Sistemas Informáticos
Curso 2003/2004

USO Y GESTIÓN DE OBJETOS DE APRENDIZAJE EN LA WEB

Patricia Díaz Ayuso.
Beatriz Díaz García.
Concepción Sanz Pineda.

Dirigido por:
Prof. Alfredo Fernández-Valmayor Crespo.
Dpto. Sistemas Informáticos y Programación.

Facultad de Informática
Universidad Complutense de Madrid

ÍNDICE

AGRADECIMIENTOS.

Nuestro más sincero agradecimiento a nuestro director de proyecto Alfredo Fernández-Valmayor Crespo y a su colaborador Antonio Sarasa Cabezuelo por el tiempo que nos han dedicado y por su apoyo en todos estos meses de trabajo.

RESUMEN.

El proyecto " Uso y gestión de objetos de aprendizaje en un entorno Web" se ha desarrollado en la asignatura Sistemas Informáticos. Tiene como objetivo el desarrollo de la versión 0.0 del Museo Virtual García Santesmases de la Facultad de Informática de la Universidad Complutense de Madrid.

Las características más importantes que se encuentran implementadas en la aplicación se centran en la necesidad de utilizar todo el material del que dispone el museo con fines educativos. Esto implica que todos los "objetos" del museo estén organizados y relacionados para facilitar la visita e investigación de alumnos, profesores e investigadores. Otras de las características más importantes que se ha tratado es proporcionar mecanismos al museo para que los usuarios accedan a la información sin necesidad de estar en el museo, y la transportabilidad de la información que compone el museo entre sistemas de características similares.

ABSTRACT.

The proyect called "Use and Management of Learning Objects in the Web" has been developed into the subject "Sistemas Informáticos". The main target is the development of the fisrt version of the "Virtual García Santesmases Museum" located at the School of Computer Science of the Complutense University of Madrid.

The most important features implemented in the aplication are related to use all the available data of the Museum for educational purposes.

This means that the "objects" must be organized and bound to provide the visit and research of the student, professors, and researchers.

In addition, the museum supplies the users with procedures to move data between systems that understand the same standards used to organize the data.

1.INTRODUCCIÓN

Este documento proporciona una visión general del proyecto "Uso y Gestión de Objetos de Aprendizaje en la Web" que se ha desarrollado en la asignatura de Sistemas Informáticos durante el curso 2003/2004.

Este proyecto se puede enmarcar dentro de los proyectos e-learning, pero a diferencia de otros proyectos educativos éste no se centra en la construcción de herramientas informáticas para la creación de unidades didácticas. Este proyecto tiene como objetivo construir un museo virtual, una aplicación informática que permita crear objetos de aprendizaje tomando como base los objetos de un museo. Un requisito importante de esta aplicación es la transportabilidad e interoperabilidad de los objetos de aprendizaje creados. Así, estos objetos podrán ser reutilizados dentro de otros entornos y sistemas educativos.

El resultado inmediato de este proyecto ha sido la construcción de la versión 0.0 del Museo de Informática García Santesmases (MIGS). Esta aplicación permite que el administrador del MIGS pueda construir objetos de aprendizaje basados en los objetos reales del museo. Posteriormente el usuario, alumno o visitante, podrá visitar a través de un entorno Web los objetos del museo. En principio estos objetos se corresponden con los objetos que se encuentran en los pasillos de la tercera planta de la Facultad de Informática de la Universidad Complutense de Madrid. El museo virtual permite acceder además a ciertos objetos que por diversas razones no están expuestos, tales como documentos y fotografías. Las posibilidades del museo virtual permiten que a partir de los objetos primarios del museo, expuestos y no expuestos, se puedan construir otros objetos que combinan objetos anteriores. Estos objetos compuestos facilitan que el alumno o visitante pueda establecer relaciones significativas entre los objetos del museo y estudiar y comprender el contexto en que estos objetos fueron creados. Finalmente la aplicación también permite exportar los objetos de aprendizaje creados y sus recursos asociados a otros entornos de enseñanza, para poder pasar a formar parte, por ejemplo de un curso e-learning.

Los temas tratados en esta memoria están organizados de la siguiente forma

1. Introducción
2. Conceptos y tecnologías de referencia:
   En este apartado se introducen los conceptos más importantes sobre los que se ha desarrollado la aplicación, como son:
   - Sistemas de enseñanza a distancia e-learning
   - Recomendaciones y estándares (para sistemas e-learning)
   - Objetos de aprendizaje (learning objects)
   - Tecnologías Web (PHP, JavaScript, XML…).

3. Antecedentes del MIGS
   Se describen otras aplicaciones que han sido referencia inmediata del Museo de Informática. Se estudian sus principales características, ventajas y desventajas.

4. Requisitos y especificaciones del MIGS
   Funciones principales desarrolladas para los distintos usuarios de la aplicación.
   - Punto de Vista del Administrador:
     - Crear Objeto

- Modificar Objeto
- Exportar/importar Objeto

- Punto de Vista del Usuario:
  - Buscar
  - Navegar

5. Aspectos más importantes del desarrollo

En este apartado se explican las tecnologías utilizadas para el desarrollo del museo.

- XML (DTDs y XML Schemata utilizados).
- Arquitectura y organización de la aplicación
- Diseño de la base de datos.

6. Estado actual de la aplicación y trabajo futuro
   Se centra en especificar el funcionamiento de la aplicación, que puntos se han cubierto, así como aquellos que se han quedado por cubrir, y que podrían ser tratados como futuros trabajos, y posibles mejoras a realizar, diferenciando en esta parte principalmente los siguientes apartados:

   - Trabajo por hacer, posibles mejoras.

7. Bibliografía

8. Apéndice: Manual de Usuario que explica el funcionamiento detallado de la aplicación.

9. Índice de Palabras: contiene una lista de palabras claves empleadas a lo largo de la memoria.

2. CONCEPTOS Y TECNOLOGÍAS DE REFERENCIA

El desarrollo de este proyecto se ha basado fundamentalmente en el concepto de "objeto de aprendizaje" (learning objects) que ha sido promovido por iniciativas como: IEEE/LTSC (Learning Technology Standards Comittee)[1] IMS [2], ADL (Advanced Distributed Learning) e IEEE/LTSC (Learning Technology Standards Comittee):SCORM (Sharable Content Object Reference Model) [3], promovida por ADL, es una especificación para la construcción de objetos de aprendizaje complejos, a partir de otros más sencillos.

Como se ha mencionado anteriormente uno de los conceptos más importantes, sobre el que se asienta este proyecto es el concepto de Objeto Virtual que podríamos decir que es un tipo específico de objeto de aprendizaje (Learning Object). Un objeto virtual puede ser definido como *"un objeto digital que sirve para agrupar toda la información relacionada con un determinado objeto real"*[4]. Cada uno de estos objetos puede ser descrito mediante un modelo de metadatos basado en la propuesta de LOM (Learning Object Metadata) [5], dicho modelo permite disponer de una forma estándar de clasificación de la información y establecer las posibles relaciones existentes entre los diferentes objetos virtuales. Según L.Mortimer[1] [6]algunos de los aspectos que caracterizan a un objeto de aprendizaje son:

1) *Contenido*. El contenido y las actividades deben soportar un aprendizaje objetivo, y su evaluación debe de estar orientada a conseguir dicha objetividad.

2) *Tamaño*. Realizar las actividades implicadas por un objeto de aprendizaje no debe llevar mucho tiempo.

3) *Contexto y capacidades*. Un objeto de aprendizaje debe poder existir de forma independiente y debe poder ser utilizado por cualquier participante (estudiante) que tenga determinadas capacidades y en el momento que lo necesite.

4) *Etiquetado y almacenado*. El contenido se describe mediante un conjunto de etiquetas normalizadas (meta-datos).

5) *Construcción incremental*. Los objetos de aprendizaje complejos (con una estructura bien definida) se construyen utilizando otros objetos de aprendizaje previamente construidos partiendo de objetos de aprendizaje más básicos (hasta llegar a los objetos atómicos).

6) *Interdependencia*. Un objeto de aprendizaje se considera que está formado por tres componentes interdependientes: el objeto de aprendizaje en sí mismo, los meta-datos (la forma estandarizada de describir su contenido) y un componente de gestión del aprendizaje (LMS o *Learning Management System*) que almacena, localiza y entrega contenidos.

Además de las características de los objetos de aprendizaje, ya mencionadas, también tienen gran importancia en este proyecto los estándares educativos, que a continuación se introducen.

La iniciativa de ADL, y en particular la de SCORM, que ha sido la principal referencia para este proyecto, además de esta otras iniciativas que también nos han servido de referencia han sido: ARIADNE, IEEE/LTSC e IMS. En SCORM [3] de define un "modelo de agregación de contenido" (content agregation model), que debe de ser pedagógicamente neutro. El modelo de agregación de contenido propuesto por SCORM está formado básicamente por tres elementos: el "modelo de contenido", los "metadatos", y el "modelo de empaquetamiento". El modelo de contenido se compone a su vez de tres elementos: assets (los contenidos más básicos), SCOs (Sharable Content Object) conjunto de

[1] (Learning) Objects of Desire: Promise & Practicality". Learning Circuits

assets que pueden ejecutarse en el entorno de enseñanza del sistema y finalmente las "estructuras de agregación" (content agregation) que son las estructuras capaces de organizar diferentes recursos de aprendizaje hasta formar una unidad didáctica coherente.
La propuesta de LOM se ha consolidado como la principal referencia para describir mediante metadatos los recursos educativos digitales y hacerlos disponibles a través de la Web. Así, el modelo de metadatos de LOM, implica que la información referente a un objeto virtual se agrupe en categorías. El esquema básico está formado por nueve categorías:

1) General: Engloba las características independientes del contexto además de descriptores del recurso.
2) Ciclo de vida: Características referentes al ciclo de vida del recurso.
3) Meta-metainformación: Aspectos de la propia descripción.
4) Técnica: Aspectos técnicos del recurso.
5) Uso educativo: Características educativas o pedagógicas de recurso.
6) Derechos: Condiciones de uso del recurso.
7) Relación: Relaciones del recurso con otro recurso.
8) Observaciones: Permite comentarios sobre el uso del recurso.
9) Clasificación: Características del recurso según lo describen diferentes catálogos.

Por otro lado, existen diferentes grupos como Dublín Core [7] y ARIADNE [8] que han ajustado sus esquemas de metadatos a LOM, pero con menos nivel de detalle.

La aproximación que utilizamos para desarrollar este proyecto tiene como base los estándares antes descritos y el concepto de objeto virtual. El dominio de este proyecto son los objetos que se encuentran en el museo de informática García Santesmases de la Facultad de Informática de la Universidad Complutense de Madrid. Es decir, lo que buscamos es crear objetos virtuales a partir de los objetos disponibles en dicho museo, abstrayendo las características fundamentales, así como las posibles relaciones que se puedan establecer entre ellos. Cada objeto virtual se describe mediante un modelo de meta-datos basado en el estándar de LOM(Learning Object Metadata). Para llevar a cabo este proceso de virtualización se ha añadido dentro del museo virtual un acceso a una herramienta que permite crear objetos virtuales según los estándares antes definidos, y que también permite modificar objetos virtuales que hayan sido creados con anterioridad, y que se encuentran almacenados en el servidor en una base de datos relacional, en la cual la entidad fundamental es el objeto virtual.

El MIGS tiene como requisito importante que la información puede ser accedida desde cualquier computador que tenga acceso a la red y disponga de un navegador web. Para diseñar este entorno Web se han usado principalmente las siguientes tecnologías Web: PHP, HTML, JavaScript para crear un entorno Web dinámico, XML y DTDs para construir los manifiestos de los objetos virtuales.

3. ANTECEDENTES DEL MIGS

El proyecto del museo virtual de Informática partió inicialmente de otro museo virtual, el museo de Arqueología de la Facultad de Geografía e Historia. Este museo es utilizado como una herramienta de trabajo por investigadores y alumnos como apoyo en la docencia de varias asignaturas y doctorados, lo que implica que la información que almacena su base de datos crezca rápidamente. Esto dificulta la gestión de los objetos que lo conforman así como el mantenimiento de la propia aplicación. Entre los objetos de este museo hay una gran cantidad de documentos y recursos de aprendizaje, sin embargo no se aprovecha completamente toda esta información porque no puede exportarse a otros sistemas.

El museo virtual de informática es un proyecto que ha intentado solucionar muchos de los problemas que tiene el museo virtual de Arqueología. Se intenta facilitar la gestión y mantenimiento mejorando el diseño de su base de datos para que toda la información y el material educativo sea consistente. Otra de las mejoras ha sido permitir que el contenido del museo sea accesible y exportable a otros entornos.

4.REQUISITOS Y ESPECIFICACIONES DEL MIGS

El sistema que quiere implementarse debe satisfacer los siguientes requisitos funcionales:

Se distingue entre dos tipos de usuarios: visitante e investigador.

1.Usuario Visitante

El usuario visitante debe poder realizar las siguientes operaciones:

- ***Navegar por todos los objetos del museo.***
  El usuario puede acceder a toda la información accediendo a todas las fichas que conforman un objeto.

- ***Buscar objetos en el museo.***
  El visitante puede realizar búsquedas sobre el museo. Para ello puede especificar cualquier número de características que debe tener el objeto que desea encontrar en el museo.

2. Usuario investigador

La funcionalidad del usuario investigador está poco desarrollada ya que en trabajos futuros debe diferenciarse entre investigadores y administrador del sistema. Actualmente el investigador tiene todos los privilegios sobre el museo, las operaciones que puede realizar son:

- ***Crear nuevo objeto.***
  El museo tiene la opción **"Crear un nuevo objeto"** para que un objeto pueda ser añadido al museo se debe rellenar todos los datos y metadatos del objeto así como los recursos relacionados. Si se añade como recurso del objeto otro objeto del museo se comprueba que las dependencias son válidas. Una vez completada la información del objeto se empaqueta siguiendo los estándares de IMS y se crea una carpeta comprimida que está lista para subirse al museo.

- ***Subir objeto.***
  Una vez creado un objeto se sube al museo y se desempaqueta comprobando si es válido para ser un objeto del museo, esto implica que en la carpeta que lo conforma aparezca el documento manifiesto, los recursos a los que hace referencia y el documento objetoVirtual.xml, que contiene todos los datos del objeto.

- ***Bajar objeto.***
  Cualquier objeto del museo puede ser recuperado y exportado a otro sistema. Un objeto se baja empaquetado según el estándar de IMS y con todos los objetos del museo de los que depende también empaquetados según IMS. Al descargarse un objeto se baja el manifiesto, los recursos y

el documento objetoVirtual.xml, que tiene asociado una hoja de estilos XSLT que permite la visualización del objeto, mostrando sus recursos y atributos.

- ***Modificar objeto*.**
  El usuario investigador tiene la capacidad de modificar de una manera segura cualquier dato del museo. Se pueden añadir nuevos recursos o eliminar dependencias de otros objetos.

5. ASPECTOS MÁS IMPORTANTES DEL DESARROLLO

5.1 Diseño de la base de datos

Nuestra aplicación requiere guardar en una base de datos relacional un documento XML (imsmanifest.xml). En este documento se tiene en formato estándar toda la información que caracteriza a un objeto de aprendizaje. La información contenida en el documento manifiesto está compuesta por : metadatos, estructura del objeto y recursos.

Se deben diseñar las tablas de forma que pueda ser fácilmente reconstruido el documento XML a partir de los datos de las tablas. Como la información de las tablas puede variar (cambiar un campo, añadir un registro, etc) no es posible almacenar el documento XML como un documento de texto tal cual, porque con simples consultas a la base de datos no podríamos modificarle.

En la actualidad existen nuevos sistemas XML construidos para manejar las demandas de datos XML en forma nativa, sin embargo la mayoría de las aplicaciones tradicionales de negocio y en concreto nuestra aplicación dependen de una base de datos relacional. Esta es la razón por la que los principales gestores de bases de datos relacionales tienen en la actualidad estrategias de almacenamiento XML, procesos de conversión, y niveles de soporte para XML.

Estrategias existentes[9]:

1. Almacenar el documento completo en forma de texto, como un gran objeto binario (BLOB).
   Esta estrategia no es válida porque se desea poder modificar directamente mediante consultas a la base de datos, de una forma rápida y sencilla la información que constituye el documento XML.
2. Almacenar una forma modificada del documento completo en el sistema de archivos.
   Este método no es válido para documentos XML grandes y complejos, además tiene varias limitaciones de escalabilidad, flexibilidad en almacenamiento y recuperación, y seguridad dado que los datos descansan fuera de la base de datos.
3. Mapear la estructura de los datos en el documento en la base de datos.
   Esta es la estrategia que se va a seguir por ser la más flexible, ya que se puede acceder a los datos del documento mediante consultas de la base de datos.

Se ha analizado el soporte XML en Oracle9i y en SQL Server 2000 Microsoft y ninguna de sus características se han ajustado a nuestro problema. Esto se debe a que ambos gestores crean una tabla por elemento del documento XML lo que haría que nuestra base de datos fuera intratable.

5.1.2 Diseño elegido para la base de datos

Finalmente se ha optado por diseñar unas tablas específicas para nuestro problema: almacenar la información de un documento XML en el menor número posible de tablas y con la posibilidad de reconstruir el documento XML a partir de las tablas sin perder ningún tipo de información.

La información que se necesita guardar en la base de datos es: Metadatos del objeto, recursos del objeto y dependencias del objeto.

- **Metadatos del objeto**

La información referida a los metadatos se encuentra en elemento <lom> que es un contenedor de otros elementos que constituyen la metainformación.

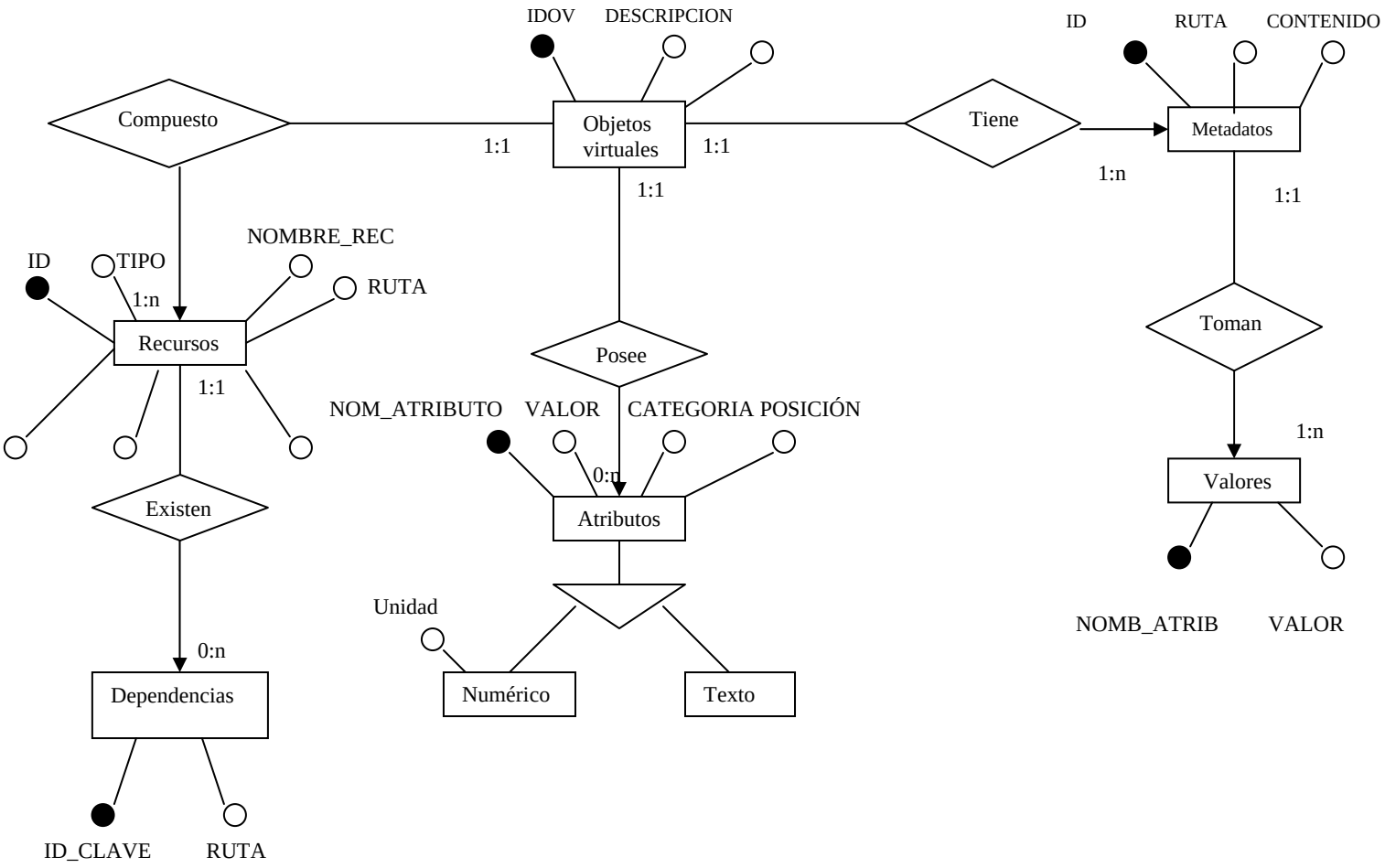
En las tablas debe guardarse la siguiente información:

- Ruta de etiquetas hasta el elemento
- Atributos y Contenido del elemento.
- Número de secuencia que permita diferenciar entre elementos que tengan la misma ruta de etiquetas.
- Objeto Virtual al que pertenece la información del documento XML, ya que cada manifiesto contiene toda la información de un objeto.

