y se utiliza en todos los archivos *.jsp* de la aplicación.

- *SentenciasSQL.java*

  Implementa todas las funciones Java que se usan explícitamente en los archivos *.jsp* para acceder o modificar la base de datos: registro de nuevo usuario, alta de usuario, baja de usuario, consulta de usuarios, login, cambio de idioma, alta de OA, baja de OA, consulta de OA, búsqueda, vista previa…

- *Userspace.java*

  Ofrece métodos para gestionar el espacio de usuario: crear y borrar el espacio de usuario, eliminar un fichero del espacio de usuario, ver espacio de usuario…; y también para eliminar el espacio físico de un OA.

- *Zipeado.java*

  Implementa los métodos relacionados con el tratamiento de ficheros *.zip*, entre los que destacan:
  - *leerXML*: lee y devuelve el contenido del archivo de metadatos de un OA.
  - *guardarZip*: crea un directorio único en la carpeta */archivos/* y almacena en él un nuevo OA.

  Estos dos métodos se usan cuando un usuario envía un OA.

6.3.- Casos de uso principales

A continuación se detallan las acciones realizadas y los archivos y clases implicados en la ejecución de los principales casos de uso de la aplicación.

Con el fin de esquematizar dichos casos de uso en la medida de lo posible, no se detallan la comprobación de existencia de atributos de sesión (realizada en todos los *.jsp* a partir de la página principal) ni la conexión y desconexión a la base de datos (realizadas en todos los *.jsp* que utilizan la clase *SentenciasSQL* antes de utilizar cualquier otro método de acceso a DB2):

- Registro como nuevo usuario

1. El usuario pulsa en *Registrar nuevo usuario* en la pantalla de Login (*login.html*) → Se redirecciona a *registro.html.*

2. El usuario rellena todo o parte del formulario

    a) Pulsa en *Cancelar* → el navegador vuelve a la página anterior de Login

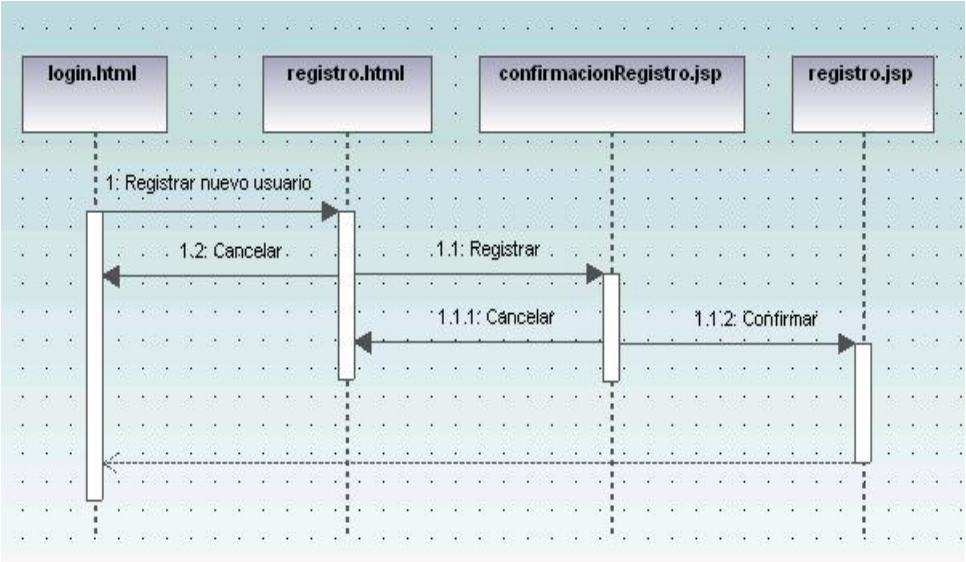
    b) Pulsa en *Registrar* sin haber rellenado todos los campos o con errores en alguno de ellos → se invoca al script *registrar()* del fichero *scriptRegistrar.js*, el cual muestra una ventana informando del error cometido (Paso 2)

    c) Pulsa en *Registrar* y todo es correcto → se invoca al script *registrar()* del fichero *scriptRegistrar.js*, el cual comprueba que todo está correcto y envía el formulario a *confirmacionRegistro.jsp* (Paso 3)

3. Se muestra un resumen de los datos proporcionados en el paso anterior, mediante *confirmacionRegistro.jsp*.

    a) El usuario pulsa *Cancelar* → se vuelve a la página anterior (Paso 2)

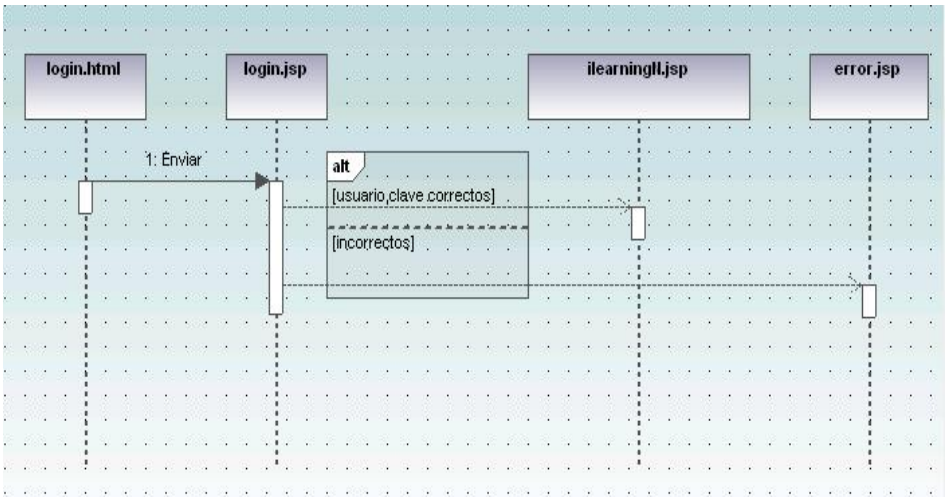
    b) El usuario pulsa *Confirmar* → se envían los datos a *registro.jsp*, donde se registra el nuevo usuario mediante el método *registro()* de la clase *SentenciasSQL* y se muestra un mensaje indicando si se ha podido realizar correctamente. Cuando el usuario pulsa *Aceptar*, se vuelve a la página de Login



- Login

1. El usuario introduce usuario y contraseña y pulsa en *Enviar* → se envían los datos a *login.jsp*, donde se comprueba si existe ese usuario con esa contraseña, mediante el método *login()* de la clase *SentenciasSQL*
    a) Si existe y es correcto: se crean atributos de sesión (usuario, clave, admin y ruta); se carga el fichero de idioma según el perfil asociado al usuario, mediante la clase *Idioma*; y se redirecciona a la página principal de la aplicación: *ilearningN.jsp*, compuesta por 2 marcos: uno fijo (*marcoLateral.jsp*) y otro dinámico, donde se muestra la pantalla de bienvenida (*marcoMain.jsp*)

    b) Si no: se redirecciona a la página de error *error.jsp*, donde se muestra un mensaje informativo.



- Enviar OA

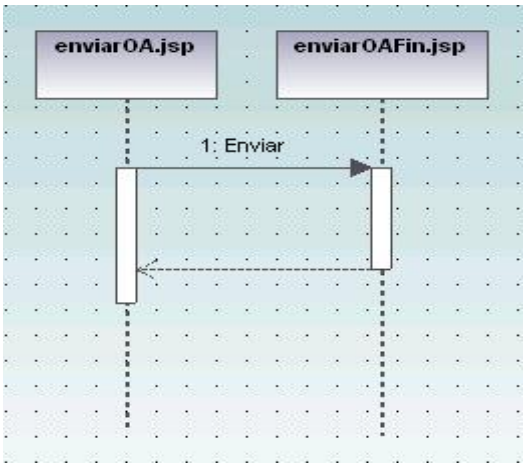
1. El usuario pulsa en *Enviar OA*, situado en el marco izquierdo → se muestra en el marco derecho la pantalla *enviarOA.jsp*.

2. El usuario introduce los datos solicitados y pulsa en *Enviar* → el script *enviarOA()* del fichero *scriptNuevosOA.js* comprueba que se han introducido todos los datos

   a) Si falta algún dato por rellenar, se muestra una pantalla de error (Paso 2)

   b) Si no, se envían los datos a *enviarOAFin.jsp*, donde se realizan las siguientes acciones:
      - Se guarda el fichero enviado en el espacio del usuario
      - Se guarda el contenido del archivo de metadatos del OA (método *leerXML()* de la clase *Zipeado*), se modifica para adecuarlo a DB2 (método *modificarXML()* de la clase *Auxiliar*) y se inserta en la base de datos con la fecha actual (método *insertarXML()* de la clase *SentenciasSQL*).
      - Creamos un directorio en la carpeta */archivos/* y guardamos el OA dentro de él, mediante el método *guardarZip()* de la clase *Zipeado*

   Por último, volvemos a *enviarOA.jsp*, mostrando un mensaje con información sobre cómo ha resultado el proceso



- Búsqueda rápida

1. El usuario pulsa en *Buscar* en el marco izquierdo → se muestra en el marco derecho la pantalla *buscarSimple.jsp*

2. El usuario introduce la palabra o palabras a buscar, selecciona *Nodo local / Nodo global* y pulsa en el botón Buscar → el script

buscarSimple() del fichero *scriptBuscar.js* recarga la página enviando los datos de búsqueda.

El método *TratarCadena()* de la clase *Buscador.java* devuelve un *ArrayList* con los campos y valores a buscar a partir de la cadena de búsqueda

a) Si el usuario ha seleccionado *Nodo local:*
   - Se invoca al método *contadorBusqueda()* de la clase *SentenciasSQL.java* para averiguar el número de OA del propio nodo que contienen, al menos, alguna de las palabras a buscar.
   - Acto seguido, se invoca al método *busquedaSimple()* de la misma clase para guardar en un *ArrayList* los datos de los primeros 10 OA resultantes de dicha búsqueda, ordenados por su código.
   - Por último, se recorre el *ArrayList* y se muestran los datos de esos OA. También se indica el número de resultados y el número de páginas en las que se pueden mostrar. En caso de que haya más de una página, se mostrará el botón *Siguiente.*

b) Si el usuario ha seleccionado *Nodo global*:
   - Se invoca al método *busquedaGlobal()* de la clase *SentenciasSQL.java*, el cual invoca a su vez al método *busquedaSimple()* para buscar en el nodo local, y a los Servicios Web del resto de nodos asociados para buscar en todos ellos. Así ya se tiene un array con todos los OA encontrados.
   - Por último, se muestran los 10 primeros OA del array resultante, indicando también el nº de resultados de la búsqueda, el nº de páginas en las que se mostrarán y, en caso de que éste sea mayor de uno, el botón *Siguiente*.

3. Si el usuario pulsa en Siguiente, se recargará la página mostrando los siguientes 10 resultados.

   a) En la búsqueda en *Nodo local*, ya tenemos el nº de resultados y por tanto, sólo se necesita invocar a *busquedaSimple()* para hallar los siguientes 10 OA.

   b) En la búsqueda en *Nodo global*, ya tenemos todos los OA resultantes almacenados en un array (variable estática de *SentenciasSQL.java*), por lo que basta con recorrer las siguientes 10 posiciones del array.

