SISTEMAS INFORMÁTICOS
Curso 2006-2007

Creación, gestión y uso de "objetos de aprendizaje" en los Sistemas Chasqui

Jose Luis Rojo Rojo
Miguel Ángel Alonso Pajuelo

Dirigido por:

Alfredo Fernández-Valmayor Crespo
Dpto. Sistemas Informáticos y Programación

Facultad de Informática
Universidad Complutense de Madrid

Contenido

Agradecimientos

Nos es grato haber contado con la ayuda y colaboración de nuestro director de proyecto Alfredo Fernández-Valmayor por el tiempo que nos ha dedicado y por su asesoramiento técnico.

RESUMEN

El proyecto "Creación, Gestión y Uso de 'Objetos de Aprendizaje' en los Sistemas Chasqui" se ha desarrollado en la asignatura Sistemas Informáticos. Tiene como objetivo el análisis de la versión 2.0, actualmente en producción, del Museo Virtual basado en el Museo físico del Departamento de Historia de América II, de la Facultad de Geografía e Historia de la Universidad Complutense de Madrid.

Este proyecto se centra en profundizar en el modo de trabajo de la nueva arquitectura utilizada a modo de prototipo para la construcción de una herramienta que sirva para la creación y gestión de recursos educativos, *Objetos de Aprendizaje*, modulares que puedan ser utilizados por diversos sistemas de enseñanza.

ABSTRACT

The proyect called "Creation, Management and Use of 'Learning Objects' in the Chasqui Systems" has been developed into the subject "Sistemas Informáticos". The goal is to analyze the 2.0 version, currently in production, of the "Virtual Museum" based on the physical Museum of the Department of History of America II, located at the "Facultad de Geografía e Historia" of the Complutense University of Madrid.

This project is centered in deepening in the way of work of the new architecture used as a prototype for the construction of a tool that is used for the creation and management as educative resources, *Learning Objects*, modular objects that they may be used by various learning systems.

PALABRAS CLAVE

E-Learning, lenguajes de marcado, XML, PHP, Java, tecnologías web.

1. Introducción

El presente documento ofrece una visión sobre las actividades realizadas en la asignatura "Sistemas Informáticos, Curso 2006 - 2007", la cual ha estado enfocada en el proyecto "Creación, Gestión y Uso de Objetos de Aprendizaje en los Sistemas Chasqui", enmarcado dentro del área e-learning.

Este proyecto se basa en los trabajos realizados en los cursos anteriores, en los que se creó una herramienta que permite crear y gestionar recursos educativos, de forma que estos puedan ser utilizados por otros sistemas de contenidos. Los "Objetos de Aprendizaje" creados y gestionados mediante dicha herramienta, se corresponden con los objetos del Museo Universitario Colecciones de Arqueología y Etnología de América, y del Laboratorio de Arqueología Americana, del Departamento de Historia de América II de la Facultad de Geografía e Historia. La herramienta creada fue bautizada con el nombre de Sistema Chasqui, utilizado como base del Museo Virtual de dicho Departamento.

Uno de los requisitos importantes de la aplicación es la transportabilidad de los objetos de aprendizaje creados, de tal manera que éstos puedan ser reutilizados dentro de otros entornos y sistemas de contenidos, así como poder ser importados desde otras aplicaciones (siempre y cuando cumplan éstas un determinado estándar).

El objetivo del anterior proyecto se centraba en el análisis de Chasqui v.2, sistema el cual estaba pensado que se pondría en producción a lo largo de ese curso. Actualmente todavía se encuentra en producción la versión 1. Su trabajo se centró en la versión 2, con la finalidad de analizar la estructura y diseño de este sistema, de identificar las limitaciones que presentaba el modelo y ofrecer una nueva arquitectura a modo de prototipo para superar dichas limitaciones. Debido a la gran densidad del sistema, la nueva arquitectura solo cubrió una parte funcional de la aplicación, la cual nos sirve de base para nuestro proyecto y para los de años posteriores.