- **Recursos y dependencias de los objetos**

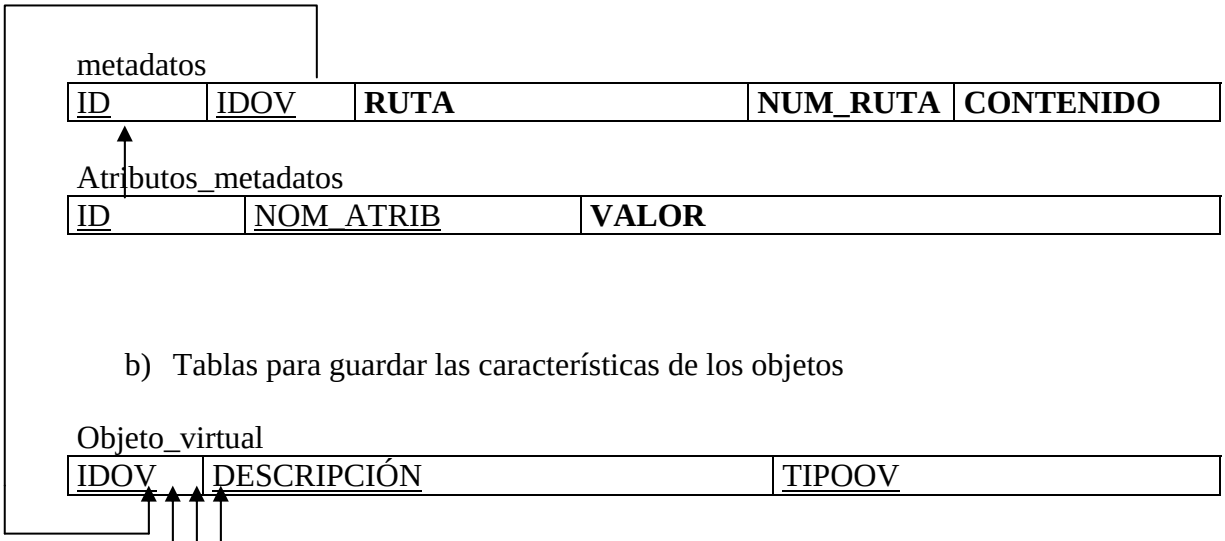
Esta información aparece en el manifiesto. Por un lado se obtiene los recursos del objeto y por otro lado se tiene un recurso especial que es otro documento xml en el que se tiene información de atributos característicos del objeto.

Modelo Entidad/Relacion

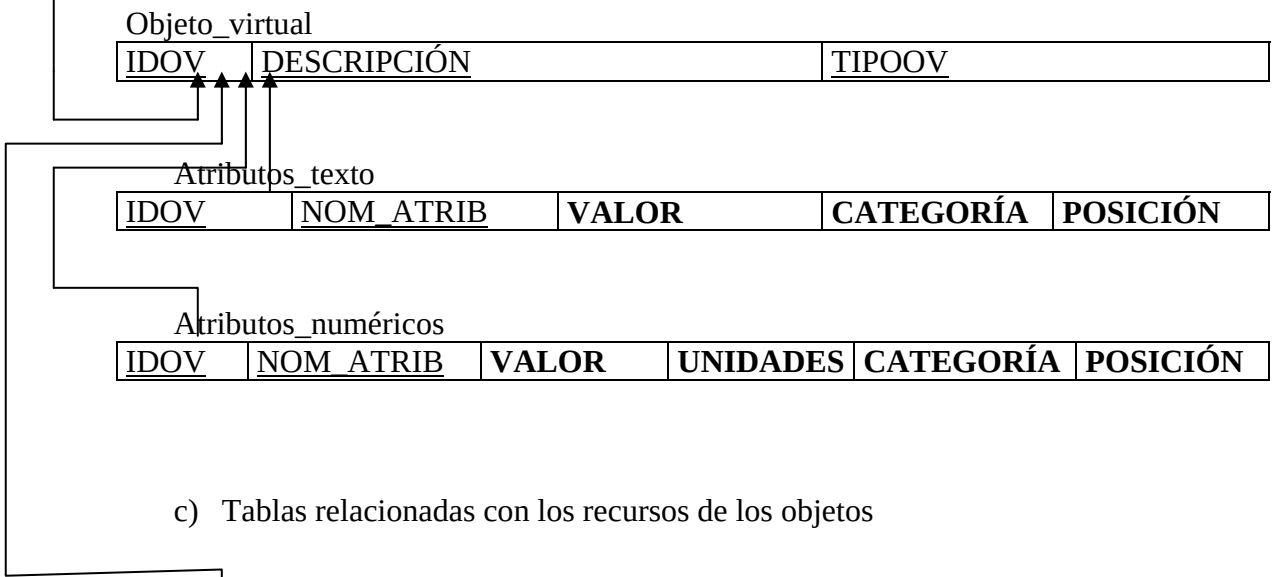


Modelo Relacional

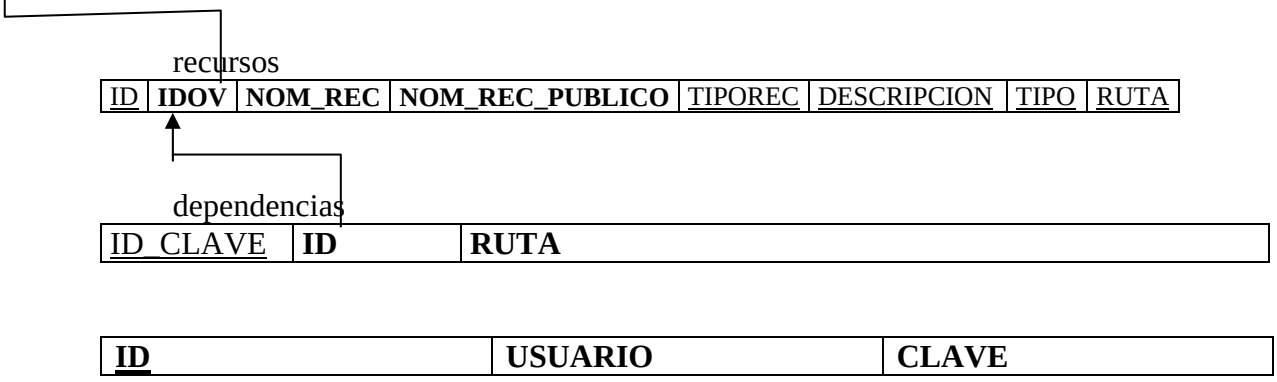
a) Tablas para guardar la información de los metadatos



b) Tablas para guardar las características de los objetos



c) Tablas relacionadas con los recursos de los objetos



ID	**USUARIO**	**CLAVE**

Descripción de las tablas:

1) metadatos:

   - ID: clave primaria.
   - IDOV: identificador de Objeto virtual, es una clave externa que hace referencia al campo IDOV de la tabla objeto_virtual.
   - Ruta: secuencia de etiquetas XML hasta el elemento del que se guarda el contenido.
   - Num_Ruta: cadena de números y puntos que indica el orden de los elementos en el documento XML. (Ejem 1.1.1).
   - Contenido: contenido del elemento indicado.

2) Atributos_metadatos:

   - ID: clave externa que hace referencia al campo ID de la tabla metadatos.
   - Nombre_atrib: atributo del elemento de la tabla metadatos.
   - Valor: valor del atributo.

3) Objeto_virtual:

   - IDOV: identificador de Objeto virtual.
   - Descripción: atributo descripción del OV, es de tipo texto por lo que se guarda en una tabla diferente de la del resto de atributos del OV.
   - TipoOv: indica el tipo de objeto (piezas, documentos o fotos).

4) Atributos_texto:

   - IDOV: clave externa que hace referencia al campo IDOV de la tabla objeto_virtual.
   - Nombre_atrib: atributo de texto del Objeto.
   - Valor: valor del atributo.
   - Categoría: clasificación de los tipos de atributos.
   - Posición: número de la ficha en la que se presenta el atributo.

5) Atributos_numéricos:
   - IDOV: clave externa que hace referencia al campo IDOV de la tabla objeto_virtual.
   - Nombre_atrib: atributo numérico del Objeto.
   - Valor: valor del atributo.
   - Unidades: unidades del atributo (MB/s, Hz, m, seg, etc).
   - Categoría: clasificación de los tipos de atributos.
   - Posición: número de la ficha en la que se presenta el atributo.

6) Recursos:

   - ID: identificador del recurso que sirve como clave de la tabla
   - IDOV: identificador del Objeto virtual.

- Nom_rec: nombre del recurso.
- Nom_rec_publico: nombre del recurso que se muestra a un usuario visitante.
- Tipo: tipo del recurso (gif, jpg, html, ov, etc, etc).
- Descripción: contiene una breve descripción del recurso.
- TipoRec: Relación entre el recurso y el objeto virtual.
- Ruta: Indica a quien pertenece el recurso, por ejemplo si el valor de este campo es " / ", entonces quiere decir que el recurso pertenece al OV de la fila. Si el contenido fuera " 1180/ ", el recurso pertenecería al OV cuyo IDOV es 1180. En caso de que el recurso tuviera tipo OV indicaría que el objeto virtual de la fila depende de todo ese objeto y no sólo de uno de sus recursos.

7) Dependencias:

- ID_CLAVE: clave primaria.
- ID: hace referencia al campo ID de recursos.
- Ruta: Indica de quien depende ese recurso.

Esta tabla recoge las dependencias de recursos como por ejemplo páginas HTML que no son OV pero que también dependen de otros "objetos" en la BD. En la tabla recursos se reflejaría el propietario del documento HTML y en la tabla dependencias se reflejaría los recursos u objetos a los que referencia ese documento. Esto implica que para eliminar un recurso o un OV es necesario mirar las dependencias en las dos tablas. En nuestro ejemplo se observa que el recurso con ID=3 (perteneciente al objeto virtual 1125) depende de estos dos recursos (por ejemplo puede tener enlaces a ellos o sencillamente contenerlos).

8) Usuarios:

- ID: clave_primario
- Usuario: nombre del usuario.
- Clave: contraseña del usuario

Esta tabla contiene a todos los usuarios con permiso total sobre la aplicación. Es decir con capacidad para crear objetos, subirlos, bajarlos, modificarlos o eliminarlos.

Ejemplo:

metadatos

ID	IDOV	**RUTA**	**CONTENIDO**
1	**1125**	metadata/lom/general/title/string	Apple II plus

atributos_metadatos

ID	NOM_ATRIB	**VALOR**
1	**language**	en

objeto_virtual

IDOV	DESCRIPCIÓN	TIPO
1125	**Microcomputador de 8 bits**	**Pieza**

atributos_texto

IDOV	NOM_ATRIB	**VALOR**	**CATEGORÍA**	**POSICIÓN**
1125	**logica**	TTL	Datos del HW	4

atributos_numéricos

IDOV	NOM_ATRIB	**VALOR**	**UNIDADES**	**CATEGORÍA**	**POSICIÓN**
1125	**Frecuencia de reloj**		Hz	Datos del HW	4

recursos

ID	**IDOV**	**NOM_REC**	**TIPO**	RUTA	NOM_REC_PUBLICO	TIPOREC	DESCRIPCION
1	1125	Foto1.jpg	Jpg	**/**	**Foto principal**	**Ordenador**	**Esta es la foto de la vista principal del objeto...**
2	1125	Foto2.gif	gif	**1183/**	**Foto del monitor**	**Periférico**	**Este es el monitor...**
3	1125	Pagina1.html	html	**/**	**Artículo 1**	**Documento relacionado**	**Este documento es una artículo...**
4	1125	Objeto Virtual	OV	**1190/**	**Procesador**	**Componente**	**Procesador ...**

dependencias

ID_CLAVE	**ID**	**RUTA**
1	3	1120/Pagina2.html
2	3	/Foto1.jpg

usuarios

ID	USUARIO	CLAVE
1	Jacinto	Eureka21

5.2 Descripción del documento ObjetoVirtual.xml

Cuando un objeto es creado (rellenando las fichas que se deseen con los datos del objeto) para ser subido posteriormente al museo, se crea un documento xml que guarda información necesaria para que el objeto se suba y se guarde correctamente en la base de datos del museo. La DTD de este documento es la siguiente:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!ELEMENT Objeto_Virtual (tipoOV, descripcion, atributos, recursos)>

<!ELEMENT descripcion (#PCDATA)>

<!ELEMENT tipoOV (#PCDATA)>

<!ELEMENT atributos (atributo+)>

<!ELEMENT atributo (nombre, valor, unidad?, categoria, posicion)>

<!ATTLIST atributo  tipo (texto | numerico) #REQUIRED>

<!ELEMENT nombre (#PCDATA)>

<!ELEMENT nombrePublico (#PCDATA)>

<!ELEMENT valor (#PCDATA)>

<!ELEMENT unidad (#PCDATA)>

<!ELEMENT categoria (#PCDATA)>

<!ELEMENT posicion (#PCDATA)>

<!ELEMENT recursos (recurso*)>

<!ELEMENT recurso (nombre, nombrePublico, tipo, tipoRec, descripcion, ruta, dependencias?)>