Ahora se mostrará el botón *Atrás* (vuelve a la pantalla anterior) y, si hay más páginas, el botón *Siguiente.*

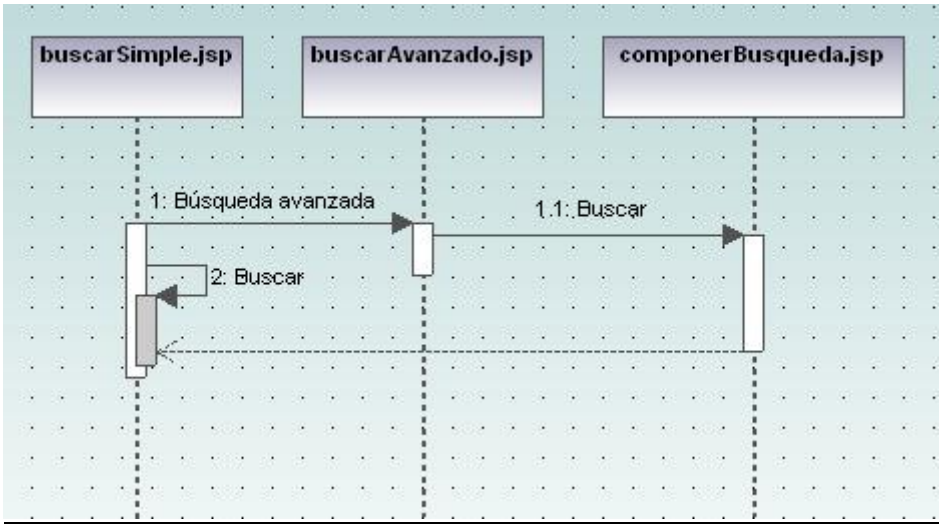
- Búsqueda avanzada

1. El usuario pulsa en *Buscar* en el marco izquierdo → se muestra en el marco derecho la pantalla *buscarSimple.jsp*

2. El usuario pulsa en *Búsqueda avanzada* → se muestra la pantalla *buscarAvanzado.jsp*

3. El usuario introduce una o varias palabras a buscar, especifica los valores de algunos o todos los campos según desee, selecciona *Nodo local/Nodo global* y pulsa en *Buscar* → se envía el formulario a *componerBusqueda.jsp*, donde se realizan las siguientes acciones:
    - Se invoca al método *componerBusqueda()* de la clase *SentenciasSQL.java* para componer un *ArrayList* con los campos y valores introducidos en la búsqueda
    - El método *TratarCampos()* de la clase *Buscador.java* genera una sentencia de búsqueda rápida a partir de dicho *ArrayList*
    - Se envía la sentencia a la página *buscarSimple.jsp*

    A partir de aquí, el proceso es el mismo que cuando se realiza una búsqueda rápida.



- Buscar por taxonomía

1. El usuario pulsa *Buscar por taxonomía* en el marco izquierdo → se carga *ArbolXML.jsp* en el marco derecho, se muestra un desplegable que contiene todas las taxonomías disponibles en el sistema, si el usuario selecciona cualquiera de ella se muestra el árbol perteneciente a la taxonomía seleccionada y haciendo click sobre los nodos del árbol en los que se permita se redirige a *buscarPorTaxonomia.jsp* pasando los parámetros necesarios para realizar la búsqueda, que en este caso son el

identificador del nodo seleccionado y el titulo de la taxonomía por la que estamos buscando. En *buscarPorTaxonomia.jsp* se realiza la búsqueda mediante el método *busquedaPorTaxonomia()* de la clase *SentenciasSQL* y se muestran los resultados obtenidos por dicha búsqueda. La pantalla de resultados es igual que la el resto de búsqueda permitidas en el sistema, por tanto, contiene los casos de usos "Vista Previa", "Ver Comentarios", "Descargar OA", "Guardar OA en espacio de usuario" y "Editar OA" que se explicarán a continuación.

2. *ArbolXML* es la página principal de una aplicación web dinámica basada en tecnología AJAX generada a partir del framework GWT. (Ver Proyecto GWT para Busqueda por taxonomía)

- Vista previa

1. Una vez realizada una búsqueda, el usuario pulsa en *Vista previa* → el script *vistaPrevia()* del fichero *scriptBuscar.js* redirecciona a la pantalla *vistaPrevia.jsp,* que utiliza el método *vistaPrevia()* de la clase *SentenciasSQL*.

Dentro de este caso de uso existen otros subcasos:

- Imprimir vista previa

2. El usuario pulsa el icono de exportar a PDF → se invoca a la función *exportarPDFPreview()* del fichero *scriptVistaPrevia.js*, la cual realiza una redirección a *vistaPreviaPDF.jsp*. Se usa de nuevo el método *vistaPrevia()* de la clase *SentenciasSQL* para generar un fichero PDF y mostrarlo directamente sobre el navegador

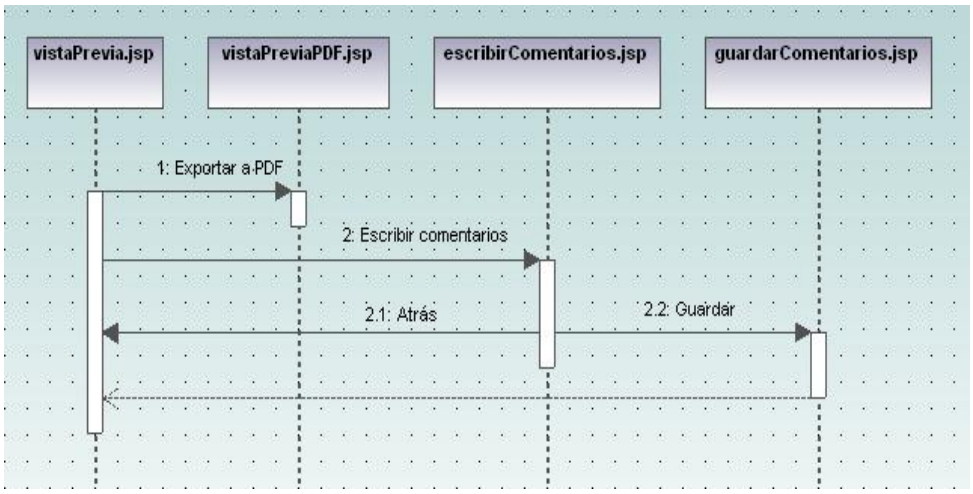
- Escribir comentario

2. El usuario pulsa el icono de escribir comentario → se invoca a la función *escribirComentarios()* del fichero *scriptVistaPrevia.js*, la cual redirecciona a *escribirComentarios.jsp*.

3. Se presenta la pantalla de Escribir comentarios.

   a. El usuario pulsa *Atrás* → se vuelve a la vista previa (Paso 2)

   b. El usuario pulsa *Guardar* → el script *guardarComentarios()* del fichero *scriptVistaPrevia.js* redirecciona a *guardarComentarios.jsp.* Se guardan los comentarios en la base de datos, mediante el método *guardarComentarios()* de la clase *SentenciasSQL* y se vuelve a la vista previa, informando del proceso a través de una ventana



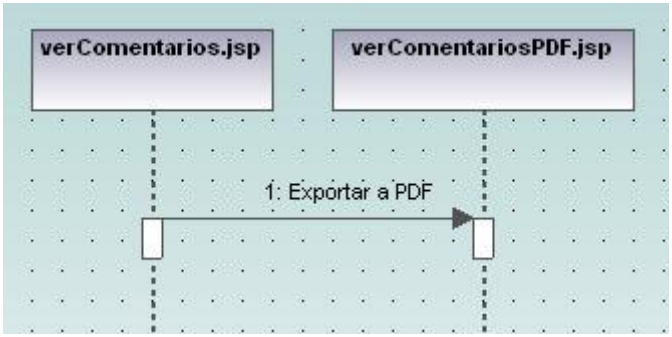
- Ver comentarios

1. Una vez realizada una búsqueda, el usuario pulsa en *Ver comentarios* → el script *verComentarios()* del fichero *scriptBuscar.js* redirecciona a la pantalla *verComentarios.jsp,* que utiliza el método *verComentarios()* de la clase *SentenciasSQL.*

Dentro de este caso de uso existe otro subcaso:

- Imprimir comentarios

2. El usuario pulsa el icono de exportar a PDF → se invoca a la función *exportarPDFComentarios()* del fichero *scriptVistaPrevia.js,* la cual realiza una redirección a *verComentariosPDF.jsp.* Se usa de nuevo el método *verComentarios()* de la clase *SentenciasSQL* para generar un fichero PDF y mostrarlo directamente sobre el navegador.



- Descargar OA de una búsqueda

1. Una vez realizada una búsqueda, el usuario pulsa *Descargar* → el navegador muestra una ventana de descarga

- Guardar OA de una búsqueda en el espacio de usuario

    1. Una vez realizada una búsqueda, el usuario pulsa *Guardar en espacio de usuario* → el script *guardarUsuario()* del fichero *scriptBuscar.js* pide confirmación mediante una ventana informativa:

        a. El usuario pulsa *Cancelar* → se vuelve a la pantalla anterior (Paso 1)

        b. El usuario pulsa *Aceptar* → se redirecciona a *guardar.jsp*. Se copia dicho OA en el espacio de usuario mediante el método *guardarEspacioUsuario()* de la clase *Userspace* y se vuelve a *buscarSimple.jsp,* informando del proceso.

- Ver espacio de usuario

    1. El usuario pulsa en *Gestionar* en el marco izquierdo → se muestra en el marco derecho la pantalla *userspace.jsp* con el contenido del espacio de usuario, para lo cual se utilizan los métodos *verEspacioUsuario(), verNombres(), verFechas()* y *verTamanyos()* de la clase *Userspace*

Dentro de este caso de uso se pueden distinguir otros subcasos:

- Eliminar fichero del espacio de usuario

    2. El usuario pulsa *Eliminar* en la pantalla de espacio de usuario → la función *eliminar()* del fichero *scriptUserspace.js* pide confirmación mediante una ventana informativa:

        a) El usuario pulsa *Cancelar* → se vuelve a la pantalla anterior (Paso 2)

        b) El usuario pulsa *Aceptar* → se redirecciona a *borrarUserspace.jsp* pasando el nombre del fichero a eliminar como parámetro. Se elimina dicho fichero mediante el método *eliminar()* de la clase *Userspace* y se vuelve a *userspace.jsp*

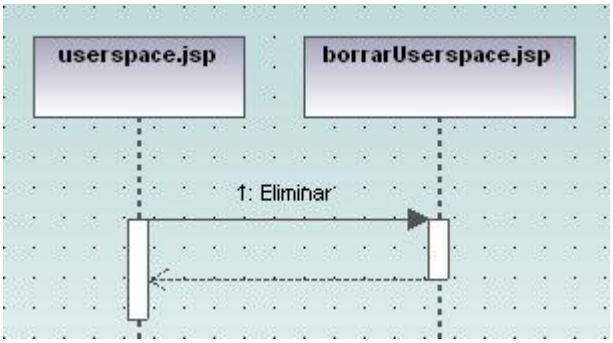
- Abrir Objeto de Aprendizaje

    3. El usuario pulsa *AbrirObjeto* en la pantalla de espacio de usuario → se carga *abrirOA.jsp* en el marco derecho, crea los parámetros necesarios para abrir el OA, como pueden ser la ruta donde se encuentra el objeto, así como el nombre del OA y el usuario que lo quiere abrir, redirige a *AbrirObjeto.jsp*, aquí se muestra un desplegable que contiene las organizaciones que se disponen para este OA, si el usuario selecciona cualquiera de ella se muestra el árbol perteneciente a la organización seleccionada y haciendo click sobre los nodos del árbol en los que se permita, se mostrará (en otra ventana del explorador) el contenido que el objeto tiene en ese nodo.