El objetivo actual del proyecto es profundizar en el modo de trabajo de la nueva arquitectura utilizada a modo de prototipo. Las novedades son básicamente dos, en primer lugar por orden de importancia y complejidad tenemos la utilización del framework ZNF donde se realiza la implementación del nuevo Chasqui, y en segundo lugar tenemos la introducción y aplicación de la tecnología AJAX que es una técnica de desarrollo web para aplicaciones interactivas. Nuestro trabajo se centra en el estudio del framework ZNF, intentando en la medida de lo posible explicar su estructuración y funcionamiento para que sirva como una base sólida para ayuda a futuros grupos que continúen este proyecto. De este modo intentamos conseguir que en años posteriores no se gaste tanto tiempo en buscar información y

comprender dicho framework, ya que resulta un trabajo costoso de por si comprender desde cero una aplicación tan extensa como es Chasqui. A parte de estudiar el framework ZNF, también hemos añadido un par de funcionalidades, como la de navegación por objetos a partir de su clasificación, y la de insertar un objeto directamente en la base de datos (servirá como base para la creación de Objetos Virtuales y su posterior inserción).

2. Resumen recordatorio de lo más importante

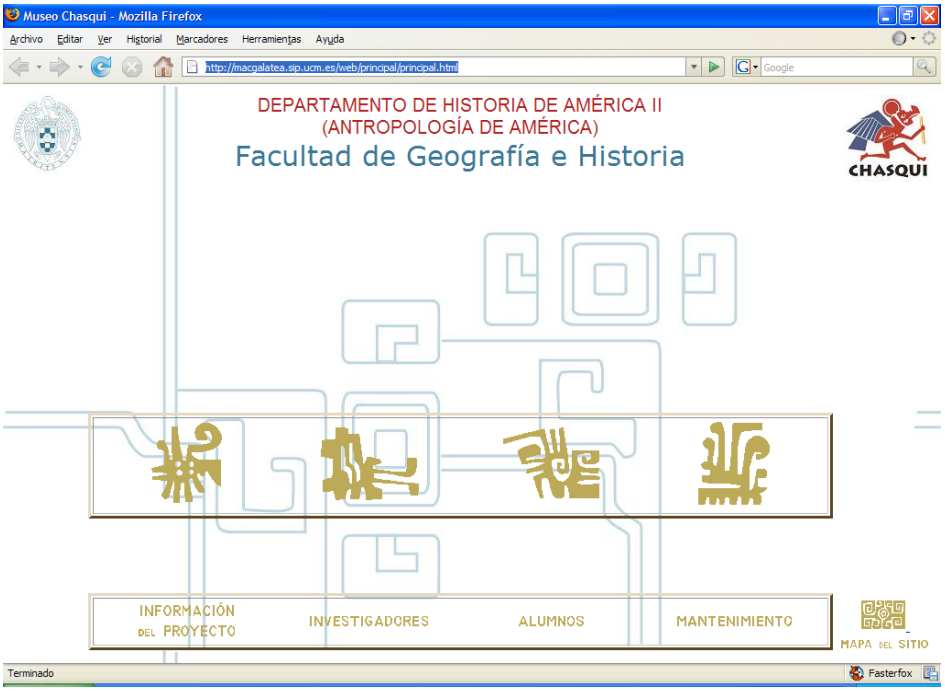
2.1. Chasqui

El concepto de "*Objeto de Aprendizaje*" (digitalización objetos reales) nace en el año 1992 inspirado precisamente en el juego Lego. Quienes fueron dotándolo de significados fueron los principales grupos desarrolladores de soluciones de "e-learning" y "metadata": entre 1992 y 1995 "The Learning Object Metadata Group" del National Institute of Science and Technology; entre 1994 y 1996 comenzaron los aportes de la IEEE, IMS, ARIADNE y Oracle; en 1998 aparece el informe de Cisco sobre "Reusable Learning Objects" y de ahí en adelante es tanta la actividad que es difícil distinguir los aportes.

Chasqui es un sistema de contenidos encargado de manejar "*Objetos Virtuales*", cuya estructura está basada fundamentalmente en los "*Objetos de Aprendizaje*". Estos "*Objetos Virtuales*" hacen referencia a los objetos físicos del Museo Universitario Colecciones de Arqueología y Etnología de América, y a los del Laboratorio de Arqueología Americana, ambos de la Facultad de Geografía e Historia. Chasqui es el fruto de la cooperación entre dos Facultades de la Universidad Complutense de Madrid: la Facultad de Geografía e Historia, con el departamento de Historia de América II (Antropología de América), y la Facultad de Informática con miembros de los departamentos de Ingeniería del Software e Inteligencia Artificial, y de Sistemas Informáticos y Computación.