<!ELEMENT dependencias (ruta*)>

<!ELEMENT ruta (#PCDATA)>

<!ELEMENT tipoRec (#PCDATA)>

<!ELEMENT tipo (#PCDATA)>
```

El documento objetovirtual.xml contiene toda la información que rellena las fichas del objeto en el museo.

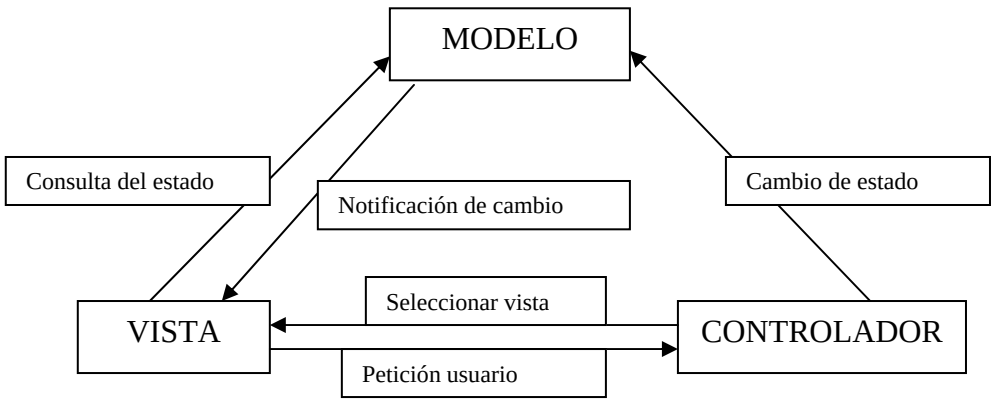
5.3 Arquitectura del sistema

La arquitectura software adecuada para una aplicación interactiva como es el museo sigue el modelo denominado Vista-Controlador (MVC) en el que se lleva a cabo una separación entre el modelo de datos, la interfaz del usuario y la lógica de control, de forma que cada componente puede ser modificado sin afectar excesivamente a los demás.

El modelo representa los datos y las operaciones que permiten el acceso y la modificación de los datos. Normalmente el modelo notifica a la vista los cambios que se producen en los datos.

La vista accede a los datos que contiene el modelo y especifica la forma en que éstos deben ser mostrados. La vista es actualizada a través de las notificaciones de cambio hechas desde el modelo o como resultado de las peticiones del usuario.

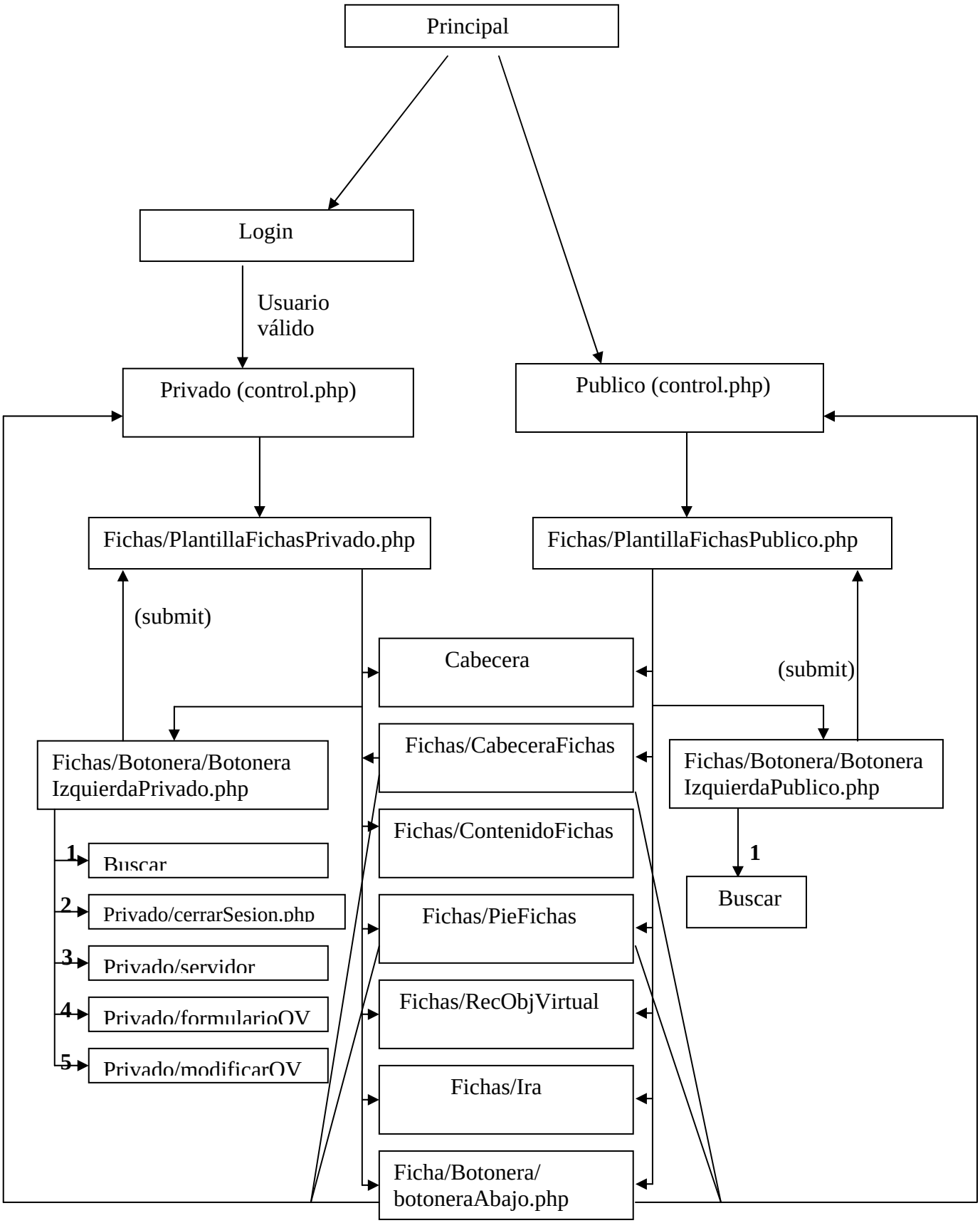
El controlador es el encargado de recibir las peticiones del usuario e interpretarlas para realizar la acción adecuada y seleccionar la vista con la que se va a presentar la información.



Arquitectura del Modelo Vista-Controlador

Este modelo ha sido adaptado a las características del entorno en el que se va a ejecutar la aplicación, la red, recibiendo el nombre de MVC 2.
En el entorno utilizado, la web, el modelo sólo puede responder a peticiones del controlador para generar una vista, pero no puede por sí solo notificar la actualización de la vista cuando se produce una modificación en los datos en el modelo.

Diagrama de correspondencia entre las acciones en el museo y los ficheros donde se implementan (visto desde el directorio raíz del museo):



Leyenda:

1	**Buscar**
2	Cerrar Sesión
3	Subir/Bajar OV
4	Crear OV
5	Modificar OV

El diagrama refleja los principales vínculos que existen entre los directorios y ficheros que forman el museo.

De todos los eventos que puede generar el usuario sólo se muestran aquellos que requieren de un tratamiento específico, indicando en cada caso el directorio o fichero encargado de ejecutar las acciones necesarias.

Una vez mostrado el diagrama anterior es posible comparar y establecer cual es la correspondencia entre la aplicación y el modelo Vista-Controlador (MVC).

- El controlador está representado en los ficheros control.php que existen en las carpetas privado y público. En este caso hay un controlador por cada tipo de usuario que se implementa (con/sin privilegios).

- La vista se encuentra en el directorio Fichas, contiene toda la información para representar la información de manera adecuada en función del usuario que realiza la consulta.

- El modelo se encuentra reflejado en los directorios clases y recursos.
  El directorio clases contiene la forma de manejar la información de la base de datos, mientras que la carpeta recursos almacena parte de los datos que forman el modelo.

El museo sigue el modelo indicado, aunque en la estructura de directorios este esquema no queda reflejado de forma exacta.

El sistema está organizado de forma que cada directorio tiene una funcionalidad perfectamente determinada y es fácil tener una idea de las funciones que se llevan a cabo en cada carpeta.

La organización según la función hace que el modelo MVC no quede perfectamente visible y módulos como el controlador o el modelo se encuentran repartidos entre diferentes carpetas.

1.DIRECTORIO : PRINCIPAL

Contiene las páginas iniciales que dan acceso a la aplicación. Cada uno de los html ofrece acceso a una sección determinada.
La página principal.html da a elegir entre el acceso público, el privado (mediante identificación) y una página de presentación del museo.

Contiene además las imágenes que van rotando en la página principal.html.

2.DIRECTORIO : RECURSOS

Contiene las carpetas que almacenan los recursos propios de un objeto.
Habrá tantas carpetas como objetos haya en la base de datos (aunque no tengan recursos propios).

3.DIRECTORIO : CABECERA

El único fichero que contiene permite visualizar la cabecera común que tiene todas páginas de la aplicación. El directorio existente almacena las imágenes utilizadas en la cabecera. La variable $seccion permite indicar al usuario en qué parte de la aplicación se encuentra (zona pública, privada o de búsqueda en la base de datos).

4.DIRECTORIO : CLASES

Contiene las clases utilizadas para acceder a la base de datos.

Existen dos ficheros, cada uno de los cuales se utiliza en momentos diferentes de la aplicación.

4.1.FICHERO:BASEDATOS

Esta clase se utiliza en el resto de acciones de la aplicación, mostrar un objeto, cambiar de pestaña, crear, modificar y borrar objetos, mostrar recursos, mostrar el resultado de una búsqueda, etc.

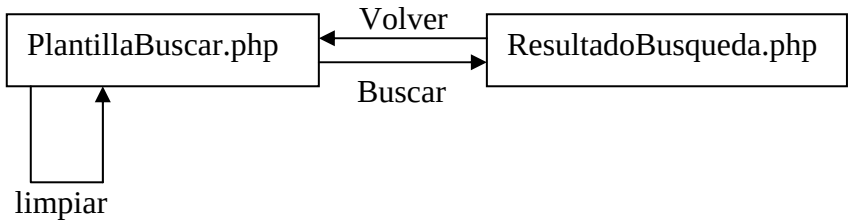
Una parte de las funciones contenidas en esta clase están orientadas a la modificación se objetos y principalmente a la modificación (inserción, eliminación o modificación) de metadatos y recursos.

4.2.FICHERO:BUSCARDATOS

Se utiliza exclusivamente cuando se desea realizar una búsqueda en la base de datos. Contiene todas las funciones necesarias para construir la página en la que se muestran todos los campos por los que se buscar y las funciones para construir la consulta sql adecuada a los datos introducidos.

5.DIRECTORIO : BUSCAR

Contiene los ficheros encargados de gestionar la búsqueda de objetos en la base de datos como resultado de la pulsación de la opción "buscar".



5.1.FICHERO: PLANTILLABUSCAR

Es el fichero que permite construir la página de búsqueda de objetos.
PlantillaBuscar.php (action="./resultadoBusqueda.php" method="Post")
|- include ->/cabecera/cabecera.php
|- include ->buscar/contenidoFichas/Ficha.php
|-include -> buscar/contenidoFichas/fichaRecursos.php

5.2.FICHERO: RESULTADOBÚSQUEDA

Es el fichero que recibe la acción del fichero plantillaBuscar.php.

Con los datos que se pasan desde el formulario se elaboran 4 arrays que permiten elaborar la consulta sql para la búsqueda en la base de datos.
Estos arrays son:

- $array1: almacena los atributos de texto y numéricos cuyos valores haya rellenado el usuario. Para cada atributo se guarda el nombre, el valor y el tipo del atributo (texto o numérico).
  Ej: $array1["nombre"]="fabricante"
  $array1["valor"]= "intel"
  $array1["tipo"]="texto"

- $arrayUnid: almacena información sobre las unidades de los atributos numéricos que tengan este campo relleno.
  Ej: $arrayUnid["nombre"]="Alto"
  $arrayUnid["unidad"]= "cm"

- $array2: almacena la información de los recursos, tiene como campos el nombre y el valor. Los posibles valores para el campo nombre son: tipo, tipoRec, dependencia, ruta, nombre_rec.

  En el caso de las dependencias, si el usuario ha indicado varias dependencias en un mismo campo, se separa cada una de ellas y se añaden al array por separado.

  Ej: $array2[“nombre”]=”propietario”
  $array2[“valor”]= “/”

- $array3: almacena información sobre los metadatos rellenados por el usuario. Posee los campos nombre, valor y ficha. El valor del campo ficha indica el tipo de metadato, sus valores posibles son “catalogación”, “ciclo”, y ”clasificacion”.

  Ej: $array3[“nombre”]=”Fecha”
  $array3[“valor”]= “2003”
  $array3[“ficha”]= “ciclo”

Una vez que se tienen formados estos arrays se pasa a formar la consulta sql y se ejecuta, obteniendo los identificadores de los objetos que cumplen todas las restricciones indicadas por el usuario.

Con el resultado de la consulta se vuelve a consultar la base de datos para mostrar la lista de los resultados indicando el tipo de recurso, la descripción en caso de tenerla y el identificador del objeto. Acompañado a cada objeto aparece una imagen que permitirá visualizar el objeto tras pulsar en el. Esta imagen varía según el objeto, se mostrará el primer recurso(sea propio o no) que sea una imagen, en caso de carecer de recursos o que ninguno de ellos sea una imagen se mostrará una imagen por defecto.

5.3.DIRECTORIO: CONTENIDOFICHAS

Contiene los ficheros que permiten elaborar la página de consulta a la base de datos.

Contiene los siguientes ficheros:

5.3.1.Fichero: Ficha

Muestra todos los atributos que existen en la base de datos, agrupándolos por su valor en el campo ‘posición’ de las tablas ‘atributos_texto’ y “atributos_numericos”. Después de los atributos muestra los recursos y posteriormente los metadatos. Los metadatos de tipo lifecycle se ponen fijos mientras que los nombres de los metadatos de tipo catalog y classification

pueden ser consultados fácilmente en la base de datos ya que aparecen en el campo "contenido" de la tabla "metadatos".

5.3.2.Fichero: FichaRecursos

Muestra los cinco campos por los que se puede buscar un recurso. Se puede indicar el nombre del recurso, el tipo, la extensión que se busca para un archivo, el propietario y las dependencias.

Los valores válidos para el campo propietario son la palabra "propio" para indicar que se desean buscar objetos que tengan recursos propios o valores numéricos si se desean buscar objetos con recursos no propios pertenecientes al objeto cuyo identificador se indica.

El campo dependencias es un textArea en el que se puede indicar un número variable de dependencias, cada una de las cuales será separada por "\n" (pulsaciones de la tecla Intro).

Los nombres del atributo name en los input de estos ficheros debe contener información suficiente para poder construir los arrays del fichero "resultadoBusqueda.php". estos nombres están formados por un número y el nombre identificativo del input. Los valores estás separados por el carácter "\r" para facilitar la separación de la información cuando se construyan los arrays, en caso de que los nombres tengan espacios en blanco, cada uno de estos espacios será reemplazado por la secuencia "|_|" ya que de otra forma al hacer submit del formulario éstos caracteres desaparecen. Cuando se formen los arrays esta secuencia será sustituida de nuevo por el carácter un espacio en blanco.

El número que acompaña a los nombres permiten saber cuando se está tratando un metadato, un recurso o un atributo. Así, en el caso del fichero "FichaRecursos.php", el número es uno más que el número de valores diferentes que hay en el campo posicion de las tablas 'atributos_texto' y 'atributos_numericos'. En "Ficha.php", cuando se tratan los atributos, el número indica el valor del campo posición que tiene cada atributo en su tabla, mientras que cuando se tratan metadatos se actúa como en el caso de los recursos.

Si indica a través de campos hidden en que número empiezan los metadatos y cuantas fichas de atributos de texto y numéricos hay en total, esto facilita la construcción de los arrays para la creación de la consulta.

6.DIRECTORIO : LOGIN

Lleva a cabo el control del acceso a la parte privada de la aplicación, se llega a él a través de la página "principal/investigadores.html".

El fichero login.php ofrece al usuario un formulario para introducir su nombre de usuario y su contraseña.

La verificación de los datos se lleva a cabo en el fichero "verificaLogin.php". Consulta en la tabla 'usuarios' de la base de datos la corrección de los datos introducidos, en caso de ser válido se crea una sesión almacenando en ella el identificador de la sesión que se va a mantener hasta que se cierre la misma. La existencia de este identificador en el fichero de la sesión permitirá entre accesos a zonas privadas realizados por usuarios con sesión abierta de aquellos que intenten acceder a estas zonas y no tengan permiso para ello.

Además de almacenar este identificador, se almacena el nombre y la contraseña del usuario (actualmente no se utilizan estos datos pero pueden ser útiles en caso de tener usuarios con distintos privilegios), así como el identificador y el tipo del primer objeto almacenado en la base de datos para evitar problemas cuando la primera acción que se lleva a cabo después de iniciar sesión es modificar un objeto.

7.DIRECTORIO : PUBLICO

Lleva a cabo el control de las peticiones que se lleva a cabo en la sección pública de la aplicación. El fichero que lleva a cabo este control es "control.php". Su estructura es la misma que la de "privado/control.php" en cuanto a las variables utilizadas, la única diferencia es la menor cantidad de acciones que puede llevar a cabo. El único valor posible de la variable $accion es "irA", en cuyo caso lee la variable $irAIdObjeto y visualiza el objeto indicado por la variable (en caso de existir).

Contiene además un fichero de error, "error.php", utilizado para indicar cualquier tipo de fallo que se produzca en la zona pública, por ejemplo, este fichero es requerido por el archivo "login/verificaLogin.php" para indicar que los datos introducidos como usuario y contraseña no se corresponden con los de un usuario válido.

8.DIRECTORIO : PRIVADO

Es el directorio en el que reside toda la funcionalidad que los administradores de la aplicación pueden tener. El contenido de este directorio permite realizar la creación, modificación, almacenamiento y descarga de los objetos que son utilizados por la aplicación. El control sobre el acceso a los ficheros contenidos en el directorio se realiza mediante sesiones, de forma que para poder acceder a alguna de las funcionalidades comentadas anteriormente es necesario haberse identificado correctamente como administrador de la aplicación.

En caso de intentar acceder a cualquier página de este directorio sin el permiso adecuado, es decir, se carece de un identificador de sesión, la petición será rechazada.

Este directorio consta de varios subdirectorios claramente diferenciados:
Servidor: subida y almacenamiento en la base de datos, así como la descarga de un objeto.

FormularioOv: creación y empaquetamiento de un objeto
ModificarOV: modificación de un objeto de la base de datos

Temporal: directorio en el que se almacena temporalmente archivo utilizados durante el proceso de creación, modificación, descarga y subida de objetos virtuales.

Además existen otros archivos que son de uso general, por lo que no están incluidos en ninguno de los anteriores directorios.

8.1.FICHERO: CERRARSESION

Actúa cuando el usuario a pulsado la opción de la botonera "cerrar sesión", se encarga de eliminar el contenido del directorio temporal de sesión, así como de eliminar el propio directorio.

Cierra la página de la aplicación que permite el control como administrador y se abre una nueva mostrando la página inicial de la aplicación (principal/principal.html).

El cierre de la sesión provoca la pérdida del identificador de sesión, por lo que si se pulsa cuando hay otras ventanas abiertas relacionadas con las funciones del administrador (creación o modificación de objetos), éstas dejarán de funcionar indicando un error por falta de permisos para llevar a cabo esas funciones.

8.2.FICHERO: ELIMINARBASURA

Este fichero se encarga de borrar todo el contenido del directorio de sesión temporal, pero sin eliminar el propio directorio. Es llamado por el fichero reconstruir.php de la carpeta "servidor" y por los ficheros rellenarManifiesto.php de los subdirectorios de la carpeta "formularioOV".

8.3.FICHERO: ERRORPRIVADO

Responsable de mostrar un mensaje de error procedente de alguno de los ficheros que forman parte del directorio "privado". El mensaje a mostrar estará relacionado con la falta de permisos de un usuario, es decir, carencia de identificador de sesión.

8.4.FICHERO: CONTROL

Es el fichero que coordina a todos los archivos relacionados con el directorio "privado".

Maneja las variables : $posición, $tipoOrden , $ordCampo, $Ficha, $FichaAux, $accion y $irAIdObjeto.

Las variables $ordCampo y $tipoOrden controlan el orden en el que se muestran las fichas de los objetos. Actualmente estos valores siempre están fijos y permiten mostrar los objetos ordenados por el campo idov de la tabla 'objeto_virtual' de forma creciente.

La variable $Ficha indica qué pestaña del objeto actual se está mostrando al usuario, cada vez que se cambia de objeto el valor de esta variable vuelve a 1 por defecto, no se recuerda que pestaña se estaba visitando en la ficha anterior. Esto facilita la presentación de los objetos ya que para cada objeto el número de pestañas es variable y resulta difícil asociar el valor de la variable $Ficha con el nombre de la pestaña.

Para entender mejor el contenido de la variable $Ficha ver el apartado dedicado a los directorio fichas/cabeceraFichas y fichas/pieFichas.

La variable $posición contiene la posición en la base de datos del objeto mostrado. Esta variable permite desplazarse al anterior objeto y al siguiente en función del criterio de ordenación indicado por las variables $ordCampo y $tipoOrden.

Actualmente esta variable es una unidad inferior al campo idov del objeto, pero resultará útil en caso de que los criterios de ordenación puedan ser elegidos por el usuario.

La variable $accion contiene información sobre la petición realizada por el usuario.

Comportamiento del fichero:

1. Se calcula el número de objeto que hay en la base de datos ($numFilas), en caso de que esté vacía se indica al usuario la ausencia de información y se le ofrece la posibilidad de crear nuevos objetos mediante la acción “nuevo”.

2. En caso de que haya información en la base de datos se pasa a considerar los distintos valores de la variable $accion:

- $accion=irA: lee el valor numérico de la variable $irAIdObjeto y consulta en la base de datos la existencia de un objeto con el identificador indicado. En caso de existir se obtiene su posición ($posicion) y se muestra al usuario. Si no existe se indica al usuario el fallo.