4. *AbrirObjeto.jsp* es la página principal de una aplicación web dinámica basada en tecnología AJAX generada a partir del framework GWT. (Ver Proyecto GWT para Abrir Objeto de Aprendizaje)

- Visualizar/Descargar fichero del espacio de usuario

1. El usuario pulsa *Visualizar/Descargar* en la pantalla de espacio de usuario → el navegador muestra una pantalla de descarga si el archivo es de tipo *.zip* o lo visualiza dentro de la propia aplicación si es de tipo *.pdf* (dependiendo del navegador utilizado, abrirá el archivo en la ventana actual o en una nueva).



o Ver OA enviados por un usuario

1. El usuario pulsa en *OA Enviados* en el marco izquierdo → se muestra en el marco derecho la pantalla *enviadosOA.jsp*, en la que se muestran los OA enviados por dicho usuario, mediante el método *dameOAUsuario()* de la clase *SentenciasSQL*

o Ver nuevos OA

1. El usuario pulsa en *Nuevos OA* en el marco izquierdo → se muestra en el marco derecho la pantalla *nuevosOA.jsp* con la lista de nuevos OA enviados, para lo cual se utiliza el método *dameOA()* de la clase *SentenciasSQL*

Dentro de este caso de uso se pueden distinguir otros subcasos:

- Descargar OA nuevo

2. El usuario pulsa *Descargar* en la pantalla de nuevos OA → el navegador muestra una pantalla de descarga.

- Registrar OA nuevo

3. El usuario pulsa *Registrar* en la pantalla de nuevos OA → la función *registrarOA()* del fichero *scriptNuevosOA.js* pide confirmación mediante una ventana informativa:

    a. El usuario pulsa *Cancelar* → se vuelve a la pantalla anterior (Paso 3)

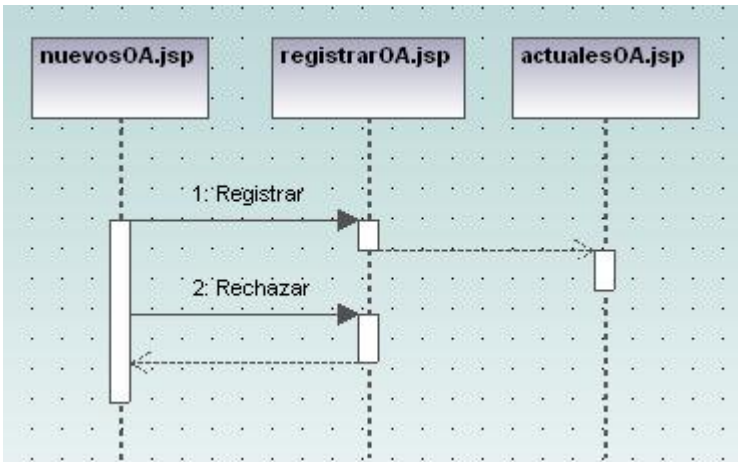
    b. El usuario pulsa *Aceptar* → se redirecciona a *registrarOA.jsp*. Se da de alta el OA mediante el método *altaOA()* de la clase *SentenciasSQL* y se redirecciona a *actualesOA.jsp* mostrando una ventana informativa

- Rechazar OA nuevo

4. El usuario pulsa *Rechazar* en la pantalla de nuevos OA → la función *rechazarOA()* del fichero *scriptNuevosOA.js* pide confirmación mediante una ventana informativa:

    a. El usuario pulsa *Cancelar* → se vuelve a la pantalla anterior (Paso 4)

    b. El usuario pulsa *Aceptar* → se redirecciona a *registrarOA.jsp*. Se rechaza el OA, mediante el método *bajaOA()* de la clase *SentenciasSQL;* se elimina el OA físico de la carpeta */archivos/*, mediante el método *borrarEspacioOA()* de la clase *Userspace* y se vuelve a *nuevosOA.jsp* mostrando una ventana informativa.



o Ver OA actuales

1. El usuario pulsa en *OA actuales* en el marco izquierdo → se muestra en el marco derecho la pantalla *actualesOA.jsp* con la lista de OA actuales

del sistema, para lo cual se utiliza el método *dameOA()* de la clase *SentenciasSQL*

Dentro de este caso de uso se pueden distinguir otros subcasos:

- Descargar OA actual

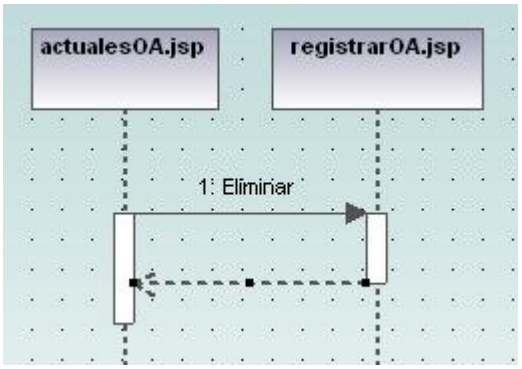
2. El usuario pulsa *Descargar* en la pantalla de OA actuales → el navegador muestra una pantalla de descarga.

- Eliminar OA actual

3. El usuario pulsa *Eliminar* en la pantalla de OA actuales → la función *eliminarOA()* del fichero *scriptNuevosOA.js* pide confirmación mediante una ventana informativa:

   a. El usuario pulsa *Cancelar* → se vuelve a la pantalla anterior (Paso 3)

   b. El usuario pulsa *Aceptar* → se redirecciona a *registrarOA.jsp*. Se elimina el OA, mediante el método *bajaOA()* de la clase *SentenciasSQL;* se elimina el OA físico de la carpeta */archivos/*, mediante el método *borrarEspacioOA()* de la clase *Userspace* y se vuelve a *actualesOA.jsp* mostrando una ventana informativa.



o Ver nuevos usuarios

1. El usuario pulsa en *Nuevos usuarios* en el marco izquierdo → se muestra en el marco derecho la pantalla *adminNuevos.jsp* con la lista de nuevos usuarios registrados, para lo cual se utiliza el método *dameUsuarios()* de la clase *SentenciasSQL*

Dentro de este caso de uso se pueden distinguir otros subcasos:

- Dar de alta usuario nuevo

2. El usuario pulsa *Dar de alta* en la pantalla de nuevos usuarios → la función *aceptar()* del fichero *scriptNuevosUsuarios.js* pide confirmación mediante una ventana informativa:

    a. El usuario pulsa *Cancelar* → se vuelve a la pantalla anterior (Paso 2)

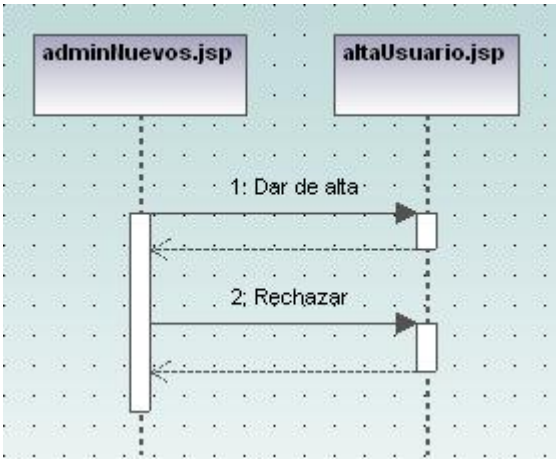
    b. El usuario pulsa *Aceptar* → se redirecciona a *altaUsuario.jsp*. Se da de alta al usuario mediante el método *altaUsuario()* de la clase *SentenciasSQL;* se crea el espacio de usuario mediante el método *crearEspacioUsuario()* de la clase *Userspace*; y se vuelve a *adminNuevos.jsp* mostrando una ventana informativa

- Rechazar usuario nuevo

3. El usuario pulsa *Rechazar* en la pantalla de nuevos usuarios → la función *rechazar()* del fichero *scriptNuevosUsuarios.js* pide confirmación mediante una ventana informativa:

    a. El usuario pulsa *Cancelar* → se vuelve a la pantalla anterior (Paso 3)

    b. El usuario pulsa *Aceptar* → se redirecciona a *altaUsuario.jsp*. Se rechaza el OA, mediante el método *bajaOA()* de la clase *SentenciasSQL;* se elimina el OA físico de la carpeta */archivos/*, mediante el método *bajaUsuario()* de la clase *SentenciasSQL* y se vuelve a *adminNuevos.jsp* mostrando una ventana informativa.



- Ver usuarios actuales

1. El usuario pulsa en *Usuarios actuales* en el marco izquierdo → se muestra en el marco derecho la pantalla *adminUsuarios.jsp* con la lista

de usuarios actuales del sistema, para lo cual se utiliza el método *dameUsuarios()* de la clase *SentenciasSQL.*

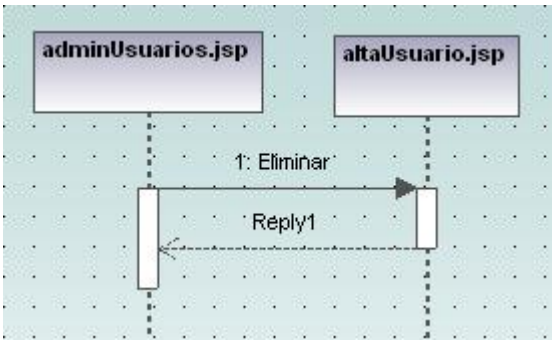
Dentro de este caso de uso existe otro subcaso:

- Eliminar usuario actual

2. El usuario pulsa *Eliminar* en la pantalla de usuarios actuales → la función *eliminar()* del fichero *scriptNuevosUsuarios.js* pide confirmación mediante una ventana informativa:

   a. El usuario pulsa *Cancelar* → se vuelve a la pantalla anterior (Paso 2)

   b. El usuario pulsa *Aceptar* → se redirecciona a *altaUsuario.jsp*. Se elimina el OA, mediante el método *bajaUsuario()* de la clase *SentenciasSQL;* se elimina el espacio de usuario, mediante el método *borrarEspacioUsuario()* de la clase *Userspace* y se vuelve a *adminUsuarios.jsp* mostrando una ventana informativa.



o Añadir Taxonomias

1. El usuario pulsa en *Añadir Taxonomia*, situado en el marco izquierdo → se muestra en el marco derecho la pantalla *anadirTaxonomia.jsp*.

2. El usuario introduce los datos solicitados y pulsa en *Enviar* → el script *anadirTaxonomia ()* del fichero *scriptNuevasTaxonomias.js* comprueba que se han introducido todos los datos.

   a) Si falta algún dato por rellenar, se muestra una pantalla de error (Paso 2)

   b) Si no, se envían los datos a *anadirTaxonomiaFin.jsp*, donde se realizan las siguientes acciones:
      - Se guarda el fichero enviado en el espacio que contiene todas las taxonomías que tenemos hasta el momento. El fichero seleccionado por el usuario se guardará si no existe ya un fichero con ese mismo nombre en el espacio

de taxonomías y si el nombre que se le da a este fichero no lo tiene asignada otra taxonomía, en caso que se produzca alguna de estas dos situaciones se informará al usuario.

Todos los ficheros presentes en el espacio de taxonomias son *.xml* que definen un método de organización para los objetos de aprendizaje. Como ya se ha comentado anteriormente estos ficheros definen taxonomías que esta sujetas a un estándar perteneciente a IMS [8] llamado IMS VDEX.

- Se guarda el contenido del archivo y se inserta en la base de datos con la fecha actual (método *insertarTaxonomia()* de la clase *SentenciasSQL*).

Por último, volvemos a *anadirTaxonomia.jsp*, mostrando un mensaje con información sobre cómo ha resultado el proceso.

o Ver taxonomías actuales

1. El usuario pulsa en *Taxonomías actuales* en el marco izquierdo → se muestra en el marco derecho la pantalla *taxonomiasActuales.jsp* con la lista de las taxonomías que tenemos actualmente en el sistema, para lo cual se utiliza el método *dameTaxonomias()* de la clase *SentenciasSQL*.