Este museo virtual, sirve de apoyo a la docencia, al trabajo de los investigadores y a la creación de puntos de información para difusión cultural, cuya finalidad es convertir en recursos educativos digitales los materiales arqueológicos y etnográficos del Museo Universitario Colecciones de Arqueología y Etnología de América, y del Laboratorio de Arqueología Americana.

Esta herramienta de gestión es utilizada como una herramienta de trabajo por investigadores y alumnos como apoyo en la docencia de varias asignaturas y doctorados, es por ello que la información almacenada en su base de datos crezca de forma notoria.



http://macgalatea.sip.ucm.es/web/principal/principal.html

2.2. Estándares de la Enseñanza en Línea (e-learning)

En la actualidad existen varios "Sistemas de Gestión de Aprendizaje" (LMS) y de "Entornos de Aprendizaje Virtual (VLE), pero todos ellos carecen de la posibilidad de poder intercambiar contenidos y estructuras de aprendizaje con otros sistemas. Debido a esto son necesarias unas especificaciones y unos estándares para los sistemas de aprendizaje virtual.

Un estándar es una tecnología, formato o método, reconocido nacional o internacionalmente, documentado en detalle y ratificado por una autoridad competente en su campo, como ISO ó IEEE. Por el contrario, una especificación es el paso previo, creado por alguna compañía u organismo, que no ha sido ratificado todavía por ninguna autoridad, y que suele usarse de manea provisional pero suficientemente respaldada.

En la actualidad sólo están disponibles especificaciones para la enseñanza en línea, las permiten modelar un elemento o una etapa del proceso educativo, trabajar con él y mantener el nuevo material funcionando exactamente igual, independientemente de la plataforma que se utilice. Es decir, puede ser migrado automáticamente y el contenido y estructura del curso son independientes de la plataforma de ejecución.

La propuesta de LOM (Learning Object Metadata) 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:

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

Los metadatos usados en Chasqui se han centrado en este estándar tomando como referencia las categorías *General*, *Ciclo de Vida* y *Clasificación*.

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 Arqueología de la Facultad de Geografía e Historia de la Universidad Complutense de Madrid. Es decir, lo que buscamos es crear *Objetos Virtuales* a partir de los objetos disponibles en dicho museo, así como las posibles relaciones que se puedan establecer entre ellos.

Cada objeto virtual se describe mediante un modelo de metadatos basado en el estándar LOM. 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.

2.3. Análisis del sistema Chasqui (v.2)

Ya que el grupo del año pasado se iba a centrar en mejorar la versión 2 de Chasqui, optaron por conocer el propio sistema, su funcionalidad, su diseño, su estructura y su código.

El sistema Chasqui se trata de una aplicación web cliente/servidor para la cual es necesario disponer de un servidor de páginas web (Apache) con soporte para el lenguaje PHP y un gestor de base de datos (MySQL).

Gracias al software en el que se apoya Chasqui (Apache, PHP y MySQL), estamos ante una aplicación multiplataforma.

Para conocer mejor el sistema sobre el que iban a trabajar instalaron el sistema en sus ordenadores personales. A continuación se dedicaron a detallar los procesos de instalación que siguieron para tener disponible el sistema Chasqui tanto en entorno Windows como en entorno Linux.

Una vez realizada la instalación de Chasqui en sus ordenadores, pasaron a comprender la funcionalidad de la aplicación más detenidamente. Se trataba de comprender los requisitos que presentaba la aplicación , el diseño y la implementación mediante los cuales se cumplían dichos requisitos.

En este punto se planteó el principal objetivo del proyecto del año pasado que debía ser uno de los 2 siguientes:

- Modificar la implementación actual de Chasqui, corrigiendo posibles defectos e intentar modificar su estructura interna para acercarlo lo máximo posible a una aplicación modelo-vista-controlador.