- $accion=actualizar: se consulta a la base de datos por el objeto con identificador más alto para mostrarlo al usuario. Esta acción permite ver el último objeto subido a la base de datos.
- $accion=borrar: permite borrar el objeto que se está visualizando en ese momento. Para poder borrar un objeto es necesario que ningún otro objeto de la base de datos tenga entre sus recursos o dependencias de recursos a este objeto o alguno de sus recursos propios. En caso de no poderse borrar se indica al usuario el motivo. Si es posible el borrado se eliminará de la base de datos, borrando además el directorio que posee en la carpeta “recursos”. Cuando se borra un objeto se pasa a mostrar el objeto anterior.
- $accion=modificar: el usuario desea modificar el objeto que se está mostrando actualmente en la aplicación, para ello se almacena en el fichero de la sesión el identificador y el tipo del objeto.
- Independientemente de la acción a realizar se llevan a cabo los cálculos necesarios para poder visualizar un objeto (ya sea el mismo que se estaba visualizando o uno nuevo indicado por las acciones “actualizar” o “irA”) y averiguar la pestaña que va a aparecer activa. La pestaña activa será la de menor posición de las que aparecen en las tablas ‘atributos_texto’ y

'atributos_numericos', en caso de no existir ninguna de ellas se elegirá el valor 1 por defecto (en este caso la pestaña activa será la primera pestaña que exista ya sea de recursos o de metadatos).

8.5.DIRECTORIO : TEMPORAL

Contiene un directorio por cada sesión abierta en la aplicación y su nombre se corresponde con el identificador de sesión que el usuario ha obtenido en el momento de hacer el login como usuario privilegiado (administrador).

De esta forma cada usuario tiene un espacio privado en el servidor para llevar a cabo sus acciones sin interferir en las acciones que pudieran estar realizando otros usuarios privilegiados en ese momento.

Las acciones que van a hacer uso de este directorio son: "subir", "bajar" y "modificar" objetos.

Todas estas peticiones necesitan almacenar ficheros en el servidor de manera temporal.

Los directorios se crean ante alguna de estas peticiones y se mantiene durante toda la sesión hasta que el usuario se desconecte de la aplicación (acción "cerrar sesión").

NOTA: Este directorio temporal debe ser controlado regularmente por el administrador ya que existe la posibilidad de que los usuarios con privilegios no siempre cierren la sesión, por lo su directorio personal no se eliminará e irá consumiendo espacio en el servidor.

8.6.DIRECTORIO : SERVIDOR

La carpeta servidor se encarga de responder a las peticiones de subida y bajada de objetos por parte del usuario con los permisos suficientes para realizar estas peticiones.

Cuando el usuario desea subir un objeto, se encarga de la comprobación y el almacenamiento de la información en la base de datos.

Ante las peticiones de descarga de un objeto, construye un zip incorporando todos aquellos objetos que de forma directa o indirecta son referenciados por el objeto pedido.

Contiene otros archivos necesarios para la reconstrucción de los objetos como son los esquemas de IMS, la dtd del objeto virtual así como la xsl que permite visualizar fácilmente la información descargada.

Los ficheros que actúan como raíz de todos los demás son subirObjeto.php y bajarObjeto.php, a partir de ellos se lleva a cabo la subida y la bajada de objetos respectivamente.

La relación entre los diferentes ficheros que componen el directorio se muestra a continuación:

SubirObjeto.php (method=post, action= tratar_objeto.php)
tratar_objeto.php
|---require->basedatos.php
|---include->descomprimir.php
|---require->tratar_atrib.php
|---require->tratar_metadatos.php
| |--- include -> class_path_parser.php
|---require->tratar_recursos.php
| |--- include -> class_path_parser.php
|---require->tratar_recurs_dep.php
bajarObjeto.php (method=post, action= reconstruir.php)
reconstruir.php
|
|---require->basedatos.php
|
|---require->construirXML.php
|
|---require->construirXMLdtd.php
|
|---require->comprimir.php

8.6.A. SUBIDA DE UN OBJETO VIRTUAL E INTRODUCCIÓN DE LA INFORMACIÓN EN LA BASE DE DATOS.

- ***SubirObjeto.php***

Muestra al usuario un formulario mediante el cual se indica el fichero zip que se quiere subir al servidor y que debe contener un objeto virtual creado mediante la herramienta correspondiente. El resultado de la acción se pasa al fichero servidor/tratar_objeto.php mediante el método post.

- ***tratar_objeto.php***

Es el fichero que coordina la acción de todos los demás ficheros relacionados con la subida y almacenamiento de la información.

La secuencia de pasos a realizar es la siguiente:

1. Se comprueba si el usuario de la sesión tiene asignado en el directorio temporal un directorio personal en el que almacenar los ficheros necesarios para realizar la petición del usuario. En caso de no existir se crea con el identificador de la sesión.

2. Puesto que se va a subir un objeto, se ha de eliminar todo aquello que proceda de anteriores subidas y que pueda interferir en el tratamiento del nuevo objeto. Para ello se elimina el directorio llamado "objeto" y el zip del mismo nombre en caso de existir ya que el nombre "objeto" es el se utiliza por defecto durante el procesos de subida de un objeto.

3. Se comprueba que el fichero subido es de tipo zip, esta comprobación varía en función del sistema operativo utilizado por el usuario (windows, linux). Si el fichero subido al servidor es del tipo adecuado se descomprime y comienza la comprobación y el almacenamiento de los datos que hay en él:

4. Se comprueba que existen los ficheros xml correspondientes al manifiesto y al objeto virtual (imsmanifest.xml y objetoVirtual.xml).La variable $es_valido va a indicar si hay un error en este punto.

5. A continuación se consulta a la base de datos cual es el identificador que debe tener el objeto que se ha subido. El nuevo objeto tendrá como identificador un número más que el valor más alto en ese momento en la base de datos.

6. Se crea una carpeta en el directorio de recursos para almacenar los recursos propios que tenga el objeto. En caso de que exista ya una carpeta con el mismo nombre se su suspende el almacenamiento del objeto. El nombre de los directorios sigue la secuencia OVX , donde X es el identificador que le corresponde al objeto.

7. Se crea una carpeta en el directorio de recursos para almacenar los recursos propios que tenga el objeto. En caso de que exista ya una carpeta con el mismo nombre se suspende el almacenamiento del objeto. El nombre de los directorios sigue la secuencia OVX , donde X es el identificador que le corresponde al objeto.

8. Se almacenan en la base de datos los atributos que se encuentran en objetovirtual.xml y los metadatos que aparezcan en imsmanifest.xml. Para ello se usan los archivos servidor/tratar_atrib.php y servidor/tratar_metadatos.php.

9. Se comprueban los recursos propios del objeto (imsmanifest.xml) y se almacenan en caso de ser correctos. Se usa el fichero servidor/tratar_recursos.php. A través de la variable

$faltan_rec_propios se indicará si existen en el zip todos los recursos declarados en el manifiesto.

10. Se comprueban los recursos no propios del objeto y las dependencias a otros objetos o recursos que tengan los recursos propios. Esta información aparece indicada en objetoVirtual.xml, se usa el fichero servidor/tratar_recurs_dep.php. La variable $malas_dependencias indicará la corrección de estos datos.

11. Mientras no se detecten fallos, los datos y recursos se van almacenando.En el momento en que se detecta un problema todo lo que se ha almacenado hasta ese momento es eliminado de la base de datos, desapareciendo además la carpeta creada para ese objeto en el directorio recursos.

El proceso termina cuando el paquete es totalmente válido, en cuyo caso se habrá almacenado toda la información en la base de datos y los recursos en una carpeta propia, o bien cuando se produce el primer fallo en alguna de las comprobaciones realizadas en el objeto subido.

En cualquiera de los casos, fallo o éxito, se eliminan todos los ficheros y directorios auxiliares (/temporal/session-id/objeto.zip y /temporal/session-id/objeto) usados para el tratamiento del objeto en el servidor.

Nota: Sólo se almacenan recursos propios, no se guardan los ficheros imsmanifest.xml ni objetoVirtual.xml.

- ***descomprimir.php***

Comprueba que el zip ha llegado al servidor, en caso de que no haya sido así segenera un error ($hay_error) que suspende el tratamiento del objeto.

Si ha llegado bien al servidor, el fichero se copia en el directorio “temporal/session-id” bajo el nombre de “objeto.zip” y se descomprime creando el directorio “/temporal/session-id /objeto”.

- ***tratar_atrib.php***

Parsea mediante DOM el archivo objetoVirtual.xml, almacenando en la tabla ‘objeto_virtual’ el identificador del objeto y su descripción, en caso de tenerla.

Además almacena los datos referidos al propio objeto en las tablas ‘atributos_texto’ y ‘atributos_numericos’, se trata de todos los atributos del objeto a excepción de los metadatos.

El almacenamiento de los atributos en la tabla atributos_texto o atributos_numericos se indica por el campo tipo que poseen todos los atributos de un objeto.

El fichero objetoVirtual.xml está codificado en UTF-8 por lo que debe ser decodificado a ISO-8859-1 para almacenar la información en la base de datos respetando el contenido original en caso de usar palabras con acentos o eñes.

Se decodifican sólo aquellos nodos cuyo contenido depende del usuario, otros como el tipo que viene marcado en la dtd no es necesario decodificarlo.

- ***tratar_metadatos.php***

Parsea el archivo imsmanifest.xml que contiene los metadatos y los almacena en las tablas 'metadatos' y 'atributos_metadatos'.

Los metadatos que se desean tratar son indicados en este fichero mediante sus rutas completas según el estándar de IMS para metadatos. Se indican todos los metadatos posibles que puede tener un manifiesto aunque los objetos de esta aplicación sólo contienen algunos de esos metadatos (general, lifecycle y classification).

El parser no se realiza a través de DOM sino mediante el fichero servidor/ class_path_parser.php que ofrece información necesaria para la posterior modificación y descarga del objeto y que deberá ser almacenada en la base de datos.

El manifiesto imsmanifest.xml está codificado directamente en ISO-8859-1 por lo que no es necesario hacer la transformación vista en el fichero tratar_atrib.php antes del almacenamiento de la información en la base de datos.

- ***tratar_recursos.php***

Parsea el archivo imsmanifest.xml que contiene los recursos propios del objeto (es decir, recursos cuyo propietario es el objeto que se esta tratando) y almacena la información en la tabla 'recursos'.

Si el recurso que se trata es correcto se almacena en el directorio de recursos del objeto al que pertenece.

En caso de que haya un recurso declarado en el manifiesto pero que no esté en el zip que el usuario ha subido se produce un fallo ($faltan_recursos_propios) y se suspende el almacenamiento del objeto.

En el manifiesto pueden aparecen declarados otros recursos como el xml en el se declaran los atributos (objetoVirtual.xml) y su dtd (objetovirtual.dtd), el xsl que permite visualizar la información de

objetoVirtual.xml y un GIF correspondiente al fondo aplicado por la xsl. Estos ficheros no se tratan como recursos propios de ningún objeto.

Para encontrar los recursos propios se parsea usando DOM.

- ***tratar_recurs_dep.php***

Parsea mediante DOM el archivo objetoVirtual.xml, almacenando en la tabla 'recursos' los datos de los recursos que no son propios del objeto pero de los cuales hace uso.

Además almacena en la tabla 'dependencias' las dependencias que puedan tener los recursos propios. Si el recurso es no propio se comprueba que en la base de datos exista ese recurso en el objeto virtual que se ha declarado.

Si se trata de una dependencia se comprueba si existe el recurso del cual depende antes de almacenarlo en la base de datos.

En caso de que alguna de estas comprobaciones no se cumpla se suspende el almacenamiento del objeto (esto se ve reflejado en la variable $malas_dependencias).

En este archivo también se rellena en la base de datos la información de los campos "descripción" y "tipoRec" de la tabla "recursos" para cada recurso que contiene un objeto virtual, tanto si es propio como si no lo es.

Un objeto virtual puede tener como recursos no propios a recursos de otros objetos o a otros objetos enteros. Lo mismo sucede con las dependencias de los recursos propios.

- ***class_path_parser.php***

Es el fichero encargado de parsear ficheros xml cuando no se utiliza DOM.

Los ficheros que lo utilizan indican qué rutas del fichero xml debe reconocer y que función debe seguir cuando encuentre la estructura pedida.

Actúa sobre el fichero imsmanifest.xml reconociendo metadatos o recursos propios del objeto. En el caso de los metadatos ofrece información acerca del orden que tienen los metadatos en el manifiesto original y que se almacenará en la base de datos para posteriormente poder reconstruirlo. Esta información es útil también durante la modificación de un objeto para poder insertar o eliminar cualquier metadato y que se siga manteniendo el orden indicado por IMS.

8.6.B. DESCARGA DE UN OBJETO VIRTUAL Y RECONSTRUCCIÓN DE LOS FICHEROS XML.

- ***bajarObjeto.php***

Mediante un formulario se pide al usuario que introduzca el identificador del objeto virtual que quiere descargar.

El resultado de la acción se pasa al fichero servidor/reconstruir.php mediante el método post.

- ***reconstruir.php***

Se consulta a la base de datos para ver si existe algún objeto con el identificador pedido. En caso de que el identificador introducido no exista en la base de datos se informa al usuario del error cometido (se controla con la variable $existe) y se le da la posibilidad de regresar a servidor/bajarObjeto.php para intentar de nuevo la descarga.

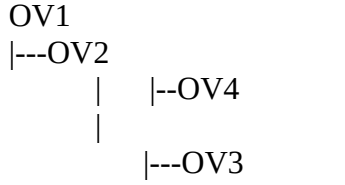
En caso de que exista, la copia del objeto pedido es controlada por este archivo. No se realiza ninguna comprobación en este proceso ya que se supone que la información en la base de datos es fiable. Los ficheros que se encargan de la reconstrucción de los archivos imsmanifest.xml y objetoVirtual.xml, así como de la copia de los recursos, son servidor/construirXML.php y servidor/construirXMLdtd.php respectivamente.

Cuando se pide descargar un objeto se elabora una lista con los identificadores de todos los objetos virtuales de los que depende directa o indirectamente el objeto pedido.

Es decir, si el objeto tiene entre sus recursos no propios a objetos virtuales completos, esos objetos también son incorporados a la lista para poder ser descargados junto con el objeto pedido. Este hecho no sólo afecta a los objetos no propios de ese objeto, el proceso se repite con todos los objetos de la lista, por lo que en el peor de los casos, si todos los objetos están relacionados entre sí, sería posible descargarse todos los objetos almacenados en la base de datos.

De forma gráfica:

Dado un objeto a descargar (OV1) con recursos no propios (OV2, OV3) como el siguiente:



La descarga generaría un directorio principal bajo el nombre descargaOV1, dentro de él estarían los directorios de los objetos OV2,

OV3 y OV4 al mismo nivel que el resto de los recursos del objeto pedido por el usuario.

Pasos a seguir:

1. Se comprueba si el usuario de la sesión tiene asignado en el directorio temporal un directorio personal en el que almacenar los ficheros necesarios para realizar la petición del usuario. En caso de no existir se crea con el identificador de la sesión.

2. Puesto que se va a bajar un objeto el nombre que se va a poner por defecto es descargaOVX, donde X es el identificador del objeto pedido, por ello, si existe un directorio con ese nombre debe ser eliminado, para que no pueda interferir en el tratamiento de la petición, y posteriormente creado de nuevo.

3. Se crea la lista de todos los identificadores de los objetos que se van a descargar y se pasa a reconstruir cada uno de ellos.

4. Una vez que se ha creado el directorio que contiene la copia (/temporal/session-id/descargaOVX) se comprime bajo el nombre (/temporal/session-id/descargaOVX.zip) renombrando el directorio descargaOVX por el de "objeto" y se pone a disposición del usuario para que pueda descargarlo si lo desea. Para comprimir el directorio se recorre recursivamente su contenido y se va añadiendo al zip que se está construyendo.

Para borrar los ficheros y directorios auxiliares creados durante la elaboración de la copia se recurre al fichero privado/eliminarbasura.php. La orden para que se lleve a cabo este borrado es realizada por el propio usuario cuando pulsa el botón "Finalizar" del formulario o cuando cierra la pantalla desde la que se ha descargado el zip (activándose el evento unLoad). El botón Finalizar resulta útil cuando en un sistema no se activa el evento unload.

- ***construirXML.php***

Reconstruye el manifiesto (imsmanifest.xml) con los metadatos y los recursos propios del objeto virtual, realizando además una copia de esos recursos. Se construye usando la codificación ISO-8859-1.

Cuando se descarga un objeto, la lista de recursos propios aumenta, aparecen como nuevos recursos propios los recursos que aparecían como pertenecientes a otros objetos.