Dentro de este caso de uso existe otro subcaso:

- Eliminar taxonomia actual

2. El usuario pulsa *Eliminar* en la pantalla de taxonomías actuales → la función *eliminarTaxonomia()* del fichero *scriptNuevosTaxonomias.js* pide confirmación mediante una ventana informativa:

   a. El usuario pulsa *Cancelar* → se vuelve a la pantalla anterior (Paso 2)

   b. El usuario pulsa *Aceptar* → se redirige a *eliminarTaxonomia.jsp*. Se elimina la taxonomía, mediante el método *eliminarTaxonomia()* de la clase *SentenciasSQL;* y se vuelve a *taxonomiasActuales.jsp* mostrando una ventana informativa, que indica si la operación se ha realizado correctamente o si ha habido algún error durante el proceso de eliminación.

o Desconectar

1. El usuario pulsa *Desconectar* en el marco izquierdo → se redirecciona a *logout.jsp*, donde se eliminan los atributos de sesión, y finalmente a *login.html*, mostrándose éste en toda la pantalla

- Cambiar de idioma

    1. El usuario pulsa en uno de los iconos de idiomas situados en el marco izquierdo → se redirecciona a *cambioIdioma.jsp*, el cual se encarga de actualizar el perfil del usuario con el idioma seleccionado (método *idioma()* de la clase *SentenciasSQL*), volver a la pantalla en la que se encontrara el usuario antes de realizar esta acción, y refrescar dicha pantalla para que se refleje el cambio de idioma realizado.

6.4.- Proyecto GWT para "Buscar por taxonomía"

Para generar el árbol que representa a las taxonomías mediante el cual realizaremos la selección de los objetos que queremos buscar hemos utilizado una aplicación AJAX esta aplicación se ha realizado mediante el framework GWT que transforma código Java en una aplicación AJAX. La forma de crear un proyecto eclipse para GWT se puede ya se ha explicado anteriormente.

El proyecto esta formado por los siguientes paquetes:

- **com/ArbolXML/client/:** Archivos fuentes y subpaquetes del lado del cliente.
  - ArbolXML.java: Es la clase principal en ella se añade al panel el desplegable donde elegiremos la taxonomía a buscar, para ello se obtiene primero los archivos que hay en el espacio *taxonomías* mediante un método llamado remotamente, este método se ejecuta en el servidor, que nos devuelve un *String* con todas las taxonomías que hay en el sistema. Cuando el usuario selecciona una de las taxonomías, se llama al método *generarArbol()* de la clase *GenerarArbolTaxonomia*, a este método, leyendo el contenido del archivo se le pasa como parámetro un *String* con todo el archivo.
  - GenerarArbolTaxonomia.java: Se genera el árbol de la taxonomía de la taxonomía que se le pasa por parámetro y se añade el árbol generado al panel.
  - SrvObtenTaxonomias.java: Interfaz síncrona del método *obtenTaxonomias()* que ofrece el servidor.
  - SrvObtenTaxonomiasAsync.java: Interfaz asíncrona del método *obtenTaxonomias()* que ofrece el servidor.
- **com/ArbolXML/server/**: Archivos fuentes y subpaquetes del lado del servidor.
  - SrvObtenTaxonomiasImpl.java: Implementación de la interfaz *SrvObtenTaxonomias* que implementa el método *obtenTaxonomias()* que se llama remotamente desde *ArbolXML* y que obtiene el nombre de las taxonomías que existen en el sistema.
- **com/ArbolXML/public/:** Recursos estáticos que se pueden servir públicamente.
  - ArbolXML.html: Página donde se mostrará el desplegable y el árbol de la taxonomía.

6.5.- Proyecto GWT para “AbrirObjeto”

Para generar los árboles que representa a las organizaciones de los objetos mediante los cuales exploraremos el objeto hemos utilizado una aplicación AJAX esta aplicación se ha realizado mediante el framework GWT que transforma código Java en una aplicación AJAX. La forma de crear un proyecto eclipse para GWT se visto anteriormente en el apartado Herramientas Software.

El proyecto esta formado por los siguientes paquetes:

- **com/ArbolXML/client/:** Archivos fuentes y subpaquetes del lado del cliente.

  - AbrirObjeto.java: Es la clase principal en ella se llama a un método remotamente llamado *abrirOA()*, este método se ejecuta en el servidor y descomprime el objeto que se encuentra en un *.zip* y devuelve la ruta donde se ha descomprimido el OA. Cuando ya tenemos la ruta donde esta el OA, se llama al método *generarArbol()* de la clase *GenerarArbol*, si entre los archivos del objeto existe el archivo *imsmanifest.xml* que es el archivo que contiene la descripción del OA y en el podremos encontrar la organizaciones que puede adoptar el OA.

  - GenerarArbol.java: En esta clase se añade al panel un desplegable con las organizaciones que puede adoptar el OA. Cuando el usuario selecciona una de estas organizaciones se genera y se añade al panel el árbol correspondiente.

  - SrvAbrirOA.java: Interfaz síncrona del método *abrirOA()* que ofrece el servidor.

  - SrvAbrirOAAsync.java: Interfaz asíncrona del método *abrirOA()* que ofrece el servidor.

- **com/ArbolXML/server/**: Archivos fuentes y subpaquetes del lado del servidor.

  - SrvAbrirOAImpl.java: Implementación de la interfaz *SrvAbrirOA* que implementa el método *abrirOA()* que se llama remotamente desde *AbrirObjete.* Se descomprime el archivo *.zip* ,que se le pasa con atributo de sesión (ya que esta clase es un servlet), en una carpeta temporal llamada *temp/”nombre usuario que habre el objeto”* y se devuelve la ruta donde se ha descomprimido el *.zip* existen en el sistema.

  - Zipeado.java: Clase con la se descomprime el archivo *.zip* que contiene los archivos con componen el objeto de aprendizaje.

- **com/ArbolXML/public/**: Recursos estáticos que se pueden servir públicamente.

    - AbrirObjeto.html: Página donde se mostrará el desplegable con la organización posible del objeto y el árbol de la organización que hemos seleccionado.

6.6.- Tratamiento de errores

Como se ha explicado a lo largo del presente documento, esta aplicación realiza frecuentemente conexiones a la base de datos, consultas y modificaciones sobre diversas tablas y tratamiento de ficheros (*.zip*, *.pdf* y *.xml*), entre otras acciones.

Todas estas operaciones pueden fallar en un momento dado debido a mantenimientos de la base de datos, archivos defectuosos, falta de espacio en el servidor y otras muchas razones. Por tanto, la aplicación debe ser lo suficientemente robusta como para intentar prever dichos errores y explicar en dicho caso al usuario lo que ha sucedido.

Con ese objetivo, realizamos numerosos tratamientos de errores en las distintas secciones de la aplicación, entre los que destacan principalmente:

- Conexión a la base de datos

  En todas aquellas pantallas en las que realizamos una conexión contra la base de datos (login, envío de OA, consulta de usuarios…), capturamos el posible error y, en caso de producirse, redirigimos a la pantalla *error.jsp* mostrando la causa del mismo (base de datos en mantenimiento, drivers DB2 no encontrados…)

- Sentencias SQL contra DB2

  También capturamos errores que pueden producirse en la consulta o modificación de las tablas de la base de datos, debidos a causas poco probables (pero posibles) como la caída del gestor DB2 o la existencia de tablas corruptas.

  En caso de producirse dichos errores, la manera de informar al usuario varía dependiendo de la acción que esté realizando. Básicamente, mostramos el error de dos formas diferentes, de forma que siempre sea lo más claro y simple posible para el usuario:

  - Ventana Javascript:

    Esta opción se usa principalmente en el área de administración y en la gestión de espacio de usuario. Es más simple a la hora de programar y también más directo y visible para el usuario.

    Resulta adecuado su utilización, ya que en estos casos se tiene que mostrar un mensaje incluso cuando la acción se ha realizado correctamente. Además, se trata de acciones que en teoría no deberían realizarse masivamente, pues en caso contrario resultaría molesto el uso de estas ventanas, al tener que pulsar contínuamente en *Aceptar*.

    Por ejemplo, cuando el administrador confirma el registro de un nuevo usuario, es aconsejable informarle de que el proceso se

ha llevado a cabo correctamente. De la misma, si no ha sido así se le informa igualmente

- Mensaje sobre la misma pantalla

  Esta opción se usa en otras áreas en las que no se muestra ningún mensaje en caso de funcionar correctamente, o en aquellas en las que resulta más cómodo hacerlo de esta forma.

  Ejemplos de esta alternativa son principalmente: envío de OA, búsquedas y en general cualquier pantalla en la que se necesita acceder a la base de datos antes de que el usuario ordene alguna acción.

  Por ejemplo, en la pantalla de nuevos OA se realiza una consulta de dichos OA sobre la base de datos, antes de que el administrador decida realizar cualquier acción sobre ellos. En este caso, si esa consulta falla por cualquier motivo, se mostrará un mensaje sobre la propia pantalla informando de lo acontecido.

- Generación de archivos PDF

  En la exportación a PDF de la vista previa o de los comentarios de un OA, pueden producirse errores de ficheros durante la creación de los mismos, por lo que es conveniente tenerlo controlado. Esto se hace mediante un mensaje sobre la propia pantalla, de la misma forma que se ha detallado anteriormente para las sentencias SQL.

- Tratamiento de ficheros

  Hay varias pantallas en las que se trabaja con ficheros, y por tanto, pueden producirse errores en la lectura o escritura de los mismos. Este es el caso de acciones tales como: guardar OA en espacio de usuario, envío de OA…

  Nuevamente informamos mediante mensajes sobre la propia pantalla a la hora de tratar este tipo de errores en la aplicación.

6.7.- Seguridad en la aplicación

Al igual que se realiza un control de errores, la aplicación también debe evitar ataques contra la seguridad de la misma. Los dos aspectos a tener en cuenta en este ámbito son:

- Acceso a la página principal o cualquier pantalla de la aplicación sin logearse previamente.

  Si no se controlara esta acción, cualquier persona podría escribir la dirección de la página principal en el navegador y realizar cualquier operación sin ni siquiera ser un usuario registrado en el sistema.

  La solución que se ha implementado es crear atributos de sesión con el nombre y clave cuando el usuario se logea, y consultar siempre esos atributos en todas las pantallas de la aplicación. De esta forma, si alguien intenta acceder a otra pantalla sin pasar previamente por el login, el sistema lo detectará y mostrará una pantalla de error.

- Acceso a pantallas de administración por parte de usuarios ordinarios.

  Esto es aún más importante si cabe que el caso anterior, puesto que si la aplicación no evitara esta situación cualquier usuario podría realizar acciones que sólo corresponden al super-administrador, como borrar usuarios, borrar OA… lo cual podría resultar desastroso para el funcionamiento del sistema.

  La solución implementada en este caso es crear otro atributo de sesión en el Login que indique si el usuario es super-administrador o no. La pantalla principal muestra las opciones del área de administración sólo si se trata del super-administrador; y lo que es más importante, todas las pantallas de dicho área comprueban que existe ese atributo y que su valor es el adecuado, mostrando una pantalla de error en caso contrario

Cabe destacar que la duración de los atributos de sesión viene determinada por el servidor web. En Apache Tomcat, se configura mediante la etiqueta *session-config* del archivo *Tomcat/conf/web.xml*.

Si la sesión caduca, cualquier intento del usuario por acceder a las diferentes pantallas de la aplicación resultará en error hasta que se logee de nuevo.