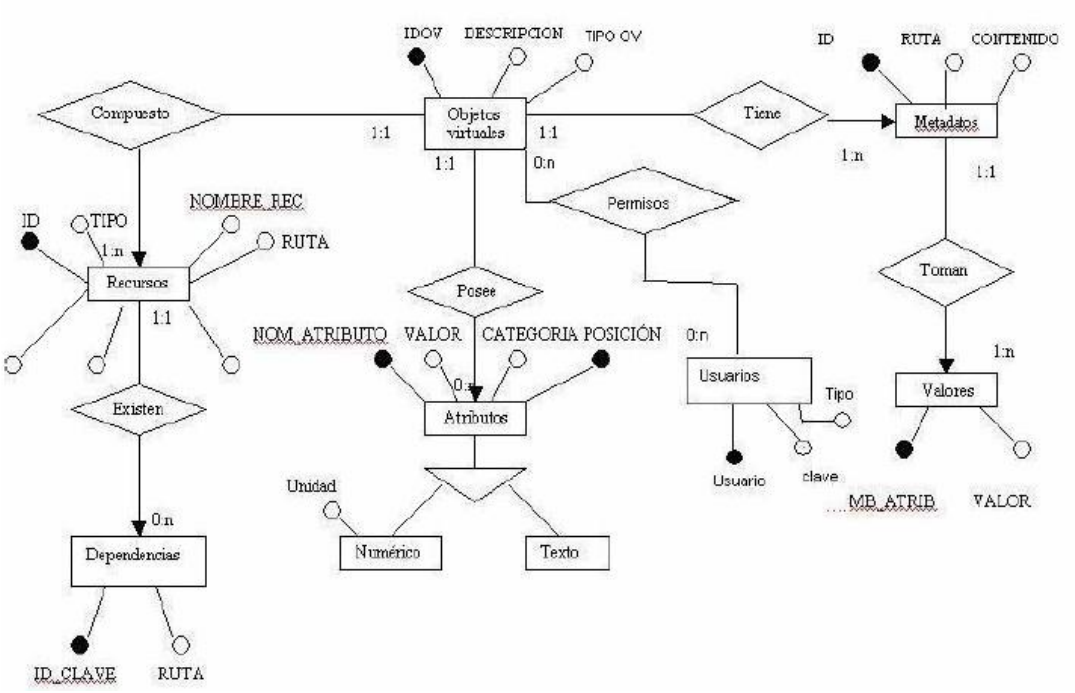
- Rehacer el sistema Chasqui desde el principio para obtener un modelo-vista-controlador puro y de paso corregir algunos de los problemas detectados.

Finalmente, la opción tomada fue rehacer la aplicación estructurándola en un MVC mediante el uso de un framework disponible en el mercado escrito en PHP5. De esta forma la aplicación estaría mejor estructurada, siendo más clara y más mantenible, adaptándose fácilmente a las tecnologías más modernas del mercado.

Debido a que se optó por la opción de rehacer la aplicación por completo, se rebajaron las funcionalidades de la aplicación debido a la carencia de tiempo.

Base de Datos

A continuación se muestra el diagrama entidad – relación que presenta la base da datos de Chasqui v.2



Descripción de cada una de las tablas que componen de dicho diagrama:

Objeto virtual: Entidad principal de la aplicación. Objeto de aprendizaje virtualizado.

IDOV: Clave primaria numérica del objeto virtual.

DESCRIPCIÓN: Breve descripción de texto del objeto virtual.

TIPO_OV: Campo no utilizado en Chasqui.

USUARIO: Usuario creador del objeto, y que tiene permisos para modificarlo.

Recursos: Recursos asociados a cada objeto virtual. Aparecen en la pestaña recursos en las fichas de cada objeto virtual. Los recursos de un objeto virtual pueden ser de 3 tipos:

- Propios del objeto virtual.
- De otros O.V.

- Un O.V. propiamente dicho.

ID: Clave primaria numérica del recurso.

IDOV: Objeto virtual al que pertenece el recurso (recurso propio del objeto virtual).

NOM_REC: Archivo en el que está contenido el recurso.

NOM_REC_PUBLICO: Nombre público del recurso. Es el que aparece en la pestaña recursos como nombre.

TIPOREC: Indica el tipo de archivo que es el recurso (ej: html, jpg , etc). Si el recurso es un objeto virtual, aparece "objeto virtual".

DESCRIPCIÓN: Descripción del recurso.

VISIBLE: Indica si el recurso es visible para los visitantes o no.

TIPO: Extensión del archivo en el que está contenido el recurso. Si se trata de un objeto virtual, aparece "objeto virtual".