Así, si el objeto a descargar tiene como recurso no propio otro objeto virtual, aparecerá como recurso propio el manifiesto a ese objeto, que también se va a descargar.

En caso de que los recursos no propios sean ficheros normales, no objetos enteros, se añadirá ese recurso como recurso propio descargándose una copia del mismo.

Además de añadir los recursos propios del objeto se añade al manifiesto la xsl, la dtd, el gif usado por la xsl y el fichero objetoVirtual.xml.

- ***construirXMLdtd.php***

Reconstruye objetoVirtual.xml con los atributos del objeto virtual, los recursos y las dependencias de los recursos propios. El fichero resultante es válido y bien formado.

El xml que se reconstruye es idéntico al que se generó en la creación del objeto. A diferencia de las modificaciones hechas en el imsmanifest.xml respecto a los recursos no propios, en este xml no se modifica nada, siguen apareciendo como recursos no propios los que se indican en la base de datos.

La información no es consistente entre los dos archivos, esto puede producir problemas cuando se sube al servidor un objeto que se había descargado anteriormente. En este caso, el nuevo objeto tendría como recursos propios los que se indican en el manifiesto pero seguiría manteniendo los recursos no propios del objeto original que se indican en objetoVirtual.xml. La solución sería modificar el nuevo objeto una vez subido para eliminar los recursos a los objetos que ya son propios.

Este fichero se encarga de hacer la copia de los recursos según se va reconstruyendo ObjetoVirtual.xml. Además se copia la dtd, el fondo de la xsl y la xsl. Hay dos xsl en el directorio servidor, uno de ellos es el que aparece en el directorio del objeto pedido por el usuario y el otro se almacenará en el directorio de cada uno de los objetos que se van a descargar con el que se ha pedido. La única diferencia entre ellos es la forma de acceder a los manifiestos de otros objetos a los que hacen referencia (es decir, sus rutas), el objeto principal tiene a los otros objetos como hijos suyos (./OVX), mientras que los otros objetos son hermanos entre sí (../OVX).

- ***comprimir.php***

Comprime el directorio "/temporal/session-id/descargaOVX" en un zip, dando como resultado el fichero comprimido que el usuario podrá descargarse.

PROBLEMAS:

- Cuando se descarga un objeto el ov.xml no es coherente con el manifiesto en los nuevos recursos propios.

- Cuando se descarga un objeto que a su vez contiene a otro y esos dos se hacen llamadas recursivas mutuas , las referencias del segundo objeto al padre no van a funcionar porque la xsl siempre apunta a un hermano, no al padre.

8.7.DIRECTORIO : FORMULARIOOV

En este directorio se gestionan todas las acciones relacionadas con la creación de un nuevo objeto virtual. Es un módulo independiente al resto de la aplicación y sólo está disponible para el usuario con los privilegios para crear objetos.

El contenido de este directorio es:

- Piezas
- Documentos
- Fotos
- Images
- Principal.php
- Comprimir_aux.php
- Otros ficheros

Los directorios piezas, documentos y fotos permiten crear tres tipos de objetos diferentes. Aunque los tres directorios tienen el mismo funcionamiento aparecen diferenciados para poder individualizar el contenido de las páginas que permiten crear el objeto. Cada tipo de objeto presenta unas determinadas características por defecto, algunas de estas características son comunes a todos los objetos (descripción, clasificación, ...), en cambio otras no tienen sentido para todos (ancho, número de páginas, ...). Diferenciando los objetos por directorios es posible manejar información coherente para cada objeto.

Puesto que el funcionamiento es el mismo en todas las carpetas, se explicará con detalle sólo una de ellas, del resto sólo se indicarán las características que existen por defecto.

8.7.1.DIRECTORIO: PIEZAS

Este directorio está formado por los ficheros php necesarios para la obtención de las características del objeto y el directorio “construirMani” que realiza el proceso final para la construcción del objeto.

8.7.1.1.Fichero: Piezas

Es el fichero que articula a todos los demás en la construcción del objeto virtual.

Muestra una lista de botones con los grupos en los que se han dividido las características de los objetos de tipo pieza.

Sus funciones son:

- En caso de no existir crea un directorio en la carpeta temporal para que cada usuario pueda actuar libremente sin interferencias con otros posibles usuarios con privilegios, se trata de una aplicación multiusuario.

- Cada una de los botones que forman la página se corresponde con un formulario que da acceso al listado de características que existe en cada grupo. Este formulario contiene información que permite saber si el usuario ya ha modificado alguna de las características de este grupo. Este dato es el que permite que el usuario pueda moverse entre los distintos grupos de características tantas veces como quiera sin que exista pérdida de información.

- Contiene un campo de texto para indicar el nombre (no hace falta incluir la extensión) que se desea dar al zip donde se va a almacenar el objeto que se ha creado. Cuando se pulsa sobre el botón "Crear" se pasa a formar el zip con la estructura propia de un paquete IMS. Se pasa el control al fichero construirObjVirtual.php del directorio construirMani.

- La principal acción que realiza es la gestión de los datos que el usuario va indicando a lo largo del proceso de creación del objeto.

El almacenamiento de la información aportada por el usuario se realiza a través de ficheros de texto. Existen tantos ficheros como grupos de características ha modificado el usuario, cada archivo contiene la información de cada grupo. En el caso de un objeto de tipo pieza existen 7 grupos:

1. Campos Generales
2. Características Físicas
3. Características tecnológicas
4. Recursos
5. Catalogación
6. Ciclo de vida
7. Clasificación

Los nombres de los ficheros auxiliares se corresponden con el número de cada grupo, así por ejemplo, las características tecnológicas estarán almacenadas en el fichero piezas3.txt.

Cuando se sale de un grupo de características se actualiza el fichero correspondiente, es decir, se elimina y se vuelve a crear con los últimos datos que el usuario a aceptado al salir de la página del grupo.

La forma en la que se almacena cada una de las características consiste en una secuencia de cadenas con los datos del usuario separados por caracteres especiales para no tener que poner restricciones en los datos que puede aportar el usuario.

Cada característica está formada por varios campos, como mínimo el nombre de la característica y el valor dado a la misma. Determinados grupos engloban a características de tipo numérico, por ejemplo "Características tecnológicas", por lo que existe un tercer campo para indicar las unidades de medida de cada característica.

Cada uno de los campos que forman una característica se introducen en el fichero de texto separados por el carácter "\r", mientras que el carácter "\0" permite separar las características del mismo grupo entre sí.

La presencia de campos vacíos también debe ser reflejada en estos ficheros.
Ej: Ancho"\r"20"\r"cm"\0"Alto"\r""\r"cm"\0"

El almacenamiento de los datos en el caso del grupo de Recursos tiene un tratamiento diferente al resto ya que en este caso la forma de almacenar la información en el fichero no se corresponde exactamente con la forma en que se visualizan los datos en la página que el usuario modifica.

En este caso se tienen dos ficheros de texto en lugar de uno como en el resto de los grupos.

El fichero piezas4nomFicha.txt contiene únicamente el nombre de la ficha.

El fichero piezas4.txt es el que contiene toda las información relativa a los recursos del objeto.

La información de cada recurso se almacena según la secuencia
$nombre."\0".$tipo."\0".$valor."\0".$num_depen."\0".$ponruta."\0".$lista_depen."\0".$tipoRec."\0".$descripcion."\0"

Esta secuencia indica:
$nombre: nombre del recurso.
$tipo: extensión del fichero en caso de existir o la palabra "OV" cuando el recurso sea un objeto virtual entero.

$valor: nombre del fichero o "OV" en caso de hacer referencia a un objeto virtual completo existente en la base de datos.
$num_depen: número de dependencias indicadas en el recurso.
$pon_ruta: indica si el recurso es propio, en cuyo caso se indica con el carácter "/", o en caso contrario se indica el identificador del objeto del que se toma el recurso (ej. 2/).
$lista_depen: lista con las dependencias indicadas por el usuario, separadas por comas. Se han de eliminar los caracteres extra que introduce el salto de línea en el textarea.
$tipoRec: tipo indicado por el usuario.
$descripcion: descripción del recurso en caso de existir.
Al contrario que en los casos anteriores, no se usan caracteres distintos para distinguir entre el fin de un campo del fin de un recurso, en todos los casos de usa el carácter "\0", esto es posible porque se conoce el número y orden de los campos y se almacenan todos los campos aunque estén vacíos.

8.7.1.2.Fichero: f1piezas

Corresponde al grupo de las características generales de la pieza, se encarga de mostrar todas las características que hay en cada momento en el grupo seleccionado.

Se pueden dar tres situaciones diferentes:

- Es la primera vez que se entra en la página desde el fichero piezas.php: en este caso se inicializa una matriz con el nombre y valor (cadena vacía) de las características por defecto que existen en el grupo.

- No es la primera vez que se accede a la página desde piezas.php: se consulta la existencia del fichero piezas1.txt, que debe existir, y se lee su contenido, reconstruyendo los nombres y valores de cada una de las características que el usuario aceptó la última vez que se visitó esta página. La reconstrucción se realiza mediante la detección de los caracteres especiales "\0" y "\r" que permiten separar los distintos campos tal como se explicó para el fichero piezas.php. Cuando se ha recorrido todo el fichero se forma una nueva matriz ($matrizatributos) que será la matriz que se manejará durante todo el proceso y que no es más que una copia de la que se ha rellenado durante la lectura del fichero. Además se contabiliza el número de atributos (características) que existen en cada momento.

- Se accede a la página procedente del fichero f1piezascajas.php: se actualiza la matriz $matrizatributos con los valores procedentes del fichero indicado a través del array POST.

Una vez que se ha creado la matriz, se visualiza el contenido de la misma, esto permite al usuario ver los datos de nuevo y decidir si los acepta definitivamente o no.

Existen tres botones:

- Editar: se vuelve al fichero f1piezascajas.php para poder seguir haciendo modificaciones en los atributos. Pasa el contenido de la matriz $matrizatributos por el método POST para que se puedan mantener los datos a lo largo de las distintas páginas.

- Aceptar: se pasa el control al fichero piezas.php, pasando a travñes del método POST todos los atributos de la matriz $matrizatributos para que con ellos se actualice el fichero piezas1.txt.

- Cancelar: se vuelve al fichero piezas.php perdiendo todas las modificaciones que se hayan podido hacer desde que se entró en esta página, manteniendo las antiguas.

El resto de los ficheros del directorio del tipo fxpiezas.php tienen el mismo comportamiento que el comentado para f1piezas.php.

El fichero f4piezas.php lleva a cabo más acciones que el resto de los ficheros del tipo fxpiezas.php, esto se debe a la información que trata este fichero, los recursos del objeto.

Independientemente del fichero del que se proceda (piezas.php o f4piezascajas.php) una vez que se ha formado la matriz $matrizatributos se lleva a cabo una verificación de los datos que almacena.

Parte de la información que contiene la matriz debe ser contrastada con la información que hay en la base de datos del servidor. De toda la información que se tiene en la matriz sobre un recurso, los campos que deben ser verificados son:

- Valor: contiene el nombre de un recurso del objeto. Puede tener dos formatos dependiendo de la propiedad del recurso:
  - Si se trata de un recurso propio debe contener el nombre del fichero que el usuario ha indicado. Se comprueba que este fichero ha llegado al servidor. Los ficheros que sean recursos propios no deben contener el carácter "/". Cuando el recurso es propio se comprueba que el nombre del fichero subido no contenga

caracteres especiales tales como vocales con acentos o ñ.

- Si se trata de un recurso que el usuario ha indicado como no propio, se comprueba que sigue el formato X/nombre o X/, donde "X" es el identificador de un objeto que ya debe existir en la base de datos y "nombre" es el nombre de un fichero que esté como recurso propio en el objeto cuyo identificador es X. Se puede agregar como recurso no propio un recurso individual de otro objeto (X/nombre) o se puede añadir un objeto entero (X/).

- Dependencias: La validez de las dependencias que se declaran en un recurso también deben ser confirmada. Un recurso no propio no puede tener ningún tipo de dependencias, por lo que se indicará como un fallo que se debe solucionar.
-

Los recursos propios son los únicos que pueden indicar esta información, las dependencias siguen el mismo formato que se utiliza para declarar recursos no propios (X/nombre o X/) por lo que la comprobación a realizar es la misma. Además existe la posibilidad de que un recurso propio del objeto que se está creando dependa de otro recurso propio de este mismo objeto, en este caso aparecerá como "/nombre" y se debe recorrer la matriz $matrizatributos para comprobar que ya existe como recurso propio.

Cuando se produzca algún fallo en los datos aportados en los recursos no se mostrará el botón "Aceptar", las únicas opciones válidas serán las de "Editar" para modificar los datos que se hayan indicado como incorrectos y "Cancelar" para rechazar los cambios que se han producido en la página de recursos.

8.7.1.3.FICHERO: F1PIEZASCAJAS

Este fichero es el encargado de permitir que el usuario pueda modificar, añadir y eliminar atributos pertenecientes al grupo de Campos Generales.

Muestra los atributos (nombre y valor) que hay por defecto en este grupo, en caso de que el usuario hubiese añadido anteriormente más atributos también aparecerán.

Esta información procede de los valores que se obtienen a través del array POST y que se tienen su origen en la matriz creada en el fichero f1piezas.php.

Existen varias opciones:

- Añadir nuevos atributos a los que ya existen, mediante el botón "Añadir". Se agrega a la lista un nuevo atributo con el nombre y valor indicado en el formulario correspondiente.
- Cancelar: Se vuelve a la página piezas.php perdiendo todas las modificaciones hechas desde la última vez que se guardó.
- Aceptar: Antes de finalizar el proceso de modificación de atributos se han de aceptar todos los cambios realizados hasta el momento ya que algunos datos pueden no haberse actualizado en la matriz $matrizatributos.
- Finalizar: Indica que se ha terminado el proceso de modificación de atributos y se devuelve el control a la página f1piezas.php.
- Además de las opciones anteriores existe la posibilidad de eliminar atributos (sólo aquellos que el usuario añadió, nunca los de que se muestran por defecto).
- Es posible cambiar incluso el nombre de la ficha, aunque no se recomienda hacerlo (no siempre se actualizan bien estos nombres).

El resto de los ficheros del directorio del tipo fxpiezascajas.php tienen el mismo comportamiento que el comentado para f1piezascajas.php, sólo varía el número de características por defecto.

A continuación se indican los atributos por defecto que existen en cada uno de los ficheros que existen en las carpetas piezas, fotos y documentos, indicando entre paréntesis el nombre por defecto que agrupa a estos atributos:

F1piezas.php (Campos Generales)

- Nombre
- Modelo
- Fabricante
- Ubicación
- Visible (sí/no)
- Conservación
- Descripción

F2piezas.php (Características Físicas)

- Ancho
- Largo
- Alto
- Peso

F3piezas.php (Características tecnológicas)

- Frecuencia de reloj
- Tiempo de acceso
- Capacidad

F4piezas.php (Recursos)

- Nombre
- Ruta
- Tipo
- Descripción
- Dependencias

F5piezas.php (Catalogación)

- Catalogación
- Inventario MIGS
- Inventario UCM
- Base Museística

F6piezas.php (Ciclo de vida)

- Entidad/Persona
- Evento
- Fecha

F7piezas.php (Clasificación)

- Clasificación
- Sección
- Categoría

F1fotos.php (Campos Generales)

- Nombre
- Categoría
- Ubicación
- Descripción

F2fotos.php (Características físicas)

- Ancho
- Alto
- Largo
- Resolución

F3fotos.php (Recursos)

Presenta los mismos campos que f4piezas.

F4fotos.php (Catalogación)

- Catalogación
- Inventario MIGS

F5fotos.php (Ciclo de vida)

Contiene los mismos metadatos que f6piezas.

F6fotos.php (Clasificación)

- Contenido

F1docu.php (Campos Generales)

- Nombre
- Título
- Autores
- Publicación
- Tipo de material
- Ubicación
- Descripción

F2docu.php (Características físicas)

- Ancho
- Alto
- Largo
- Número de páginas

F3docu.php (Recursos)

Presenta los mismos campos que f4piezas.

F4docu.php (Catalogación)

- Inventario MIGS
- CDU
- ISBN/SIN
- Signatura UCM

F5docu.php (Ciclo de vida)

Contiene los mismos metadatos que f6piezas.

F6docu.php (Clasificación)

- Sección

8.7.1.4. DIRECTORIO: CONSTRUIRMANI

Este directorio es el encargado de crear el paquete zip según IMS que podrá ser subido al servidor.

El contenido de este directorio es:

- ConstruirObjVirtual.php
- CrearOVAtri.php

- CrearOVRec.php
- RellenarManifiesto
- Otros ficheros

8.7.1.4.1.Fichero: construirObjVirtual

La estructura de este fichero es:

```
construirObjVirtual.php
|
|---include-> crearOVAtri.php
|
|---include->crearOVRec.php
```

A través de los ficheros crearOVAtri y crearOVRec se crean dos matrices $atributos y $recursos que servirán para construir construir el fichero objetoVirtual.xml.

Una vez creadas las matrices indicadas uTiliza DOM para rellenar el xml siguiendo la dtd. De cada una de las matrices que se han creado no se almacena el primer componente porque sólo contiene el nombre de la ficha, este dato ya está almacenado en el resto de los atributos dentro del campo "categoría".

Una vez creado el fichero se almacena junto con una copia de su dtd en el directorio creado en la sesión para almacenar la información del objeto.

Una vez creado el xml con los datos el objeto (recursos y atributos) se debe construir el manifiesto, por lo que se pasa el control mediante un formulario al fichero rellenarManifiesto.php. A través de este formulario se pasan por el array POST todos los metadatos que el usuario hay indicado en el proceso de creación del objeto.

8.7.1.4.2.Fichero: CrearOVAtri

Este fichero crea una serie de matrices a partir de la información almacenada en los ficheros txt que se han ido generando. Crea matriz $atributos con todos los datos de los ficheros que no almacenan recursos ni metadatos.

Por cada atributo se añade una fila a la matriz indicando el nombre del atributo, el valor, la unidad de medida en caso de tenerla, la categoría (nombre de la ficha en la que va a aparecer el atributo) y la posición de la ficha (permite asociar los atributos de todas las fichas que tienen el mismo tipo de atributos aunque el nombre de la categoría haya cambiado). Los datos de la matriz se

codifican en UTF8 para que los caracteres especiales (acentos, ñ, ...) sean interpreten de manera correcta cuando se almacenen en el fichero objetoVirtual.xml.
Además de la matriz para los atributos, se crean otras tres matrices, $metadatos_catal, $metadatos_lifecycle y $metadatos_clas, cada una de ellas almacena uno de los tres tipos de metadatos que puede tener un objeto. Esta información se obtiene de los ficheros txt que existan en cada caso. Estas matrices tienen los mismos campos que la matriz $atributos, aunque el campo "unidad" siempre está vacío.

Para cada una de estas matrices se indica el número de atributos que almacena para su posterior tratamiento.

8.7.1.4.3.Fichero: CrearOVRec

Crea el directorio en el que se van a almacenar todos los ficheros, posteriormente se comprimirá para formar el paquete IMS. El nombre de este directorio es el que el usuario indicó en la página piezas.php, en caso de existir en el directorio de sesión otra carpeta con el mismo nombre se indica al usuario que debe elegir otro nombre para el paquete (error.php).

Este fichero lee el txt donde se almacenan los recursos indicados por el usuario en caso de que exista. Con esta información se crea una matriz con los datos de los recursos de forma similar a lo que se hace con los atributos o los metadatos.

La matriz $recursos tiene los mismos campos que el txt donde se almacenan los recursos (piezas4.txt):

- Atributo
- Tipo
- Num_depen
- Ruta
- Nombre
- Dependencia
- Descripción
- TipoRec

Estos campos se almacenan con codificación UTF8 para almacenarlos en el fichero objetoVirtual.xml. Según se va creando la matriz se copian los ficheros de los recursos que son propios al directorio que se ha creado.

El nombre de la ficha que se almacena en el fichero piezas4nomficha.txt actualmente no se usa, en el museo se visualiza siempre el nombre por defecto “recursos”.

8.7.1.4.4.Fichero: RellenarManifiesto

Este fichero permite crear el manifiesto en el que se declara el contenido del zip y los metadatos del objeto en caso de tenerlos.

Se deben declarar en este fichero todos los ficheros que forman el paquete IMS, por lo tanto debe contener, entre otros archivos, los recursos que el usuario ha indicado como propios. Esta información se consigue a partir del fichero objetovirtual.xml que en este momento del proceso ya está creado.

De este fichero interesa el nombre y la ruta de todos los recursos que aparezcan en él. El fichero se parsea mediante la clase class_path_parser.php y los datos se almacenan en el fichero auxiliar.txt.

Una vez que se tiene la información se usa DOM para construir el manifiesto.

Antes de construir el xml se reconstruyen las matrices de los metadatos con los datos que llegan a través del array POST, de esta forma se facilita posteriormente el tratamiento de estos datos. La información se extrae fácilmente ya que cada metadato aparece identificado por un nombre único caracterizado por el tipo de metadato (catalogación, clasificación, ciclo de vida) y el número de orden para diferenciarlo del resto de metadatos del mismo tipo que puedan existir.

Con estas matrices lo primero que se introduce en el xml es la información de los metadatos. Al igual que pasa con los atributos, no se almacena el primer componente de cada matriz porque no contiene metadatos, sólo el nombre de la ficha, al igual que ocurre con los recursos, este nombre tampoco se utiliza en la visualización de los datos en el museo.

Se añade al manifiesto cada una de las rutas de los metadatos, no existe una forma general para almacenar esta información por lo que si se quiere añadir un nuevo metadato se ha de hacer de forma expresa el tratamiento del nuevo atributo.

Algunos de los metadatos poseen atributos obligatorios, normalmente hacen referencia al idioma utilizado o a identificadores en el caso de los recursos, estos datos no se consultan al usuario por lo que se ponen unos valores por defecto adecuados a cada uno de los atributos.

Una vez almacenados los metadatos toca el turno de los recursos.

Se crea una matriz para almacenar los datos que se leen del fichero auxiliar.txt y se rellenan los distintos campos de los recursos. El fichero auxiliar.txt contiene todos los recursos del objeto, tanto propios como no propios, sólo se deben almacenar aquellos recursos cuya ruta sea "/". Además de los recursos propios del objeto se deben declarar el resto de ficheros que van a ir comprimidos en el zip, por lo que se debe añadir al manifiesto el fichero objetoVirtual.xml y su dtd.

Una vez creado el manifiesto se almacena en el directorio del objeto, donde también se copian los esquemas que necesita el manifiesto.

Con toda la información en el directorio el último paso es comprimir el directorio y mostrar al usuario la pantalla que indica que el proceso ha finalizado y ya puede realizarse la descarga del zip para subirlo posteriormente al servidor.

8.7.1.4.5.Otros ficheros

Existen otros ficheros en el directorio para fines diversos.

El fichero error.php muestra un mensaje de error en caso de que ya exista un directorio con el mismo nombre que el indicado por el usuario para el zip.

Existen plantillas para construir los xml del manifiesto y del objeto virtual.

El fichero class_path_parser.php tiene la misma función que la explicada en el directorio privado/servidor.

8.7.2.DIRECTORIO: IMAGES

Contiene las imágenes utilizadas para la configuración de las pantallas que se van a visualizar durante el proceso de creación del objeto virtual.

8.7.3.FICHERO: PRINCIPAL

Es la página que indica el comienzo del proceso de creación de un objeto. En caso de que durante la sesión actual ya se hubiese creado un objeto, se encarga de eliminar todos aquellos ficheros que se generan durante el proceso de construcción de un objeto.

Se encarga de mostrar los tipos de objetos que se pueden crear:

- Piezas
- Documentos
- Fotos

La pulsación de cada uno de estos enlaces permite iniciar el proceso de construcción de un objeto haciendo desaparecer la ventana actual y abriendo una específica para el tipo de objeto que se desea crear.

8.7.4.FICHERO: COMPRIMIR_AUX

Este fichero es el encargado de la creación del zip una vez que se ha construido en el servidor el objeto virtual.

Recibe el nombre del directorio que se debe comprimir, el nombre del zip que se crea tiene el mismo nombre que el directorio.

Para comprimir se recorre todo el directorio y se comprime cada uno de los ficheros que contiene.

Cuando el zip está creado aparece con el nombre que el usuario le ha dado, cuando se descomprime aparece un nuevo directorio llamado “objeto” que es el que realmente contiene el objeto. Esto facilita el tratamiento del zip cuando se sube al servidor.

La instrucción $zipfile -> add_file($data,"objeto/".$fich) es la que se encarga de que el zip contenga ese nuevo directorio.

8.7.5.OTROS FICHEROS

El directorio formularioOV contiene una copia de los ficheros que deben incluirse en el zip que empaqueta el objeto creado para que cumpla con los estándares de IMS. Estos ficheros son :

- Imscp_v1p1.xsd e imsmd_v1p2p2.xsd: son los esquemas utilizados por el archivo imsmanifest.xml
- Objetovirtual.dtd: es la dtd que debe seguir el fichero objetoVirtual.xml que va a formar parte del paquete.

8.8.DIRECTORIO : MODIFICAROV

Contiene todas las páginas que se utilizan durante el proceso de modificación de un objeto de la base de datos. Tiene la misma estructura de carpetas que el directorio formularioOV y su funcionamiento es similar.

8.8.1.FICHERO: PRINCIPAL

Es la página inicial en el procese de modificación, informa al usuario del identificador y el tipo del objeto que ha seleccionado para modificar.

En caso de haber realizado anteriormente la creación de un objeto, borra todos los ficheros producidos durante este proceso.

En función del tipo del objeto que se va a modificar, el enlace de esta página irá dirigido a alguna de las tres carpetas existentes: piezas, documentos y fotos.

El funcionamiento de cada uno de los directorios que están contenidos en modificarOV es el mismo , por lo que sólo se explica uno de ellos, el directorio piezas.

8.8.2.DIRECTORIO: PIEZAS.

Esta página da acceso a cada uno de los grupos de atributos que puede tener un objeto. Cuando desde cualquiera de esas otras páginas se regresa a ésta, se actualiza la base de datos con la información que contiene el array POST procedente de la última página visitada. Al contrario de lo que sucede en la página del mismo nombre de la carpeta formularioOV, en este caso los datos no se van almacenando en ficheros auxiliares de texto sino que directamente se modifica la base de datos, perdiendo toda la información que hubiera anteriormente.

El resto de los ficheros del directorio (fxpiezas.php y fxpiezascajas.php) tienen el mismo funcionamiento visto para el directorio formularioOV. La única diferencia que existe entre ellos está en el origen de la información que se muestra.

Cada vez que se visitan las páginas fxpiezas.php se muestra la información que existe en la base de datos para esa ficha en particular. Esta información será la que se pueda modificar en las páginas fxpiezascajas.php de la misma forma que para la creación de un objeto.

En el caso de la ficha de los recursos, cada vez que se entra en la página f4piezas.php, se hace una copia de todos los recursos propios del objeto en la carpeta de sesión del usuario. Será esta copia la que se modifique directamente con los cambios que realice el usuario. Al igual que ocurre con el resto de las fichas, la información y la sustitución de los recursos antiguos por los nuevos

recursos propios no se actualizarán hasta que se llegue de nuevo a la página piezas.php previa aceptación de los datos en la página correspondiente.

Sólo se podrá cambiar o eliminar un recurso cuando ningún objeto (incluido él mismo) tenga entre sus recursos o dependencias a ese recurso. Cuando otros objetos referencian al objeto entero se permitirán todas las modificaciones posibles.

La forma en que se actualiza la base de datos difiere según el tipo de tabla que se vaya a modificar.

A continuación se indica cómo se llevan a cabo las modificaciones:

- Las tres primeras fichas afectan a las tablas "atributos_texto" y "atributos_numericos" que no poseen ningún campo que se incremente automáticamente. Para modificar los datos que afectan a estas tablas se borran todas las filas que contienen en el campo posición el número que cada ficha tiene asignado. Una vez borrada toda la información, se almacenan los nuevos datos en las tablas correspondientes. El proceso es el mismo tanto si se modifica un atributo como si se modifican todos pero se simplifica el proceso de actualización de estas tablas.

- La ficha 4 (recursos) afecta a las tablas "recursos" y "dependencias" de la base de datos, ambas tablas contienen campos *autoincrement* por lo que en la actualización de las filas no se actúa como en el caso anterior. Se intenta reducir al máximo el número de filas que se añaden para evitar que el valor de este campo crezca demasiado.

La actualización de la información se lleva a cabo de la siguiente manera:

1. Si el nuevo número de atributos es menor que el número de filas de la base de datos se eliminan las filas que sobran y las que quedan se modifican con los datos de los nuevos recursos.
2. Si el número de recursos es mayor que el que existe en la base de datos se modifican las filas que ya existen y se añaden nuevas filas para los recursos que falten por guardar.
3. Si el número de filas y recursos coincide simplemente se modifican las filas.
4. Cada fila puede estar siendo referenciada desde la tabla "dependencias" por lo que cada vez que se modifica una fila de la tabla "recursos" deben tratarse estas referencias. El tratamiento de estas filas en la tabla "dependencias" se lleva a cabo de la misma forma que las filas de la tabla "recursos".

Cada vez que se almacena un recurso propio se copia a la carpeta recursos/OVX el fichero correspondiente.