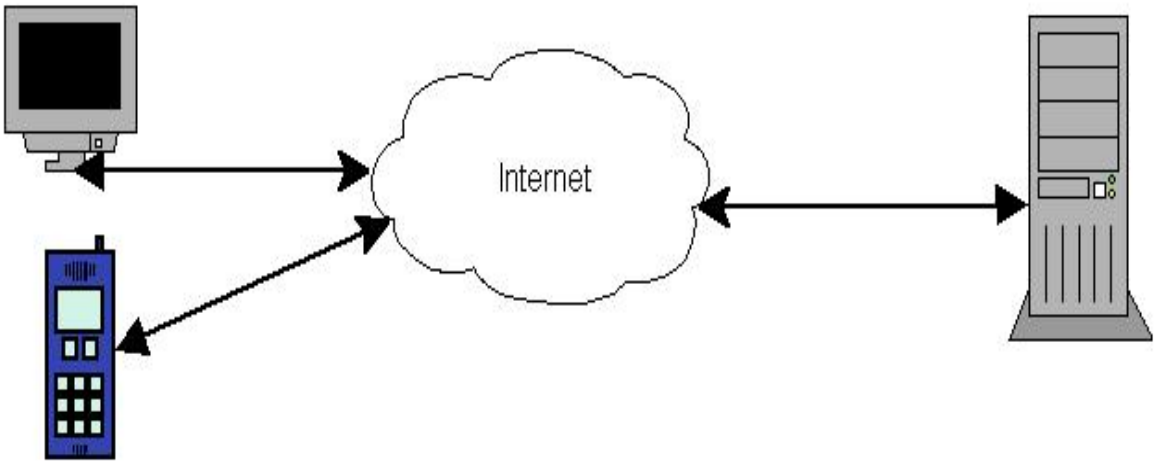
Por último, queda recalcar que si el usuario visualiza otras páginas en la misma ventana donde estaba utilizando la aplicación, sin haber realizado Desconexión, los atributos de sesión siguen activos, por lo que cualquiera podría entrar aprovechándose del Login realizado previamente.

6.8.- Servicios Web

6.8.1.- Introducción a los Servicios Web:

Los sistemas distribuidos y el acceso a servicios remotos, como es el caso de nuestro proyecto, han centrado gran parte de los esfuerzos tecnológicos de los últimos años, habiéndose creado distintas tecnologías y estándares distintos como por ejemplo: Java RMI, CORBA, COM/DCOM, las clásicas RPCs o los sistemas MOM basados en el paso de mensajes.

Con el crecimiento de la WWW (World Wide Web) surgió la idea de realizar estas tareas en sistemas distribuidos usando un protocolo estandarizado como HTTP:



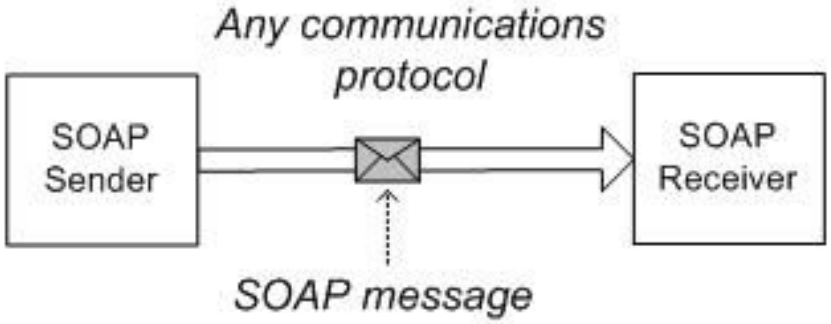
Como el propio consorcio responsable de la definición de estándares para el Web, World Wide Web Consortium, W3C, especifica en su documento Web Services Glossary, los servicios Web (Web Services) son sistemas software pensados para dar soporte a interacciones entre máquinas. Para que estas interacciones sean posibles, todo Servicio Web posee una interfaz descrita en un formato procesable por una máquina (utilizando el lenguaje WSDL, Web Service Description Language). En esta descripción se indica cómo otros sistemas pueden interactuar con el Servicio Web: qué mensajes se intercambian, qué parámetros deben pasarse al servicio para que lleve a cabo su función, qué resultados se obtienen o qué fallos pueden producirse.

Normalmente la invocación remota a un Servicio Web se lleva a cabo utilizando mensajes SOAP (Simple Object Access Protocol, a veces también referido como Service Oriented Access Protocol), serializados utilizando XML, Extensible Markup Language y transmitidos mediante un protocolo HTTP, HyperText Transfer Protocol. Para facilitar la localización de servicios, es posible registrar los mismos en un directorio o registro UDDI, Universal Description, Discovery and Integration.

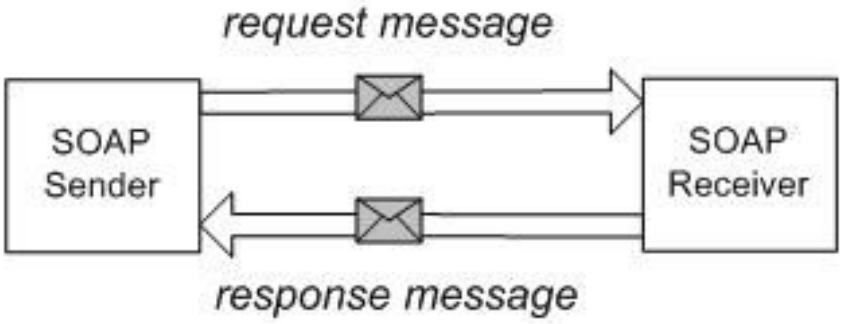
La descripción del servicio almacenada en el registro UDDI contiene información como el nombre del servicio, detalles sobre el proveedor del servicio, el tipo de servicio ofrecido, su ubicación URL (Uniform Resource Locutor) y detalles técnicos sobre el mismo (el WSDL del servicio). La información de esta descripción es utilizada por las aplicaciones cliente para decidir qué servicio pueden invocar.

Los Web Services son una tecnología middleware adecuada cuando se pretenden desarrollar arquitecturas orientadas a servicio, SOA (Service Oriented Architecture), que mediante SOAP, define el modo de transmitir mensajes XML de un punto a otro. Lo realiza proporcionando un protocolo de mensajes que es:

1.- Extensible (se basa en XML).
2.- Usable por una gran variedad de protocolos de transporte, tanto por TCP, HTTP, SMTP o incluso MQs (colas de mensajes).
3.- Independiente de las plataformas o lenguajes de programación.



SOAP engloba cada mensaje dentro de un sobre (envelope):



Vamos a ver las principales ventajas e inconvenientes de SOAP, para así poder decidir en cada caso si es valido para nuestra aplicación o si por el contrario, nos decantamos por otros mecanismos como son RMI, CORBA o DCOM.

Las mayores ventajas de SOAP son las siguientes:

- Interoperabilidad: el uso de estándares abiertos favorece la comunicación de sistemas heterogéneos.
- Seguridad: el uso de HTTP asegura la comunicación a través de firewallls.
- Desacoplamiento: Clientes y servidores no se han de conocer entre si a priori para comunicarse. Un Servicio Web está siempre disponible en Internet y puede ser invocado por diferentes clientes en momentos diferentes. De este modo el servicio está siempre activo dentro del contenedor Web. Además HTTP no mantiene una conexión permanente entre cliente y servidor por lo que se produce la comunicación sólo cuando es necesaria.

Por otro lado, SOAP tiene como mayores inconvenientes:

- Eficiencia: el uso de XML es más ineficiente que la codificación binaria pues requiere el paseado de XML.
- Mantenimiento de sesiones: como HTTP no mantiene la conexión se requieren mecanismo de mantenimiento de sesión cuando el servicio requiere de ello. Sin embargo, en la mayoría de los Servicios Web encontrados son sin estados (stateless) y no requieren mantenimiento de sesiones.
- Interoperabilidad: SOAP todavía está madurando y se pueden encontrar conflictos entre diversas implementaciones. También es difícil la comunicación entre estructuras de datos complejas entre implementaciones diferentes.

Por ello, los servicios Web SOAP no son la solución a todos los problemas. Por ejemplo, en ámbitos Intranet donde la eficiencia es clave, sería recomendable utilizar serialización binaria. En cambio, si la comunicación se produce entre sistemas heterogéneos en puntos remotos de Internet, entonces SOAP es una solución adecuada.

Antes de meternos de lleno en la creación e invocación de un Servicio Web, vamos a detenernos un momento en el “contrato” elegido por SOAP para definir el nombre y parámetros de entrada y de salida: el WSDL.

WSDL (Web Service Description Language) es un lenguaje XML que actúa como un IDL (Interface Definition Language), definiendo el nombre de los parámetros y los tipos de parámetros que aceptan. Además, existe una herramienta, el WSDL2JAVA, que genera stubs a partir de dicho contrato.

Para hacernos una idea de ello, clarificar un poco todos estos conceptos, vamos a mostrar un ejemplo sencillo de una clase Java y su correspondiente WSDL:

```
public class Calculator {
      public int add(int i1, int i2) {
            return i1 + i2;
      }
      public int subtract(int i1, int i2) {
```

```
            return i1 - i2;
      }
}
```

```
<wsdl:definitions targetNamespace="http://localhost:8080/axis/Calculator.jws">
<!--
WSDL created by Apache Axis version: 1.2RC3
Built on Feb 28, 2005 (10:15:14 EST)
-->
<wsdl:message name="subtractRequest">
<wsdl:part name="i1" type="xsd:int"/>
<wsdl:part name="i2" type="xsd:int"/>
</wsdl:message>

<wsdl:message name="subtractResponse">
<wsdl:part name="subtractReturn" type="xsd:int"/>
</wsdl:message>

<wsdl:message name="addResponse">
<wsdl:part name="addReturn" type="xsd:int"/>
</wsdl:message>

<wsdl:message name="addRequest">
<wsdl:part name="i1" type="xsd:int"/>
<wsdl:part name="i2" type="xsd:int"/>
</wsdl:message>
<wsdl:portType name="Calculator">
<wsdl:operation name="add" parameterOrder="i1 i2">
<wsdl:input message="impl:addRequest" name="addRequest"/>
<wsdl:output message="impl:addResponse" name="addResponse"/>
</wsdl:operation>

<wsdl:operation name="subtract" parameterOrder="i1 i2">
<wsdl:input message="impl:subtractRequest" name="subtractRequest"/>
<wsdl:output message="impl:subtractResponse" name="subtractResponse"/>
</wsdl:operation>
</wsdl:portType>

<wsdl:binding name="CalculatorSoapBinding" type="impl:Calculator">
<wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>

<wsdl:operation name="add">
<wsdlsoap:operation soapAction=""/>

<wsdl:input name="addRequest">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://DefaultNamespace" use="encoded"/>
</wsdl:input>

<wsdl:output name="addResponse">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://localhost:8080/axis/Calculator.jws" use="encoded"/>
</wsdl:output>
</wsdl:operation>

<wsdl:operation name="subtract">
<wsdlsoap:operation soapAction=""/>

<wsdl:input name="subtractRequest">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://DefaultNamespace" use="encoded"/>
</wsdl:input>

<wsdl:output name="subtractResponse">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://localhost:8080/axis/Calculator.jws" use="encoded"/>
</wsdl:output>
</wsdl:operation>
</wsdl:binding>

<wsdl:service name="CalculatorService">

<wsdl:port binding="impl:CalculatorSoapBinding" name="Calculator">
<wsdlsoap:address location="http://localhost:8080/axis/Calculator.jws"/>
</wsdl:port>
```

```
</wsdl:service>
</wsdl:definitions>
```

Aunque no entraremos a describir todo el contenido de WSDL, podemos observar a grandes rasgos que en dicho fichero encontramos una definición de las operaciones ofertadas por el servicio (portType, operation), de los mensajes de solicitud y respuesta intercambiados (message) incluyendo tipos de parámetros (type) y de los protocolos de comunicación (blindings).