RUTA: Indica la ruta donde está localizado el recurso en el directorio "Proyecto/recursos".

Dependencias: Indica los recursos que tienen alguna dependencia (por ejemplo un recurso que es una página web y depende de una hoja de estilo).

ID_CLAVE: Clave primaria numérica.

ID: Identificador del recurso que tiene la dependencia. Clave primaria de la tabla recursos.

RUTA: Ruta del archivo, en el directorio "Proyecto/recursos", con el que se tiene la dependencia.

Atributos_numericos: Atributos de tipo numérico de un objeto virtual. Aparecen normalmente en la pestaña Dimensiones de las fichas de los objetos virtuales en el acceso directo a base de datos.

IDOV: Identificador del objeto virtual al que pertenece el atributo.

NOM_ATRIB: Nombre de dicho atributo (generalmente la dimensión).

VALOR: valor del atributo.

UNIDADES: Unidades de longitud, peso, etc, en las que viene expresadas el valor.

CATEGORIA: Pestaña de la ficha de los objetos virtuales en la que se coloca el atributo.

POSICION: posición de dicha categoría en la ficha del objeto virtual

Atributos_texto: Atributos de tipo texto de un objeto virtual Aparecen en la pestaña General o en la pestaña Análisis de las fichas de los objetos virtuales en el acceso directo a base de datos.

IDOV: Identificador del objeto virtual al que pertenece el atributo.

NOM_ATRIB: Nombre de dicho atributo.

VALOR: Valor del atributo identificado para cada nom_atrib.

CATEGORIA: Pestaña de la ficha de los objetos virtuales en la que se coloca el atributo.

POSICION: Posición de dicha categoría en la ficha del objeto virtual.

Usuarios: Tabla que almacena los usuarios registrados en el sistema. Solo hay un único usuario root (museo). El resto de usuarios son usuarios investigadores.

ID: Clave primaria numérica de la tabla.

USUARIO: Login del usuario.

TIPO_USUARIO: Información sobre el usuario. Solo es informativo.

CLAVE: Contraseña del usuario.

DESCRIPCION: Descripción del usuario.

No hay tipo de usuario como tal. Por ejemplo para eliminar o modificar un objeto virtual se comprueba si el usuario que quiere hacerlo es "root" o si el usuario es el propietario del objeto, solo en ese caso se puede hacer la operación.

Metadatos: Metadatos de un objeto virtual. Sirven para poder clasificar un objeto virtual por diferentes categorías.

ID: Clave primaria numérica.

IDOV: Identificador del objeto virtual al que pertenece el metadato.

RUTA: Es la ruta que tiene el metadato en el manifiesto. Al serializar un objeto virtual siguiendo el estándar IMS_CP se guarda como un fichero .zip. En este fichero comprimido hay un archivo llamado imanifest.xml, en el que se serializan en lenguaje XML los metadatos del objeto. El campo ruta, por tanto, es la ruta del metadato en el archivo imanifest.xml. Son las etiquetas xml abiertas hasta encontrar el metadato.

CONTENIDO: Es el valor del metadato. Lo que aparece en el archivo imanifest.xml tras todas las etiquetas de su ruta.

NUM_RUTA: Codificación de la ruta del metadato en el archivo imanifest.xml.(*)

Atributos_metadatos: Atributos referentes a los metadatos. Se utilizan para serializar los metadatos. Aparecen en el fichero imanifest.xml en la etiqueta anterior al valor del atributo.

ID: Clave primaria numérica.

NOM_ATRIB: Nombre del atributo. Aparece en el fichero imanifest.xml en la etiqueta anterior al valor del metadato. Esta etiqueta se llama langstring.

VALOR: Valor del atributo. Aparece también en la misma etiqueta que el nombre del atributo. Generalmente en el fichero imanifest.xml: <langstring nom_atrib="valor">.

Las tablas descritas anteriormente son las tablas propias del modelo de datos de Chasqui. A continuación se muestran una serie de tablas las cuales se crean a partir de las tablas anteriores. Estas tablas son utilizadas para la navegación por los objetos virtuales a través de sus categorías.