- Las fichas de metadatos (fichas 5, 6 y 7) afectan a las tablas ”metadatos” y ”atributos_metadatos”. La tabla “metadatos” tiene un campo *autoincrement* por lo que el tratamiento de las filas se lleva a cabo igual que para los recursos.

Cuando se modifican metadatos la variación del campo “num_ruta” debe hacerse de forma manual teniendo en cuenta a todos los metadatos que se encuentran declarados en ese objeto.
Cuando se descarga un objeto de la base de datos, el campo “num_ruta” permite reconstruir los metadatos del manifiesto siguiendo el mismo orden que tenían cuando se creó el objeto. Este campo indica el orden en el que aparecen los metadatos (catalog, lifecycle y classification) y para cada uno de ellos permite saber que campos (taxon, source, ...) forman parte del mismo metadato, cual es la jerarquía que existe entre ellos, etc.

De toda la secuencia de dígitos de este campo, el 4º indica el orden en el que aparecen los tres tipos de metadatos, si sólo existen atributos de tipo Classification este dígito tendrá el valor 1, en cambio, si hay metadatos de los tres tipos este dígito valdrá 1 para los de tipo Catalog, 2 para los de tipo Lifecycle y 3 para Classification.

Este tipo de cambios son los que se deben realizar cuando se añaden metadatos, además de los cambios en otros dígitos dependiendo de los hijos de cada metadato.

Cuando se añaden metadatos de un tipo determinado por primera vez a la base de datos se han de tener en cuenta:

1. El cuarto dígito del campo “num_ruta” dependerá de qué tipo de metadato se añada y de qué tipos de metadatos existan ya en la base de datos:
   Cuando existen los tres tipos de metadatos el valor del dígito es:
   Catalog →1
   Lifecycle →2
   Classification→3

2. Si se añaden metadatos de tipo Classification por primera vez el restod e metatados que ya existen no se ven modificados.

3. Si se añaden metadatos de tipo Lifecycle por primera vez y ya existen metadatos de tipo Classification habrá que actualizar el 4º dígito (incrementándolo en 1) de los metadatos de tipo Classification.

4. Si se añaden metadatos de tipo Catalog por primera vez y existen otros metadatos habrá que actualizar el 4º dígito (incrementándolo en 1) de los otros metadatos.

A continuación se muestran los esquemas que reflejan la comunicación existente entre los distintos ficheros que forman los directorios formularioOV y modificarOV.

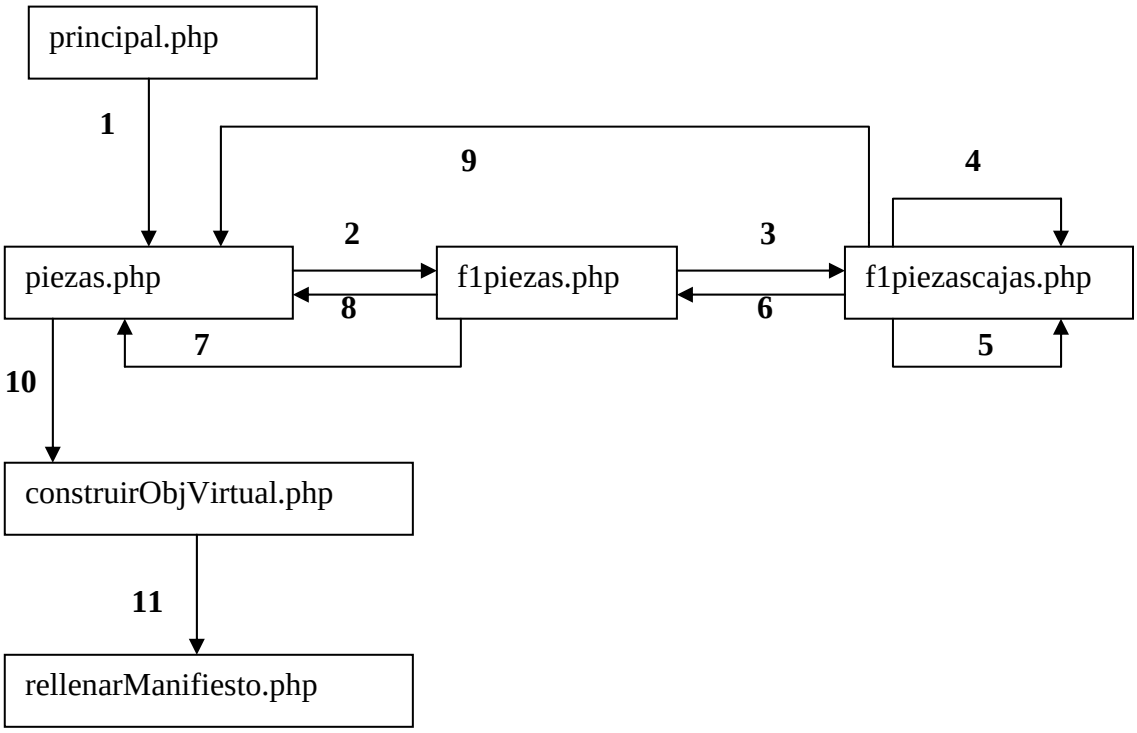
Los esquemas están simplificados, se muestra el primer grupo de características de los objetos de tipo pieza. Para el resto de objetos y grupos de atributos el esquema es el mismo.

Leyenda:

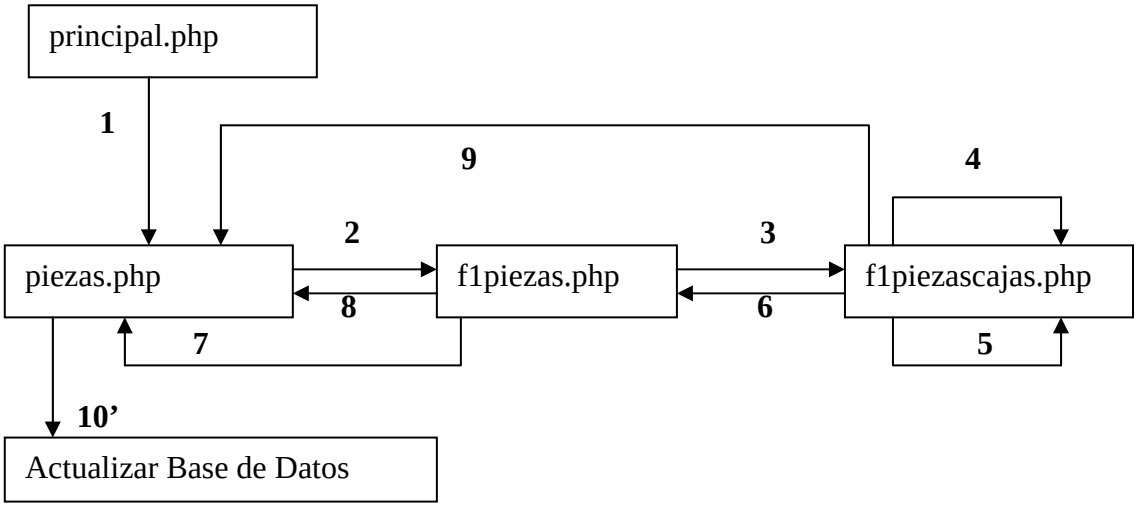
1	**Elegir tipo de objeto**
2	Pulsar botón
3	Editar
4	Añadir
5	Aceptar
6	Finalizar
7	Aceptar
8	Finalizar
9	Cancelar
10	Crear objeto
10’	Aceptar
11	Crear Manifiesto

La relación entre los distintos ficheros de la carpeta formularioOV y modificarOV se muestra a continuación junto con los eventos que producen las transiciones.

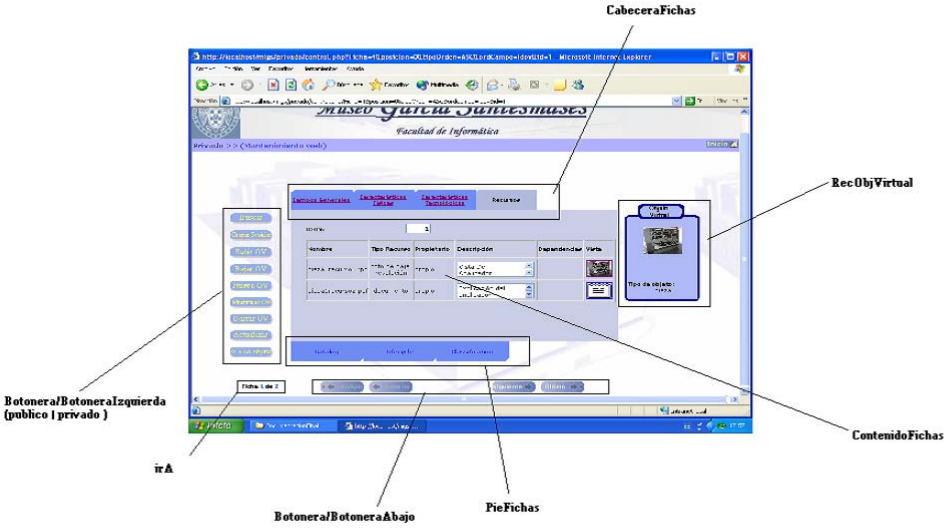
Formulario OV



Modificar OV



9.DIRECTORIO : FICHAS



Descripción de una ficha de un objeto

Es el directorio encargado de la presentación de los objetos.

Los ficheros PlantillaFichasPrivado y PlantillaFichasPublico indican el orden en el que se incluyen el resto de ficheros para formar la ficha de presentación de los objetos.

Cada uno de ellos contiene un formulario que va a permitir la transmisión de los datos a lo largo de la aplicación , la acción de este formulario recae sobre el fichero privado/control.php o publico/control.php respectivamente.

La estructura de estos ficheros es la siguiente:

PlantillaFichas[Privado/Publico].php
|
|---include-> cabecera/cabecera.php
|
|---include-> ../fichas/botonera/botoneraIzquierda[Privado/Publico].php
|
|---include-> ../fichas/cabeceraFichas/cabeceraFichas.php
|
|---include-> ../fichas/contenidoFichas/Ficha.php
|
|---include-> ../fichas/pieFichas/pieFichas.php
|
|---include-> ../fichas/recObjVirtual/recObjVirtual.php
|

|---include-> ../fichas/irA/numFicha.php
|
|---include-> ../fichas/botonera/botoneraAbajo.php

9.1.DIRECTORIO:BOTONERA

Contiene los ficheros que muestran las botoneras que aparecen a la izquierda y en la parte inferior de las fichas.

El fichero "botoneraAbajo.php" permite la visita de los objetos de uno en uno. En función del valor de la variable $posición y $numFilas es posible saber si se está en el último objeto, en cuyo caso se deshabilitan los botones "siguiente" y "último", en el primero, en ese caso se deshabilitan los botones "primero" y "anterior", o se está en un objeto intermedio, momento en el que se habilitan todos los botones.

El siguiente, el anterior y el último dependen del criterio de ordenación indicado. Los botones contienen etiquetas href en cuyas referencias se indican los valores de las variables $posición, $ordCampo y $tipoOrd y $Ficha. Estas variables no se pasan mediante método POST (formularios) sino mediante GET.

El fichero botoneraIzquierdaPrivado.php y botoneraIzquierdaPublico.php muestran las opciones que tiene cada usuario en función de la sección en la que se encuentre (público, privado). Las opciones buscar e ir a un determinado objeto son comunes a ambas botoneras.

La botonera de público además permite cerrar la ventana actual mediante un botón..

La botonera de privado ofrece la posibilidad de crear, modificar, borrar, subir, bajar un objeto, actualizar y cerrar la sesión. Este fichero es el encargado de abrir las nuevas ventanas en las que poder realizar las acciones de modificar y crear objetos.

En el caso de la acción de modificación de un objeto existe una secuencia de acciones que deben realizarse en un determinado orden y que involucra a varios ficheros por lo que tiene un tratamiento especial al resto de opciones.
Al pulsar el botón "modificarOV" las acciones que se deben realizar son:

1. Actualizar las variables de sesión "idovAModificar" y "tipoOVAModificar". Esto se realiza en el fichero privado/control.php cuando la opción elegida es la de modificar el objeto actual.
2. Abrir una nueva ventana en la que se muestra el identificador y el tipo del objeto que se desea modificar.

Estas dos acciones deben realizarse en este orden, para asegurarlo la apertura de la nueva ventana se lleva a cabo aplicando el retardo suficiente para que las variables de sesión sean actualizadas antes de que se abra la ventana.

9.2.DIRECTORIO:CABECERAFICHAS

Permite mostrar las pestañas que aparecen en la parte superior de las fichas, en ellas se indica el tipo de atributos que contiene la pestaña, estos atributos pueden ser atributos de texto y numéricos, o la existencia de recursos.

Se muestra una pestaña por cada valor diferente del campo posición que tenga ese objeto en las tablas 'atributos_texto' y 'atributos_numericos', el nombre de la pestaña es el contenido del campo categoría de esas tablas.

Se consulta la tabla 'recursos' para ver si el objeto tiene recursos (propios o no), en caso de tenerlos se añade la pestaña correspondiente.

En función del valor de la variable $Ficha las pestañas cambian de color para indicar cual es la que se está visualizando en cada momento.

En las referencias existentes en las pestañas, la variable $Ficha toma el valor del campo posición que engloba a todos los atributos de la pestaña pulsada. En caso de pulsar en la pestaña de recursos, el valor de la variable es uno más que el número de valores diferentes que posee el objeto en las tablas 'atributos_texto' y 'atributos_numericos', es decir el número de categorías de atributos que posee el objeto.

9.3.DIRECTORIO:PIEFICHAS

Tiene la misma funcionalidad que el fichero cabeceraFichas.php. En este caso se encarga de mostrar en la parte inferior de la ficha las pestañas correspondientes a los metadatos.

El valor que toma la variable $Ficha tiene en cuenta la existencia de atributos y recursos del objeto. Se averigua cual es el valor más alto puesto para la variable $Ficha en el archivo que construye la cabecera ("cabeceraFichas/cabeceraFichas.php") y a partir de ese número se empieza a dar valor a la variable $Ficha. Es una forma de indicar un orden entre las distintas pestañas y saber si se trata de pestañas de recursos, metadatos y atributos.

9.4.DIRECTORIO: IRA

Contiene el código que permite mostrar en qué posición de la lista de objetos, respecto al número total de objetos que hay en la base de datos, se encuentra el objeto que se está visualizando actualmente.

Esta indicación se muestra a la izquierda de la botonera inferior.

9.5.DIRECTORIO: RECOBJVIRTUAL

Muestra un resumen del objeto que se está mostrando.

Aparece ocupando la parte derecha de la pantalla.

El fichero "recObjVirtual.php" muestra el tipo del objeto (documento, pieza o foto) y una imagen asociada al mismo. Esta imagen será uno de los

recursos del objeto en caso de que posea imágenes, si no posee imágenes o no tiene recursos aparecerá una imagen por defecto.

Cuando se pulse esta imagen se abrirá una nueva ventana en la que mostrará un resumen del objeto, de ello se encarga el fichero "muestraResumen.php". La información que se muestra es la descripción del objeto y los atributos de texto y numéricos que tengan contenido.

9.6.DIRECTORIO: CONTENIDOFICHAS

En este directorio se encuentran los ficheros que se encargan de mostrar los datos de las distintas pestañas.

En todas las fichas se muestra el identificador del objeto que se esta visualizando.

El fichero "Ficha.php" muestra los datos de los metadatos y de los atributos de texto y numérico.

Mediante el valor de la variable $Ficha es posible saber si la pestaña pulsada es de atributos, recursos o metadatos, y dentro de los metadatos y los atributos permite diferenciar entre los distintos tipos.

Sólo se muestran los datos de los campos que tienen el valor relleno, no se muestran campos vacíos.

Si el valor de $Ficha es 1 se muestra además la descripción en caso de estar rellena.

El fichero "FichaRecursos.php" se encarga de mostrar la información de los recursos del objeto, ya sea propio o no. Para cada objeto se muestra el nombre del fichero, su extensión, el tipo del recurso, el propietario, las dependencias y una pequeña vista que permita ver el contenido del recurso en caso de ser una imagen o una imagen por defecto en caso de que sea un recurso no gráfico (pdf, doc, html,...).

La vista que acompaña a cada recurso (sea imagen o no) tiene una referencia al recurso original. Cuando se pulsa la imagen se abre una ventana nueva en la que se muestra el recurso.

Los ficheros encargados de mostrar los recursos son "muestraImagen.php" y "muestraRecurso.php".

El fichero "muestraImagen.php" se encarga de mostrar aquellos recursos que son imágenes, en la nueva pantalla se muestra el nombre del fichero, el tipo, el propietario y la descripción del mismo. Esta ventana muestra la imagen con posibilidad de ampliar y reducir el tamaño de la misma para poder visualizarla mejor.

El fichero "muestraRecurso.php" se encarga de mostrar aquellos recursos que no son imágenes, en la nueva ventana se muestra el nombre del fichero, el tipo y la descripción.

En el caso de mostrar un recurso que no es una imagen el fichero "FichaRecursos.php" abre una segunda ventana en la que poder visualizar el documento.

10.OTROS DIRECTORIOS

En la aplicación existen otros directorios:

- Images: contiene la imagen utilizada en el fondo de la aplicación así como alguna imagen más usada en los ficheros de error de la sección privada y pública. El resto de las imágenes están almacenadas en los directorios donde se encuentran los ficheros que hacen uso de ellas.
- Css Contiene todos los estilos utilizados en la aplicación para mostrar la información. Hay definidos más estilos de los que se usan actualmente.
- Js : contiene las funciones javascript encargadas de la carga de imágenes e iconos y la rotación de imágenes que aparece en la página principal/ principal.html.

6. ESTADO ACTUAL DE LA APLICACIÓN Y TRABAJO FUTURO

El museo distingue entre dos tipos de usuarios uno con control absoluto sobre el museo que sería el administrador (crear, subir, modificar, bajar y eliminar), y un visitante que solo puede navegar en el museo y realizar búsquedas de objetos a partir de una plantilla donde rellena las características del objeto que desea encontrar.
El acceso a los objetos se puede realizar en base a su número identificador, o de una forma secuencial desde el primer objeto, o a través de las dependencias y relaciones entre objetos.

El trabajo futuro se debe centrar en las siguientes tareas:

- *Mejorar el interfaz gráfico del museo.*
  Se debe mejorar el interfaz visible del museo, dotándole de un aspecto visual más agradable.

- *Ampliar y desarrollar la navegación en el museo.*
  La navegación por el museo es muy sencilla se limita a la vista de las fichas de los objetos y en un trabajo futuro podría enriquecerse añadiendo vistas más complejas de los objetos del museo.

- *Desarrollar búsquedas clasificadas de objetos en el museo.*
  Actualmente se realizan búsquedas rellenando una plantilla para encontrar objetos en el museo que cumplan unas determinadas características. Una mejora muy importante sería aprovechar toda la información de metadatos que se encuentra en la base de datos para realizar búsquedas más sofisticadas, por ejemplo búsquedas por clasificación de los objetos.

- *Crear una jerarquía de usuarios, distinguiendo entre administrador, investigadores y visitantes.*