6.8.2.- Creación de un servicio Web: Servicio de búsqueda

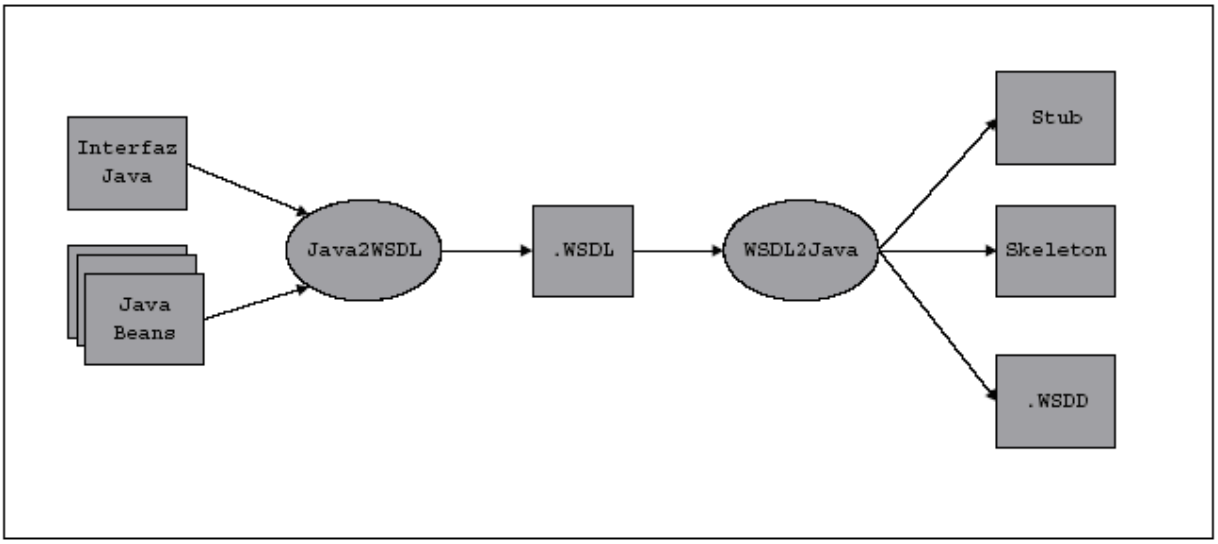
Dado que vamos a usar la herramienta AXIS para Tomcat, los pasos básicos para conseguir publicar nuestro servicio son los siguientes, esquematizado en la figura 1:

1. Transformar todas las clases que implemente nuestro Servicio Web en clases JAVA tipo Beans.
2. Construcción de un interfaz de los métodos que deba implementar nuestro servicio Web y compilación junto con la clase que use dichos métodos.
3. Utilizar Java2WSDL para generar el archivo WSDL del interfaz anterior.
4. Utilizar WSDL2Java para generar los esqueletos del código necesario para desplegar el servicio y los stubs para invocarlo.
5. Añadir al esqueleto lo necesario para completar el servicio Web.
6. Creación de una librería que contenga todas las clases del servicio

Como aclaración, diremos que para que una clase sea tipo Beans, tiene que cumplir, como mínimo, las siguientes características:

- Debe tener un constructor sin argumentos
- Debe tener accesores y mutadores para cada uno de los atributos que lo forman.

Ahora, mostraremos un pequeño esquema sobre los pasos anteriormente expuestos, para reflejar mejor el proceso:



A continuación detallaremos, con la mayor claridad posible, cada uno de los pasos del proceso anterior, dando detalles de los problemas que nos hemos ido encontrando en cada uno de ellos.

1.- Transformación de las clases en JAVA Beans:

La realización este paso es muy sencilla, dado que sólo consiste en crear un constructor sin argumentos y los accesotes y mutadores para cada uno de los atributos de la clase. Esto facilitará a AXIS la creación de los stubs, evitándonos problemas innecesarios.

Además, transformarlas en Beans nos facilitará más tarde, la serialización de nuestras clases, sobre todo las que se envían a través del SOAP.

2.- Construcción de la interfaz:

Una vez realizado el paso anterior correctamente, vamos a crear una interfaz teniendo en cuenta los métodos que compondrán nuestro servicio, así con los tipos y las clases que utilizará.

Seguidamente, mostraremos el código que hemos creado para nuestro servicio. Lo hemos llamado ServicioBusqueda y tendrá el siguiente código:

```
package buscador;

public interface ServicioBusqueda extends java.rmi.Remote {
     public ConjuntoResults[] busquedaSimple(AtrVal[] in0, int in1)
         throws java.rmi.RemoteException;
}
```

Una vez creada esta interfaz, podemos crear la clase que implemente dicho método. En nuestro caso, esta clase se llamará búsqueda y tendrá el siguiente código:

```
package buscador;


import java.sql.*;
import java.util.*;


public class búsqueda{

   private Auxiliar aux = new Auxiliar();
   private Db db = new Db();

   private String cadenaError = "";

   public int conectar() throws Exception{
```

```
      int conex = db.connect();
      if(conex != 1) cadenaError = db.getError();

      return conex;

   } // fin conectar

public void desconectar() throws Exception{

      db.disconnect();
      cadenaError = db.getError();

   }


public ConjuntoResults[] busquedaSimple(AtrVal[] palabras, int cod){

   ConjuntoResults[] resultsets;
   int numConect;

   try {
         numConect = conectar();
           if(numConect != 1){
            if(numConect == -2){
               resultsets = new ConjuntoResults[1];
               ConjuntoResults conjunto = new ConjuntoResults();
               conjunto.setAutor("fallo");
               conjunto.setCodigo("error -2");
               conjunto.setNombre("");
               conjunto.setRuta("");
               resultsets[0] = conjunto;
               return resultsets;
              }
             else if(numConect == -3){
               resultsets = new ConjuntoResults[1];
               ConjuntoResults conjunto = new ConjuntoResults();
               conjunto.setAutor("fallo");
               conjunto.setCodigo("error -3");
               conjunto.setNombre("");
               conjunto.setRuta("");
               resultsets[0] = conjunto;
               return resultsets;
              }
             else if(numConect == -1){
               resultsets = new ConjuntoResults[1];
               ConjuntoResults conjunto = new ConjuntoResults();
               conjunto.setAutor("fallo");
               conjunto.setCodigo("error -1");
               conjunto.setNombre("");
               conjunto.setRuta("");
               resultsets[0] = conjunto;
               return resultsets;
                }
           }
   } catch (Exception e) {
      // TODO Bloque catch generado automáticamente
      e.printStackTrace();
   }

   String whereExists = "";
```

```
int cont = 0;
if (palabras == null ){
   resultsets = new ConjuntoResults[1];
   ConjuntoResults conjunto3 = new ConjuntoResults();
   conjunto3.setAutor("null");
   conjunto3.setCodigo("");
   conjunto3.setNombre("");
   conjunto3.setRuta("");
   resultsets[0] = conjunto3;
   return resultsets;
}

while((palabras != null) && (cont < palabras.length) &&
   (palabras[cont] != null)){
      String campo = (palabras[cont]).getCampo();
      String[] valores = (palabras[cont]).getValor();
      if((campo !=  null) && (campo.equals("Todos")))
         campo = "manifest/metadata/lom";
      whereExists += " AND (";
      int cont2 = 1;
      whereExists += "XMLExists ('$o [contains (" + campo + ",
      \"" + valores[0] + "\")]' PASSING objeto as \"o\") ";
      while((valores != null) && (cont2 < valores.length)){
            whereExists += "OR XMLExists ('$o [contains (" +
               campo + ", \"" + valores[cont2] + "\")]' PASSING
               objeto as \"o\") ";
            cont2++;
         }

      whereExists += ")";

      cont++;

   }
String whereCod = " WHERE codigo>" + cod;
try{
   int numResultados = contadorBusqueda(palabras);
   if (numResultados == -1){
         resultsets = new ConjuntoResults[1];
         ConjuntoResults conjunto = new ConjuntoResults();
         conjunto.setAutor("fallo");
         conjunto.setCodigo("error consulta contador");
         conjunto.setNombre("");
         conjunto.setRuta("");
         resultsets[0] = conjunto;
         return resultsets;
   }
   String consultaSimple = "SELECT nombre,autor,codigo,ruta FROM
                  " + esquema + "." + tablaObjetos
                  + whereCod + whereExists
                  + " AND registrado=1"
                  + " ORDER BY codigo";

       ResultSet rsetConsulta = db.execSQL(consultaSimple);

       if(rsetConsulta == null){
         resultsets = new ConjuntoResults[1];
         ConjuntoResults conjunto = new ConjuntoResults();
         conjunto.setAutor("fallo");
         conjunto.setCodigo("error consulta");
         conjunto.setNombre("");
```

```
                conjunto.setRuta("");
                resultsets[0] = conjunto;
                return resultsets;
             }

             resultsets = new ConjuntoResults[numResultados+1];
             ConjuntoResults conjunto = new ConjuntoResults();
             conjunto.setAutor("funciona");
             conjunto.setCodigo("");
             conjunto.setNombre("");
             conjunto.setRuta("");
             resultsets[0] = conjunto;
             int i = 1;
             while((rsetConsulta.next()) && (i < resultsets.length)){
                   ConjuntoResults conjunto2 = new ConjuntoResults();
                   conjunto2.setAutor(rsetConsulta.getString(1));
                   conjunto2.setCodigo(rsetConsulta.getString(2));
                   conjunto2.setNombre(rsetConsulta.getString(3));
                   conjunto2.setRuta(rsetConsulta.getString(4));
                   resultsets[i] = conjunto2;
                   i++;
             } // fin while

              rsetConsulta.close();

            try {
             desconectar();
              } catch (Exception e) {
                // TODO Bloque catch generado automáticamente
                e.printStackTrace();
             }
             return resultsets;
          }

       catch(SQLException sqle){
          cadenaError = sqle.getMessage();
          return null;
          }

       finally{ cadenaError = aux.formatearError(cadenaError); }

   }
}
```

Como se puede observar, la función principal de la clase, y por tanto el servicio Web que vamos a implantar, es busquedaSimple, que utilizará las funciones conectar y desconectar para realizar la búsqueda sobre la base de datos.

También podemos ver los parámetros tanto de entrada como de salida. Los tipos propios AtrVal (entrada) y ConjuntoResults (salida) están creados como beans.

3.- Utilización de Java2WSDL:

Una vez compilados tanto la interfaz ServicioBusqueda como la clase búsqueda como todas las demás clases de las que depende la clase búsqueda, utilizamos la herramienta Java2WSDL para generar el archivo WSDL que describe nuestro Servicio Web en función de su interfaz.

La información que pasamos a Java2WSDL es la siguiente:

- Nombre del archivo WSDL a generar (busqueda.wsdl)
- URL que tendrá el servicio Web. Esto será lo único que habrá que cambiar para describir el servicio Web en diferentes máquinas. En caso de que queramos hacerlo local, el URL seria http://localhost:8080/axis/services/serviciobusqueda. En caso de que queramos hacerlo en otra máquina, sustituiremos "localhost" por la dirección estática de dicha máquina.
- Espacio de nombres de destino para el WSDL (urn:serviciobusqueda)
- Mapeo entre el paquete Java y su espacio de nombres correspondiente(serviciobusqueda = urn:serviciobusqueda)
- La clase completamente cualificada (buscador.ServicioBusqueda)

La línea de comandos para ejecutar la llamada es la siguiente:

```
java  org.apache.axis.wsdl.Java2WSDL -o busqueda.wsdl  -
l"http://localhost:8080/axis/services/serviciobusqueda" -n
urn:serviciobusqueda -p"serviciobusqueda" urn:serviciobusqueda
```

Una vez ejecutada dicho comando, obtenemos el archivo "busqueda.wsdl" que define nuestro servicio Web, y el cuál enlazará con el archivo de la descripción semántica del servicio.