  Es necesario desarrollar una gestión de usuarios en la aplicación. Actualmente existe una clasificación muy sencilla de los usuarios de la aplicación: un visitante que sólo puede visitar y navegar por el museo y un investigador que tiene la responsabilidad de crear objetos, subirlos, modificarlos, etc. Una posible mejora sería que los objetos del museo tuvieran un propietario de forma que sólo pudieran ser modificados o eliminados por su propietario. Debería un usuario investigador especial que sería el administrador del sistema con todos los privilegios, y varios investigadores con capacidad de crear sus propios objetos, bajar objetos del museo, modificar sus objetos, borrar, etc.

7. BIBLIOGRAFÍA

1. IEEE/LTSC: http://ltsc.ieee.org/
2. IMS: www.imsglobal.org
3. ADL: www.adlnet.org
4. A. Fernández-Valmayor, M. Guinea, M. Jiménez, A. Navarro, A. Sarasa
Virtual objects an approach to building learning objects in archaelogy, in. M. Llamas-Nistal et al. (Eds) Computers and Education. Towards a Long-Life Learning Society 2003, pp.191- 202. Kluwer Academic Publisher. Dordrecht, Neederlans
5. LOM: http://ltsc.ieee.org/wg12/
6. L. Mortimer: http://www.learningcircuits.org/2002/apr2002/mortimer.html
7. Dublin Core: http://dublincore.org/
8. ARIADNE: http://www.ariadne.ac.uk/
9. Bourret, Ronald. "XML and Databases." September, 1999-2003 http://www.rpbourret.com/xml/XMLAndDatabases.htm

8. MANUAL DE USUARIO

El siguiente documento explica el funcionamiento del Museo Virtual de Informática García Santesmases.

Para acceder al museo se debe teclear en la barra de direcciones del navegador: http://gala2/migs/museogs

De esta forma se llega a la página principal del museo virtual:

Figura 1. Página principal del Museo García Santesmases

Existen tres enlaces principales:

- Enlace Presentación: Se accede a una página donde se encuentra una introducción que explica brevemente las características del museo real.

Figura 2. Página de presentación

Los otros dos enlaces sirven para acceder al museo a los usuarios. Existen dos tipos de usuarios en la aplicación: usuario visitante y usuario investigador:

1.USUARIO VISITANTE

Un usuario visitante puede navegar por todos los objetos del museo o puede realizar una búsqueda de objetos específicos.

1.1 Navegación por el museo.

Pulsando sobre el enlace Visitantes se accede directamente a la página que permite navegar entre los objetos virtuales del museo. Esta página consiste en un conjunto de fichas rellenas con los datos del objeto como puede observarse en la figura3. Todos los objetos no tienen el mismo número de fichas porque sólo aparecen aquellas fichas en las que existe algún dato.

Existen dos tipos de fichas:

- Fichas de datos: contienen datos característicos del propio objeto, entre estas fichas se encuentran **"Campos generales"**, **"Características físicas"**, **"Características tecnológicas"** y **"Recursos"**.

- Fichas de metadatos: contienen información útil para la administración y la gestión del propio objeto en el museo, esta información está relacionada por ejemplo con la clasificación del objeto. Estas fichas son **"Clasificación"**, **"Catalogación"**, **"Ciclo de vida".**

Figura 3. Vista de un objeto virtual para un usuario visitante

Para acceder a las fichas se pulsa sobre las pestañas de cada una. En la figura 3 se muestra la ficha Campos generales del objeto primero de la base de datos. Pulsando sobre la imagen de la derecha aparece una ventana que permite ver todos los datos del objeto, es decir muestra la información de todas las fichas de datos (no las de metadatos):

Figura 4. Detalle de la información de los datos de un objeto virtual

Una de las fichas más interesantes es la ficha de recursos, la cual permite acceder a diferentes documentos u objetos del museo que están relacionados con el objeto actual. De esta forma se puede realizar una navegación entre distintos objetos que tienen algún tipo de vinculación.

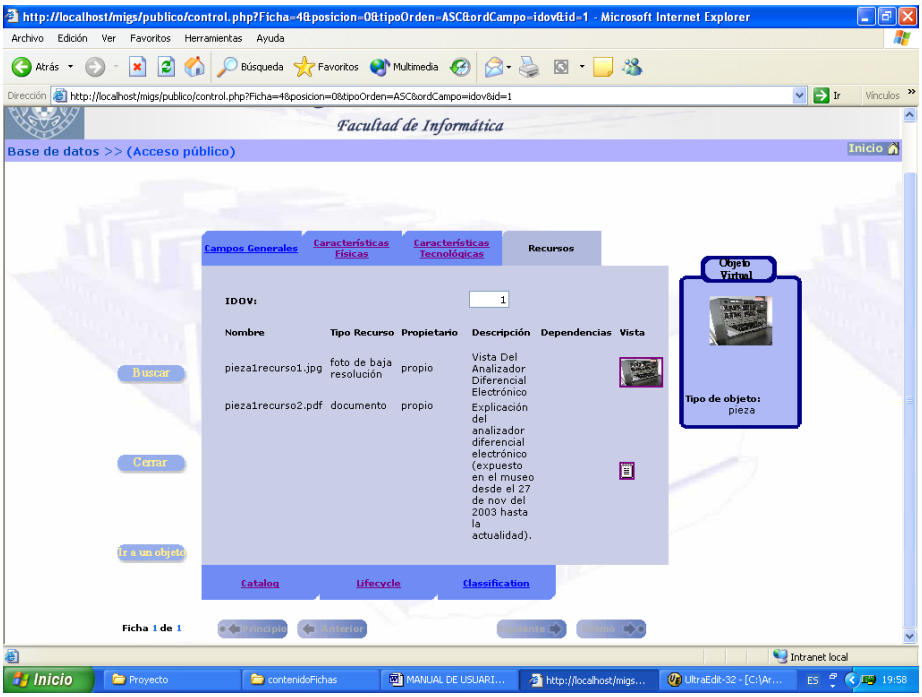


Figura 5. ficha de recursos de un objeto virtual

En la figura 5 se muestra la ficha de recursos de un objeto virtual, se observa que el objeto actual tiene dos recursos.

- Pulsando sobre la imagen del primero aparece una nueva ventana que permite ver la imagen más detallada:

Figura 6. vista de un recurso detallada.

1.2 Buscar objetos

Además de la navegación por el museo objeto por objeto, un usuario también tiene posibilidad de buscar objetos con unas características específicas. Pulsando sobre el primer botón de la botonera izquierda, "Buscar", se llega a la página de la figura 7:

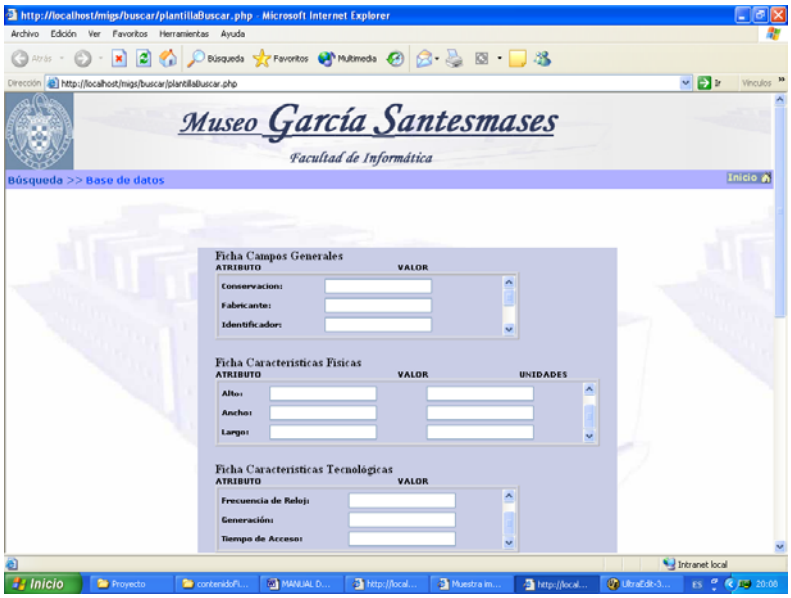


Figura 7. Página para buscar un objeto en el museo

Se rellenan todos aquellos campos que el usuario desee especificar, y pulsando sobre el botón situado en la parte izquierda de la página Buscador, se muestran todos aquellos objetos que tengan los datos fijados por el usuario. Por ejemplo se puede pedir que se busquen todos los objetos que pertenezcan al catálogo de piezas, o que pertenezcan a la generación primera, etc.

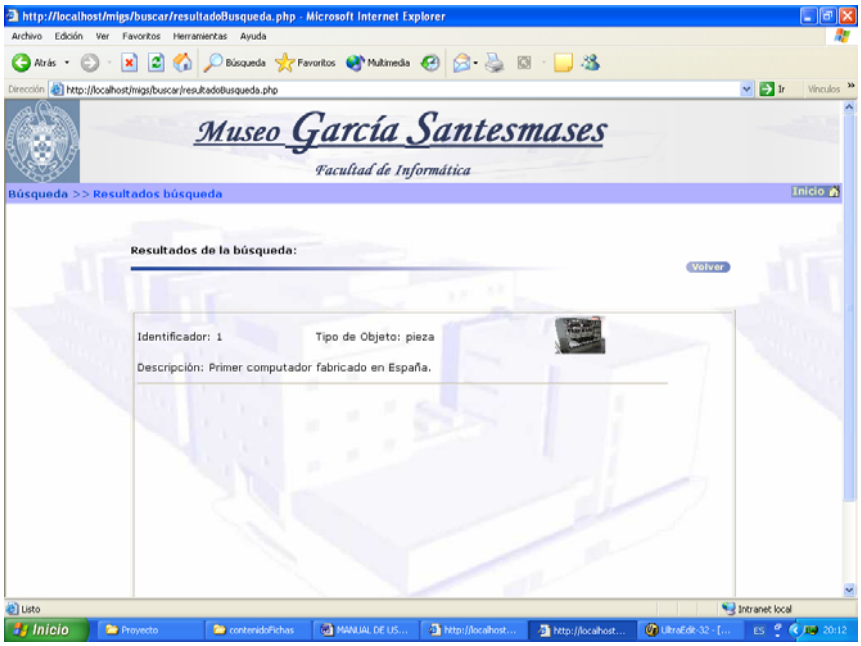


Figura 8. Página de resultados de una búsqueda.

- Pulsando sobre el botón “Ir a un objeto” se va hasta el objeto con el identificador indicado, siempre y cuando exista, en otro caso aparece un mensaje indicando que el objeto no existe.

2. USUARIO INVESTIGADOR

El usuario investigador tiene la capacidad de crear nuevos objetos virtuales, subir objetos virtuales al museo, modificar objetos del museo y bajar objetos para poder exportarlos a otros sistemas.

En un trabajo futuro debe mejorarse la funcionalidad de este tipo de usuario, ya que actualmente no se diferencia entre investigadores y administrador. Una de las funcionalidades que podría implementarse sería que los objetos tuvieran un usuario propietario (el usuario investigador que los creó), de forma que no todos los usuarios investigadores tuvieran la capacidad de borrar objetos del museo ni modificarlos. Existiría un usuario especial que sería el administrador del museo y que tendría todos los privilegios.

- Pulsando sobre el enlace Investigador se llega a la página mostrada en la figura9, donde el usuario debe introducir su nombre de usuario y su clave para poder acceder al museo.

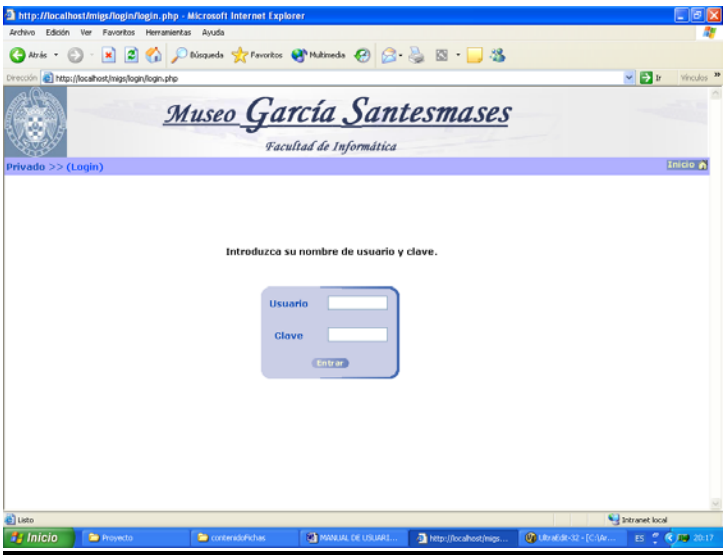


Figura 9: Login del usuario investigador

Para poder acceder como investigador del museo el usuario debe estar registrado en la base de datos con un nombre de usuario y una clave que debe introducir para identificarse antes de acceder a las fichas. Una vez introducido el nombre de usuario y contraseña, se accede a la página mostrada en la figura 10.

Figura 10a.Vista de un objeto virtual para un usuario investigador

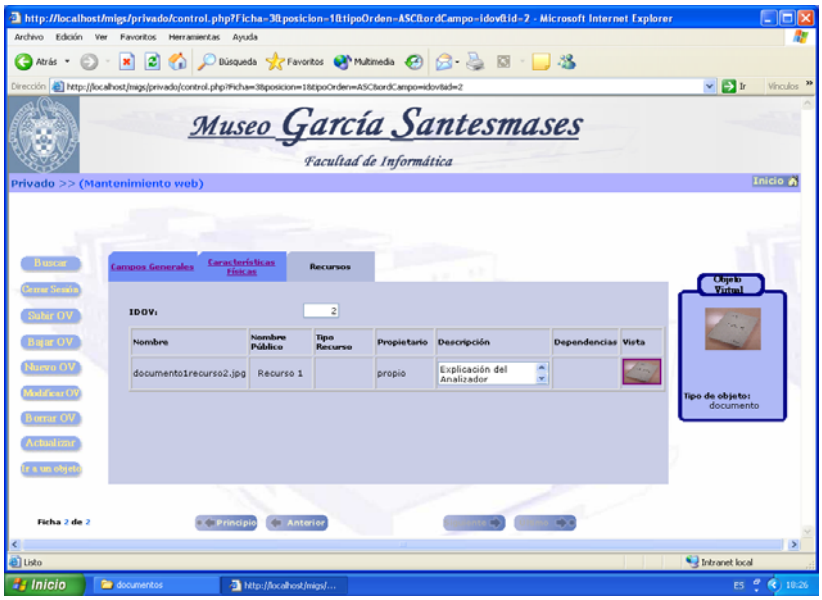


Figura 10b. Vista de la ficha de recursos de un objeto virtual.

Se observa que es la misma información que para el usuario visitante pero que aparecen más botones en el lado izquierdo. Cada botón permite realizar una operación:

1. Buscar.
2. Cerrar Sesión.
3. Subir OV.
4. Bajar OV.
5. Nuevo OV.
6. Modificar OV.
7. Borrar OV.

2.1 Buscar

Permite buscar todos los objetos virtuales del museo que cumplan unas características o tengan un determinado valor en unos campos. Este botón es el mimo que el que aparece cuando se accede como visitante al museo.

2.2 Cerrar Sesión

Sirve para cerrar la sesión iniciada por un investigador. Es útil porque elimina ciertas carpetas temporales que no son necesarias y que se crean cuando el investigador empaqueta algún objeto.

Subir OV

Permite subir un objeto virtual empaquetado que se encuentra en la máquina del usuario. Se abre un navegador de archivos que permite acceder hasta la carpeta comprimida que se quiere subir, figura 11. Cuando un objeto se sube se comprueba si está bien empaquetado y si es válido para pertenecer al museo. Para que el objeto sea válido, la carpeta debe tener un archivo manifiesto (imsmanifest.xml), un recurso llamado objetovirtual.xml donde se recogen los datos de las fichas y las dependencias entre ese objeto y otros del museo. En caso de que el objeto no sea válido, es decir no haya sido empaquetado con la herramienta que dispone el museo, se muestra un mensaje de error.

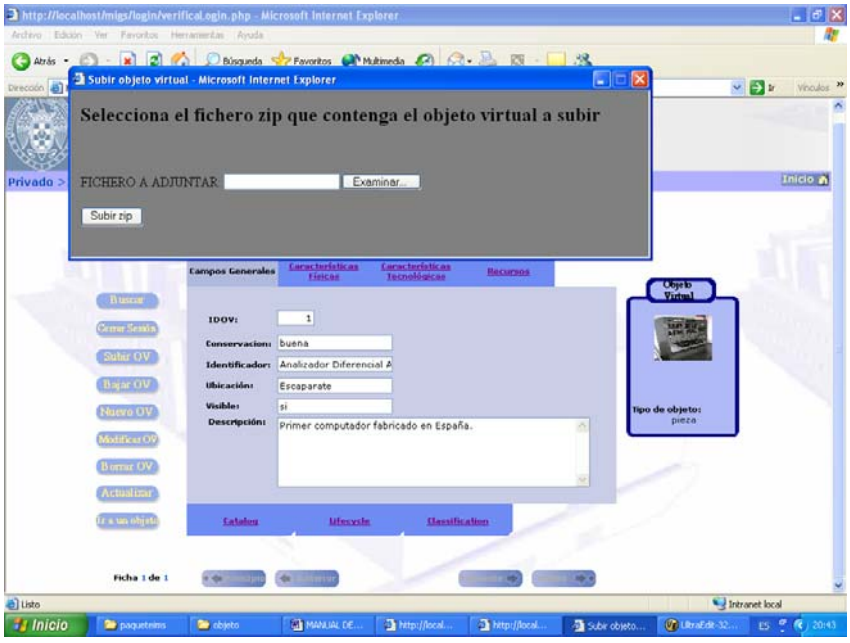


Figura 11. Subir OV

2.4 Bajar OV

Permite bajar o exportar un objeto virtual del museo para que pueda ser utilizado en otros sistemas. Este objeto baja empaquetado según el estándar de IMS.

El objeto que se baja es una carpeta comprimida que contiene los siguientes archivos:

- El archivo manifiesto (imsmanifest.xml).
- Los esquemas para validar este archivo xml (ims_xml.xsd, imsmd_v1p2p2.xsd e imscp_v1p1p3).
- Los recursos propios del objeto, entre los que se encuentra uno especial, objetovirtual.xml. El recurso objetovirtual.xml contiene todos los datos de todas las fichas del objeto.
- La DTD del objetovirtual.xml.
- Una hoja de estilo xsl que permite visualizar de una manera amigable el archivo objetovirtual.xml, y que se llava ObjetoVirtual.xsl. Abriendo este archivo se tiene una visión del objeto en formato html. Este html contiene todos los datos de todas las fichas del objeto, así como enlaces a los recursos propios y a otros archivos objetovirtual.xsl de otros objetos virtuales de los que dependía el objeto principal.
- Todos los objetos de los que depende el objeto virtual también empaquetados, y todos los objetos de los que dependen éstos también empaquetados y en un mismo nivel.

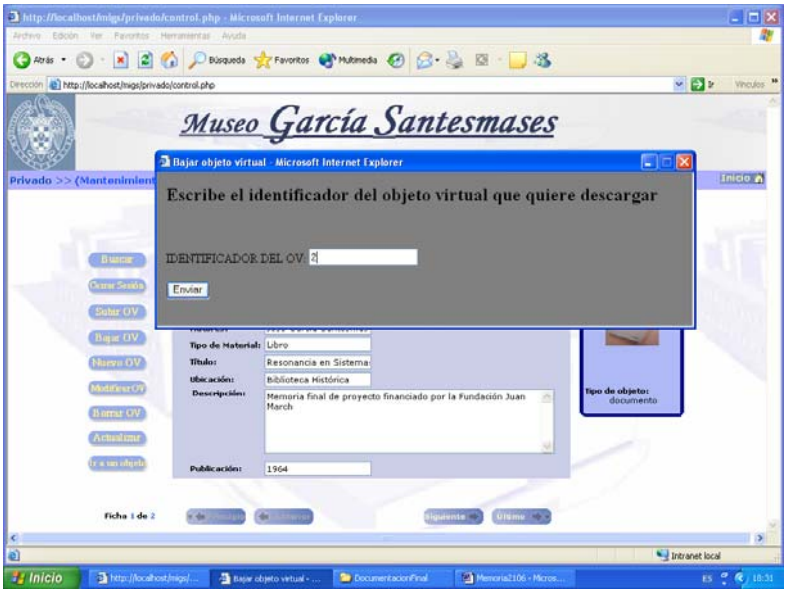


Figura 12. Bajar un objeto virtual del museo.

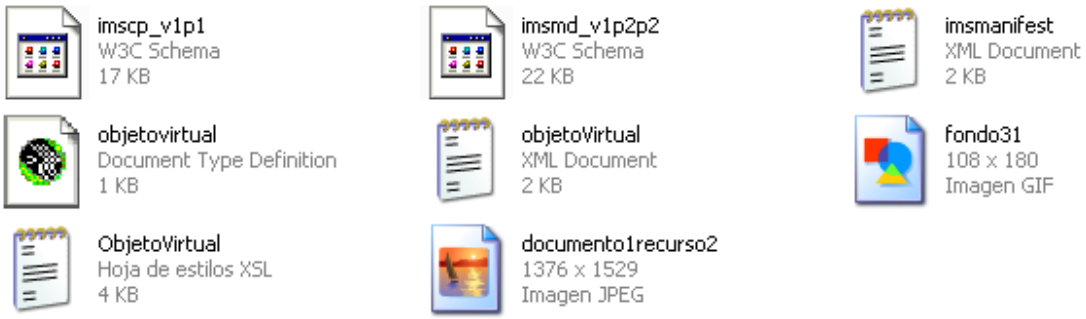


Figura 13. Contenido del objeto que se ha bajado del museo.

2.5 Nuevo OV

Con esta opción se permite crear un nuevo objeto virtual que podrá ser subido posteriormente al museo. Pulsando sobre este botón se accede a la herramienta de empaquetado que aparece en la figura 13-a.

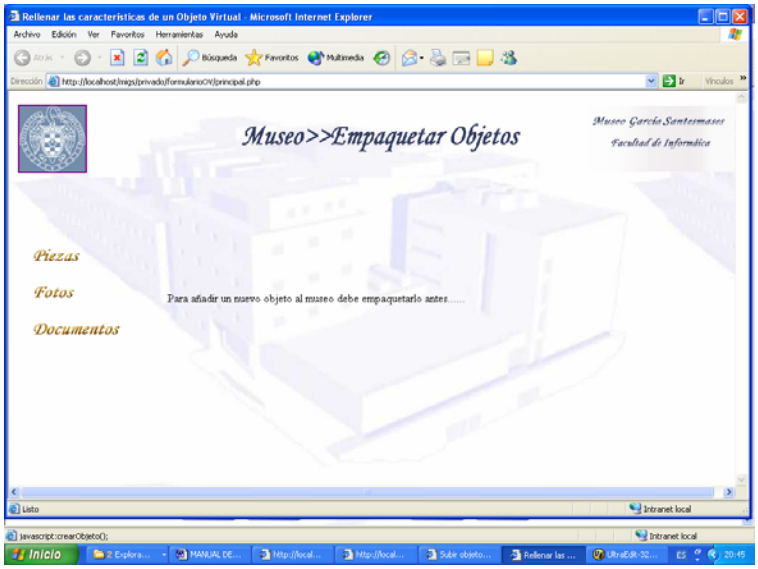


Figura 14-a. Herramienta de empaquetado

En el museo existen tres tipos de objetos: Piezas, Fotos y Documentos, por tanto lo primero que hay que hacer es seleccionar el tipo de objeto se va a empaquetar. Dependiendo del objeto se rellenan unas fichas u otras.

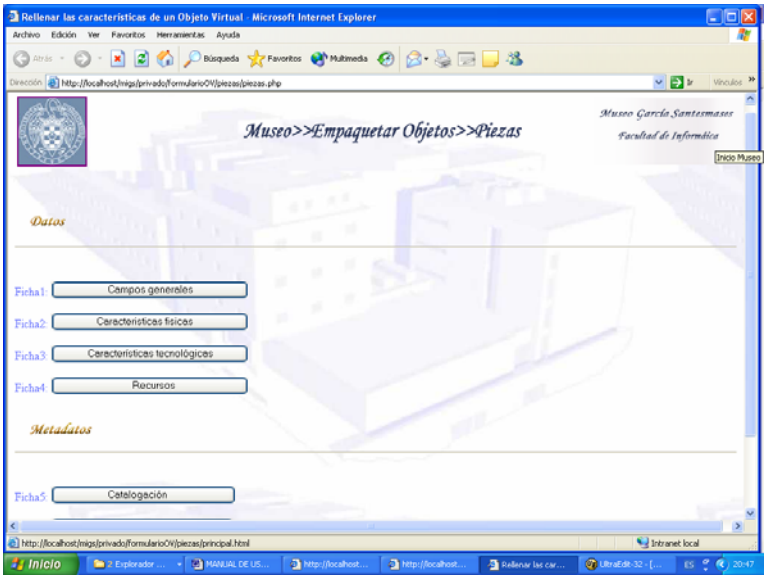


Figura 14-b. Página desde la que se accede a todas las fichas que se pueden rellenar para empaquetar una pieza.

Se encuentran diferenciadas entre las fichas de datos y las fichas de metadatos. Entre las de datos se encuentra la ficha Recursos, que es donde se añaden los recursos y las dependencias del objeto virtual.

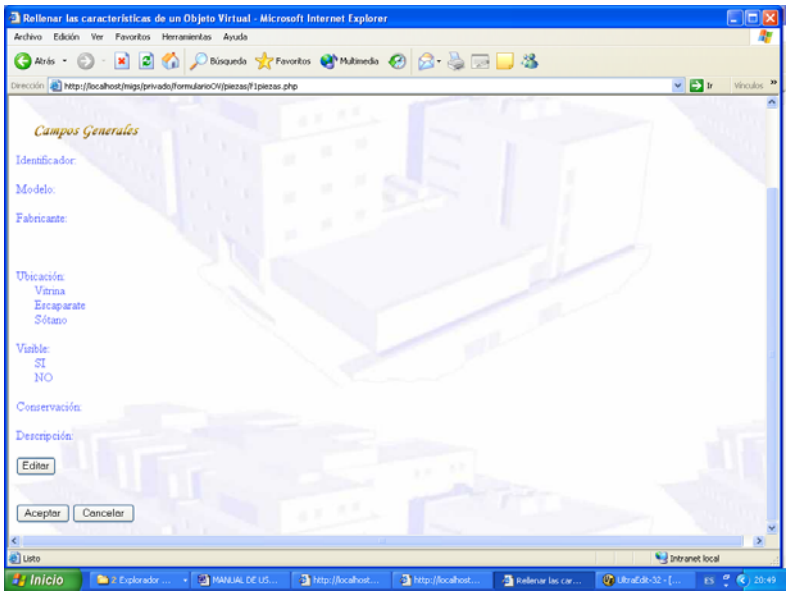


Figura 14-c. Campos de la ficha "Campos generales" de una pieza.

Pulsando sobre el botón editar de la figura 13-c se accede a la página donde se rellena los datos, figura 14-c. Una vez introducidos los datos en la página de la figura 13-d se vuelve a la página 13-c que tendrá los campos rellenos, si el usuario está conforme debe pulsar sobre el botón "Aceptar", todos los datos de la ficha quedan guardados y se vuelve a la página de la figura 13-b donde se puede volver a rellenar otra ficha si se desea. En caso de que el usuario no esté conforme con algún dato se puede modificar pulsando de nuevo sobre el botón "Editar".

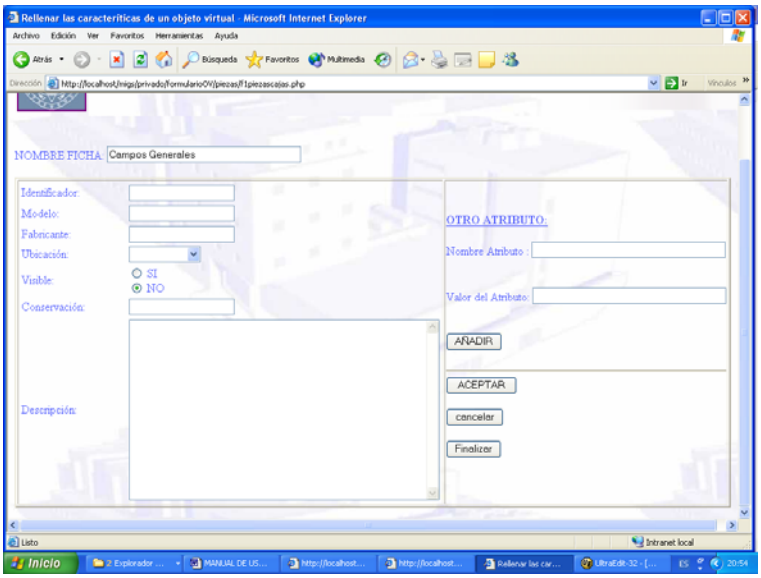


Figura 14-d. Formulario para rellenar los datos de la ficha.

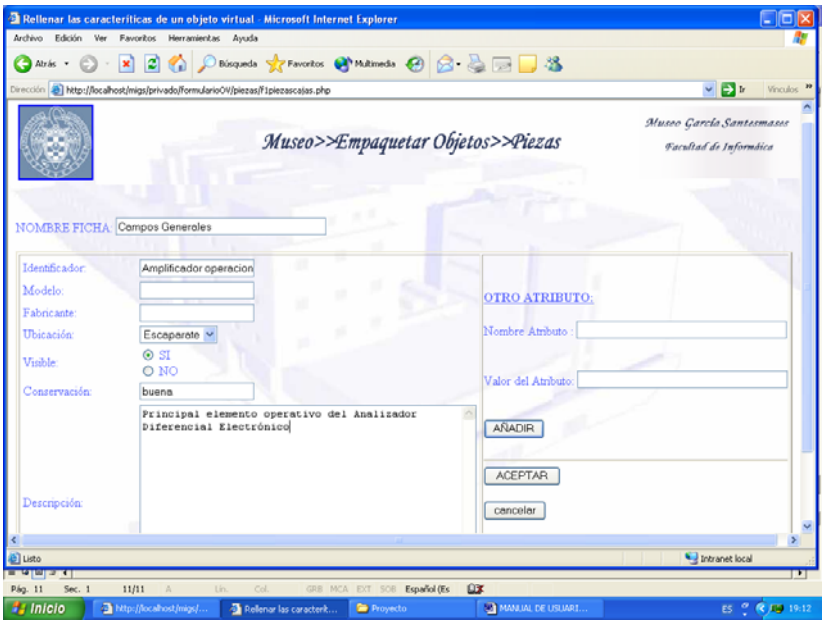


Figura 14-e. Formulario de una ficha relleno

Una vez de acuerdo con los datos se debe pulsar en el botón **ACEPTAR** para guardar los cambios y si se está conforme se debe pulsar en el botón **FINALIZAR**.

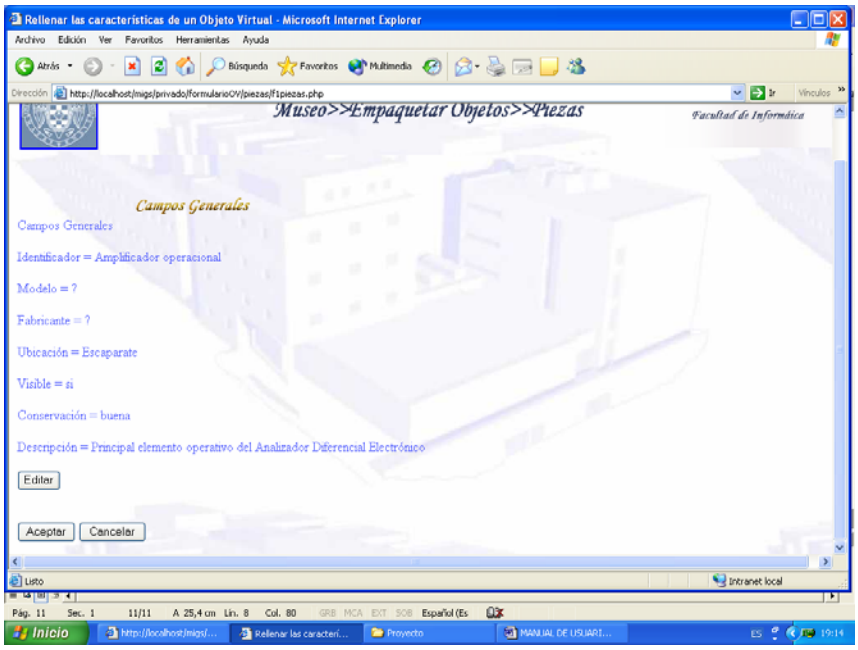


Figura 14-f. Página donde se muestran los datos introducidos

Una vez pulsado en el botón **Finalizar** se llega a la página de la figura 13-f, donde se muestran los datos introducidos por el usuario, en caso de que no esté conforme se pueden modificar volviendo a hacer clic en el botón **Editar** que llevará hasta la página de la Figura 13-3. En caso de que el usuario esté conforme con los datos introducidos debe pulsar en el botón **Aceptar** y volverá a la página de la Figura 13-b para seguir introduciendo nuevas fichas al objeto virtual.

Una vez el usuario haya rellenado todas las fichas que deseé debe introducir un nombre para el nombre del paquete que se va a crear en la caja de texto de la figura 14 y pulsar sobre el botón **"Crear Objeto Virtual".**

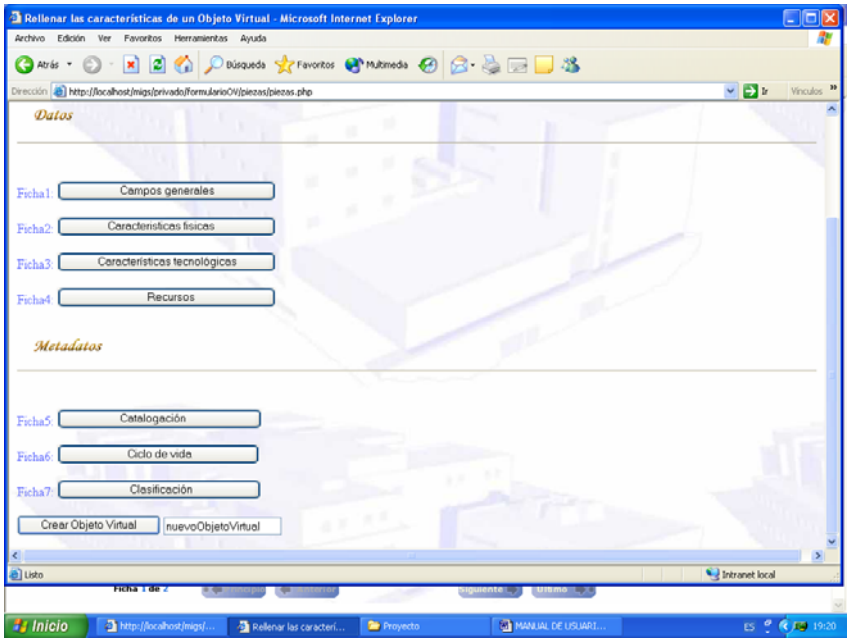


Figura 15. Página desde donde se acceden a todas las fichas del objeto.

Pulsado el botón **"Crear Objeto Virtual"** se llega hasta una página de información que indica que el objetoVirtual.xml ha sido creado (recurso imprescindible para el paquete) y un botón **"Crear Manifiesto".** Una vez creado el objeto se puede subir al museo.

2.6 Modificar OV

Aunque actualmente todos los usuarios investigadores pueden modificar objetos, esto debe mejorarse en un futuro y permitir que sólo el usuario investigador propietario de un objeto o el administrador pueda modificar un objeto del museo.

2.7 Borrar OV

Igual que la opción anterior, aunque actualmente cualquier investigador puede borrar un OV, se debe controlar que sólo el propietario o el administrador pueda eliminar un objeto del museo.

Sólo se puede borrar objetos de los que no depende ningún otro objeto. Con esto se evitan los posibles errores en la navegación entre objetos relacionados.

9. ÍNDICE DE PALABRAS

Se autoriza a la Universidad Complutense de Madrid a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto en la propia memoria, como el código, la documentación y/o el prototipo desarrollado para la asignatura de Sistemas Informáticos del curso 2003/2004 bajo el nombre "Uso y Gestión de Objetos de Aprendizaje en la Web".

Las Autoras:

Patricia Díaz Ayuso

Beatriz Díaz García

Concepción Sanz Pineda