4.- Utilización de WSDL2Java:

Llegados a este punto, lo que tenemos que hacer es utilizar el fichero busqueda.wsdl para generar mediante la herramienta WSDL2Java el esqueleto básico del servicio y todo lo necesario para desplegarlo o quitarlo.

Para evitar copias innecesarias de código, vamos a generar los stubs directamente en el mismo paquete (carpeta) donde tenemos los archivos anteriores. Por tanto, el paquete será buscador. La información dada a WSDL2Java es la siguiente:

- Directorio base de salida (.)
- Alcance del despliegue (aplicación, petición o sesión)
- Solicitud de la generación del código del lado del cliente (no habría sido así si sólo hubiésemos querido acceder a un servicio Web externo ya construido)
- Paquete donde colocar el código (buscador)
- WSDL a utilizar (busqueda.wsdl)

La línea de comandos para la llamada es la siguiente:

```
java  org.apache.axis.wsdl.WSDL2Java -o . -d Session -s -p buscador
busqueda.wsdl
```

Después de ejecutar dicha instrucción obtenemos los siguientes archivos en el directorio buscador:

- ServiciobusquedaSoapBindingImpl.java : código que implementa nuestro servicio Web, será el único archivo que tendremos que editar para que utilice la clase que implementa lo que queremos que haga el servicio.
- ServicioBusqueda.java: interfaz remota al sistema ServicioBusqueda (extends remote y métodos del ServicioBusqueda.java original, throws RemoteExceptions)
- ServicioBusquedaservice.java: Interfaz del servicio Web, ServiceLocator lo implementa
- ServicioBusquedaServiceLocator.java: "Helper Factory" para conseguir un manejador del servicio.
- ServicioBusquedaSoapBindingSkeleton.java: código del "skeleton" del lado del servidor.
- ServicioBusquedaSoapBindingStub.java: código del "stub" de lado del cliente que encapsula accesos a él.
- Deploy.wsdd: Descriptor de desplegado que pasaremos al sistema Axis para que el servicio Web sea accesible.
- Undeploy.wsdd: Descriptor de desplegado para eliminar el servicio del sistema Axis.

Además, si abrimos los archivos AtrVal.java y ConjuntoResults.java, veremos que han sido modificados con respecto al original. Esto es porque son tipos propios construidos que se envían a través del canal. Por tanto, tiene que ser serializables. Para conseguirlo, Axis añade funciones de serialización y deserialización para dichos tipos.

5.- Modificar el esqueleto

Como se ha comentado anteriormente, al llegar a este punto es necesario modificar el archivo ServicioBusquedaSoapBindingImpl.java para que utilice la clase busqueda.java y así realizar la tarea que queremos que haga nuestro servicio Web. Las modificaciones que vamos a hacer aparecerán en negrita:

```
package buscador;

public class ServiciobusquedaSoapBindingImpl implements
buscador.ServicioBusqueda{
   búsqueda bubsqueda = new búsqueda();
    public buscador.ConjuntoResults[] busquedaSimple(buscador.AtrVal[]
in0, int in1) throws java.rmi.RemoteException {
        return busqueda.busquedaSimple(in0,in1);
    }
```

6.- Creación de la librería:

Para poder utilizar el servicio Web, necesitamos crear una librería que contenga todo las clases y .wsdd que definan nuestro servicio. Así, mediante la siguiente llamada podremos crear el buscador.jar, previa compilación:

```
javac paquete\ws\*.java

jar cvf envio.jar paquete/*.class paquete/ws/*.class
```

6.8.3.- Despliegue de un servicio Web: servicio búsqueda

Tras la creación de nuestro Servicio Web, lo único que nos queda en desplegarlo en la máquina que va a ofrecer dicho servicio para que pueda ser llamada desde otras máquinas, o localmente en su defecto.

Como comentó en el apartado de "creación de un servicio Web", de lo que ahora mismo disponemos es de tres archivos:

- Deploy.wsdd: Descriptor de desplegado que pasaremos al sistema Axis para que el servicio Web sea accesible.
- Undeploy.wsdd: Descriptor de desplegado para eliminar el servicio del sistema Axis.
- Buscador.jar: Librería que contendrá las clases que implementan nuestro Servicio Web.

Para realizar el despliegue seguimos tres pasos básicos:

1.- Copiamos la librería buscador.jar a TOMCAT_HOME/Webapps/Axis/WEB_INF/lib

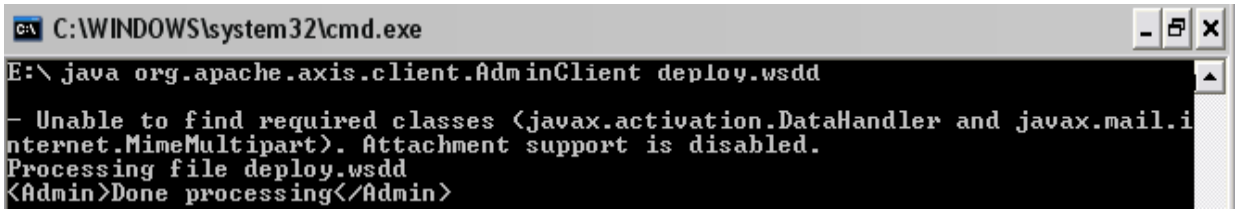
2.- Arranca el Tomcat, ya sea desde el monitor o desde configure Tomcat (Start Tomcat)

3.- Ejecutamos la siguiente instrucción, teniendo en cuanta que el deploy.wsdd se generó a partir de un .wsdl (busqueda.wsdl) que contiene la dirección de la máquina donde estamos publicando el servicio. La instrucción sería:

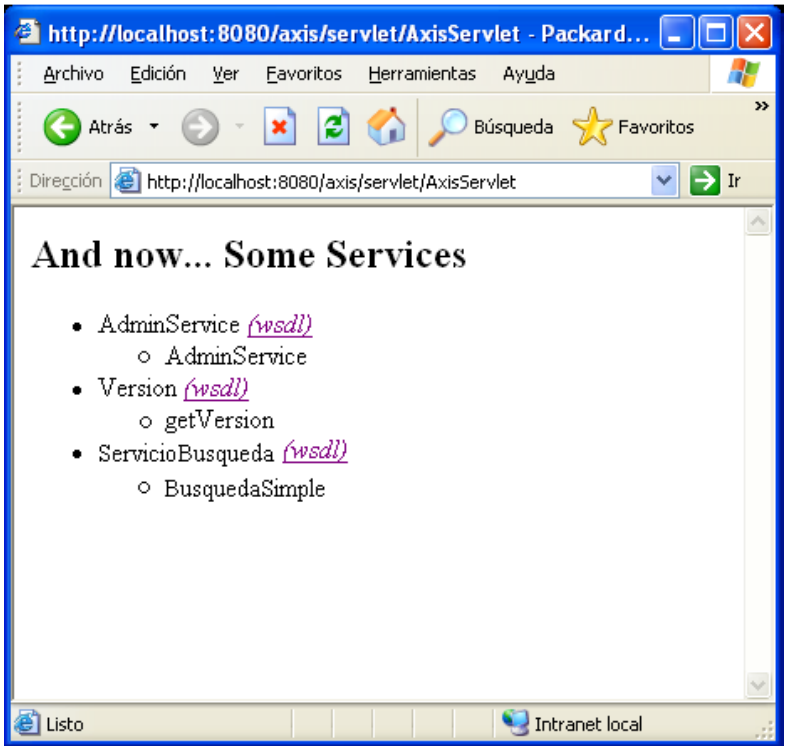
```
java org.apache.axis.client.AdminClient deploy.wsdd
```

Recordamos que para que esta instrucción funcione, deben estar añadidas al classpath todas las librerías de Axis.

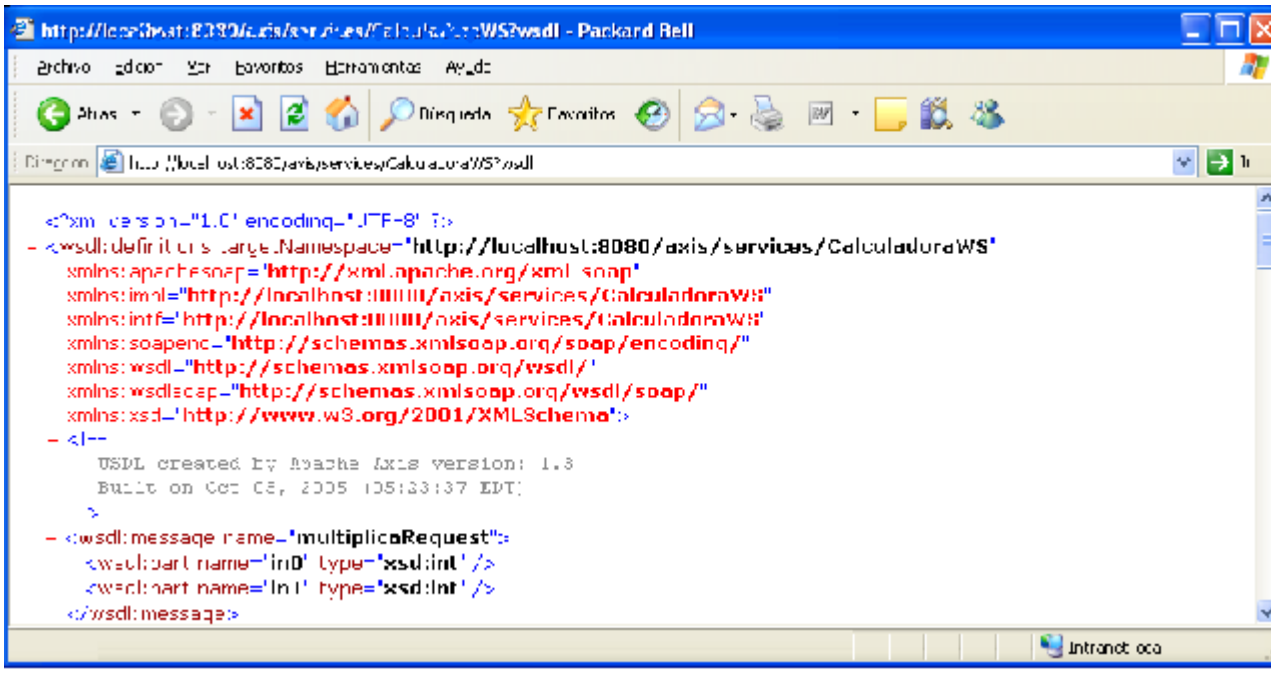
Si se ha publicado correctamente, aparecerá en siguiente mensaje



Ahora, para comprobar que nuestro servicio Web está activo, con Apache Tomcat arrancado, abrimos un explorador Web y escribimos en el URL la siguiente dirección: http://localhost:8080/axis/servlet/AxisServlet, además de ver todos los servicios activos en dicha máquina.



Si a continuación pulsamos en el enlace *(wsdl)*, podremos ver el código wsdl de nuestro servicio.



Si todos estos pasos han funcionado correctamente, desde este momento nuestro servicio Web empieza a estar disponible y ahora podemos crear nuestro cliente, para que llame al servicio.

No vamos a extendernos en este apartado, dado que la programación de un cliente es muy sencilla. En nuestro caso, tendríamos que recoger los datos proporcionados por el usuario, concatenarnos e introducirlos en un array y pasárselo como parámetro al Servicio Web. Y aquí es donde nos queríamos detener un momento, y mostrar las instrucciones necesarias para llamar a nuestro Servicio Web. Las instrucciones son las siguientes:

```
ServicioBusqueda busq = new
ServicioBusquedaServiceLocator().getserviciobusqueda();
ConjuntoResults[] result = busq.busquedaSimple(prueba, -1);
```

Así, mediante estas sencillas instrucciones, hacemos la llamada al Servicio Web, mandando a través del SOAP la llamada y los parámetros y evolviéndonos el resultado de la búsqueda.

7.- CONCLUSIONES Y TRABAJO FUTURO

Este proyecto nos ha permitido experimentar diversas e interesantes tecnologías que se usan en la actualidad, como DB2, Ajax, Web Services, JSP…Cabe destacar el uso del lenguaje XQuery y la versión 9 de DB2, de reciente publicación y sobre los que existe actualmente poca documentación.

Como trabajo futuro proponemos la inserción de esquemas de validación (.xsd) en DB2 para comprobar la corrección de los objetos de aprendizaje que se envíen en base a un estándar determinado (mediante el lenguaje XQuery). En la actualidad el objeto de aprendizaje no se valida debido que no se disponía de un esquema que pudiera ser insertado correctamente en DB2; no obstante, la interfaz gráfica muestra la posibilidad de elegir un esquema de validación, si existiera, a la hora de enviar un objeto de aprendizaje

También se puede profundizar en el aspecto de los permisos y usuarios. Queda, pues, abierta la posibilidad de otorgar diferentes permisos según el tipo de usuario (por ejemplo,profesor o alumno), que actualmente sólo se distingue entre usuario ordinario y administrador.

8.- APENDICE

8.1.- Apéndice A: pruebabd.jsp

Código del fichero *pruebabd.jsp*:

```
<%@ page import= "java.sql.*" %>
<%@ page import= "java.util.*" %>
<%@ page import= "java.io.*" %>
<%@ page import= "java.lang.*" %>


<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>LOGIN JSP</TITLE>
</HEAD>

<%!
    /************ PARAMETROS A CONFIGURAR ************/

    String userId = "riesgo"; // usuario con el que nos conectamos a la BD
    String password = "nov1siao"; // clave con la que nos conectamos a la BD
    String alias = "riesgo"; // BD a la que nos conectamos

    String colorError = "#55FF55"; // color de la pantalla en caso de error

    /************ FIN PARAMETROS A CONFIGURAR ************/
%>
<%!
    /************ CLASE DB  ************/

String cadenaError = ""; // mensaje de error de las excepciones

// Clase para manejar la base de datos

class Db{

   public Connection con = null;  // Conexion

   public Db(){}  // Constructor

   public int connect() throws Exception{

       String url = null;

       // In Partitioned Database environment, set this to the node number
       // to which you wish to connect (leave as "0" in non-Partitioned Database environment)
       String nodeNumber = "0";

       Properties props = new Properties();

       url = "jdbc:db2://10.10.0.26:50000/" + alias;

       try{

          Class.forName("com.ibm.db2.jcc.DB2Driver").newInstance();
```

```
            props.setProperty("CONNECTNODE", nodeNumber);

            con = DriverManager.getConnection( url, userId, password );

            // enable transactions
            con.setAutoCommit(false);

            return 1;
          }

        catch(ClassNotFoundException cle){
            cadenaError = cle.getMessage();
            return -3;
        }

        catch(SQLException sqle){
            sqle.printStackTrace();
            cadenaError = sqle.getMessage();
            return -2;
        }

        catch(Exception ne){
            ne.printStackTrace();
            cadenaError = ne.getMessage();
            return -1;
          }

    } // fin connect

public void disconnect() throws Exception{

    try{
        // makes all changes made since the previous commit/rollback permanent
        // and releases any database locks currrently held by the Connection.
        con.commit();

        // immediately disconnects from database and releases JDBC resources
        con.close();
      }

    catch(SQLException sqle){
        sqle.printStackTrace();
        cadenaError = sqle.getMessage();
      }

    catch(Exception ne){
        ne.printStackTrace();
        cadenaError = ne.getMessage();
      }

    } // fin disconnect

public ResultSet execSQL(String sql) throws SQLException{

      try{
          Statement s = con.createStatement();
          ResultSet r = s.executeQuery(sql);
          return (r == null) ? null : r;

        }
```

```
        catch(SQLException sqle){
             sqle.printStackTrace();
             cadenaError = sqle.getMessage();
             return null;
          }

        catch (Throwable e){
              e.printStackTrace();
              cadenaError = e.getMessage();
              return null;
          }

      } // fin execSQL

     public int updateSQL(String sql) throws SQLException{

          try{
              Statement s = con.createStatement();
              int r = s.executeUpdate(sql);
              return r;
           }

          catch(SQLException sqle){
              sqle.printStackTrace();
              cadenaError = sqle.getMessage();
              return 0;
           }

          catch (Throwable e){
               e.printStackTrace();
               cadenaError = e.getMessage();
               return 0;
            }

      } // fin updateSQL

    public ResultSet procSQL(String sql) throws SQLException{
        try{
             CallableStatement cs = con.prepareCall(sql);
             cs.execute();
             return cs.getResultSet();

          }
        catch(SQLException sqle){
             sqle.printStackTrace();
             cadenaError = sqle.getMessage();
             return null;
          }
        catch (Throwable e){
              e.printStackTrace();
              cadenaError = e.getMessage();
              return null;
          }
      } // fin procSQL
  } // fin Db

/************    FIN CLASE DB   ************/

%>
```

```
<%
 /************    CONEXION   ************/
  Db db = null ;

  db = new Db();

  int numConect = db.connect();

  if(numConect != 1){
  %>

       <BODY BGCOLOR='<%= colorError %>'>

           <BR> <BR>

        <CENTER>

   <%

       if(numConect == -2){

   %>
            <H2> ERROR EN CONEXION: Revise usuario, clave y BD <H2>
   <%
         }

       else if(numConect == -3){

   %>
           <H2> DRIVER DB2 NO ENCONTRADO <H2>
   <%

         }

       else if(numConect == -1){

   %>
           <H2> ERROR EN CONEXION: Vuelva a intentarlo <H2>
   <%

         }

   %>
       </CENTER>

       <br> <br> <br>

       <H4> <%= cadenaError %> </H4>

       </BODY>

    <%
      return;
    }
  /************ FIN CONEXION    ************/
 %>
<BODY BGCOLOR="lightgreen">
 Conectado
</BODY>
</HTML>
```

8.1.- Apéndice B: Lenguaje de búsqueda

Cada día se impone más la rapidez para realizar acciones en el mundo informático. Una de estas mejoras, que está implantada ya en los buscadores con más renombre, es el "lenguaje de búsqueda".

Dicho lenguaje nos proporciona, a cambio de recordar una serie de nemotécnicos o caracteres, una búsqueda más rápida de lo que queremos, sin tener que estar accediendo a la "búsqueda avanzada" e ir rellenando campo por campo las diferentes etiquetas del LOM por las que queremos buscar. Así, simplemente desde la página de "búsqueda simple", podemos acortar la búsqueda mucho más y con una ganancia de tiempo mayor.

Para ello, hemos creado una serie de caracteres que identifican unívocamente a cada una de las etiquetas por las que está permitido buscar en nuestro buscador. Simplemente, se tendrá que poner una "#" seguido de la letra que identifique a campo y entre comillas, lo que estamos buscando. Para poner varios valores, se separan entre comas.

Un ejemplo de esto seria #t"Don quijote de la Mancha", #k"Cervantes", que acotaría la búsqueda a titulo: Don Quijote de la Mancha y palabra clave : Cervantes.

A continuación hacemos un listado de las letras que identifican a cada campo:

- '**t**' - 'lom/general/title': El nombre asignado a este objeto educativo.

- '**l**' - 'lom/general/language': El idioma o idiomas humanos predominantes en este objeto educativo para la comunicación con el usuario.

- '**d**' - 'lom/general/descripcion': Una descripción textual del contenido de este objeto educativo.

- '**k**' - 'lom/general/keyword': Una palabra clave o frase que describe el tema principal del objeto educativo.

- '**s**' - 'lom/lifeCycle/status': El estado de completitud o condición de este objeto educativo.

- '**r**' - 'lom/lifeCycle/contribute/role': Tipo de contribución.

- '**e**' - 'lom/lifeCycle/contribute/entity': La identificación e información de las entidades (personas u organizaciones) que han contribuido a este objeto educativo. Las entidades deben ser ordenadas de forma que aparezcan en primer lugar las más relevantes.

- '**a**' - 'lom/lifeCycle/contribute/date': La fecha de la contribución

- ‘**f**’ - ‘lom/technical/format’: El(los) tipo(s) de datos de (todos los componentes) este objeto educativo.
- ‘**z**’ - ‘lom/technical/size’: El tamaño del objeto educativo digital expresado en octetos. El tamaño se representa como un valor decimal (base 10). Por lo tanto , solo deben ser utilizados los dígitos del '0' al '9'. La unidad es el octeto, no MB, ni GB, etc.

- ‘**y**’ - ‘lom/technical/requirement/orComposite/type’: La tecnología requerida para usar este objeto educativo, por ejemplo, hardware, software, red, etc.

- ‘**n**’ - ‘lom/technical/requirement/orComposite/name’: El nombre de la tecnología requerida para utilizar este objeto educativo.

- ‘**c**’ - ‘lom/educational/interactivityType’: El tipo de aprendizaje predominante soportado por este objeto educativo.

- ‘**g**’ - ‘lom/educational/learningResourceType’: El tipo específico de recurso educativo. El tipo predominante debe aparecer en primer lugar.

- ‘**v**’ - ‘lom/educational/interactivityLevel’: El grado de interactividad que caracteriza a este objeto educativo. La interactividad en este contexto se refiere al grado en el que el aprendiz puede influir en el aspecto o comportamiento del objeto educativo.

- ‘**m**’ - ‘lom/educational/SemanticDensity’: El grado de concisión de un objeto educativo. La densidad semántica de un objeto educativo puede ser estimada en función de su tamaño, ámbito o – en el caso de recursos auto-regulados tales como audio y vídeo – duración.

- ‘**u**’ - ‘lom/educational/intendedEndUsedRole’: El usuario(s) principal(es) para el que ha sido diseñado este objeto educativo. El predominante debe aparecer al principio.

- ‘**x**’ - ‘lom/educational/context’: El entorno principal en el que se utilizará este objeto educativo.

- ‘**p**’ - ‘lom/educational/typicalAgeRange’: Edad del destinatario típico.

- ‘**i**’ - ‘lom/educational/difficulty’: Este elemento describe lo difícil que resulta, para los destinatarios típicos, trabajar con y utilizar este objeto educativo.

- ‘**o**’ - ‘lom/educational/description’: Comentarios sobre cómo debe utilizarse este objeto educativo.

- ‘**w**’ - ‘lom/educational/languaje’: El idioma utilizado por el destinatario típico de este objeto educativo.

Nota: toda la información referente a los campos del LOM han sido sacados directamente de la traducción del estándar. Para más información, puede descargarse dicho estándar desde :

http://www.cenorm.be/cenorm/businessdomains/businessdomains/isss/activity/lomspanish1.doc

9.- BIBLIOGRAFIA

[1]IMS Global Learning Consortium: http://www.imsglobal.org

[2]ADL SCORM: http://www.adlnet.org/scorm/index.cfm

[3]IEEE LTSC: http://ieeeltsc.org/

[4]IBM http://www.ibm.com

[5]Apache Tomcat: http://tomcat.apache.org/

[6]Apache Axis: http://ws.apache.org/axis/

[7]Apache FileUpload: http://jakarta.apache.org/commons/fileupload

[8]Google Web Toolkit: http://code.google.com/webtoolkit/

[9]Servicios Web:http://www.programacion.net/java/tutorial/servic_web/

[10] Servicios Web:
http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=axis

[11] Servicios Web: http://www.it.uc3m.es/berto/SI/WS/index.html

[12] Servicios Web:
http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=soap