En estas tablas aparecen los objetos virtuales clasificados por el metadato Sección/Sección. Este metadato es el que da nombre a las tablas. En la anterior navegación estática había seis tablas fijas para la navegación, que se correspondían

con los valores del metadato *Sección/Sección*. Estas tablas eran: **material_documental, arqueología, etnología, reproducciones, material_docente y agha**. Si se subía un nuevo objeto virtual, y se le ponía como valor del metadato Sección/Sección uno distinto de los 6 anteriores, se creaba una nueva tabla cuyo nombre fuera el valor introducido.

Para la nueva navegación dinámica, se crean tantas tablas como valores del metadato *Sección/Sección* existan, más otra que nos indicará las secciones que hay actualmente en la base de datos. Su explicación detallada se muestra más adelante, en la implementación de las nuevas funcionalidades.

Estas seis tablas tienen la misma estructura de atributos, el id de cada fila, que es numérico. El *idov* que es el identificador del objeto virtual con el que se corresponde esa fila. El resto de campos de la tabla se corresponden con otros metadatos de tipo Clasificación, que son los que aparecen en la pestaña Clasificación de las fichas de los objetos virtuales. No todos estos campos están rellenos, ya que un objeto virtual no tiene porque poseer todos esos metadatos. Estos campos aparecen rellenos con "sin asignar" si el objeto virtual no posee ese metadato.

Por último hay una tabla llamada **historial** en la cual se reflejan todas las acciones que se realizan los usuarios sobre la aplicación desde la parte de mantenimiento. Estas acciones van desde iniciar sesión, hasta crear, actualizar o borrar los objetos virtuales. Sus campos son:

ID: Clave primaria numérica de la tabla.

ACCION: Describe cuál ha sido la acción realizada por un usuario.

ID_USUARIO: Identificador del usuario. Es la clave primaria numérica de la tabla usuarios.

TIPO_USUARIO: Es el login del usuario. Campo usuario de la tabla usuarios.

ID_OBJETO: Identificador del objeto virtual sobre el que se realiza la acción. Es el campo idov de la tabla objeto_virtual.

TIPO_OBJETO: Indica el componente sobre el que se ha realizado la acción. Puede ser un objeto virtual, metadatos.

FECHA_HORA: Timestamp de la acción.

Posibles mejoras en la base de datos:

En la tabla *objeto_virtual* quitar el campo *tipoOV* el cual siempre contiene el valor 'objeto' y nunca es utilizado.

En la tabla usuarios eliminar el campo *id* que es la clave primaria y poner como clave primaria el nombre de usuario (*login*) ya que no puede haber dos usuarios con el mismo *login*. De esta forma, para evitar el alta de usuarios repetidos, se restringirá directamente con la base de datos y no habrá que buscar en ella si ya existe algún usuario con ese *login*.

Hacer una gestión más avanzada de usuarios, de tal forma que el campo *tipo_usuario* no sea solo descriptivo sino que haya tipos de usuarios distintos, y cada tipo de usuario tenga unos permisos. En este momento solo se contemplan dos tipos de usuarios, pero es posible que en algún momento futuro se quieran crear nuevos tipos de usuarios. De esta forma, cuando se crea un nuevo usuario se le puede asignar el tipo de usuario que se desee.

Adicionalmente, si se quiere cambiar el *login* del usuario *root*, actualmente *museo*, esto implica cambios en el código cada vez que se cambie el *login*. Si se pone *tipo_usuario* se podrá hacer una selección *("select")* que obtiene el usuario tipo *root* de tal forma que si se quiere cambiar el *login* no haya que modificar el código.

En la tabla historial se almacena el *id* y el *login* del usuario que realiza la acción. Esto es redundante ya que con conocer uno de los dos, sabemos unívocamente el usuario que realiza la acción. Lo ideal sería, partiendo de que el campo id de usuario debería desaparecer, almacenar únicamente el nombre de usuario.

2.4. Instalación usando el paquete XAMPP

Existe una forma más sencilla de realizar la instalación del entorno web necesario para Chasqui (Apache + PHP + MySql). Esta forma es mediante el paquete XAMPP. Este paquete incluye Apache, MySql y PHP ya configurados, de manera que para hacer funcionar la aplicación web no es necesario modificar ningún fichero de configuración de PHP ni de Apache.

Para instalar XAMPP lo único necesario es bajarse el paquete XAMPP y descomprimirlo. Para desinstalarlo solo basta con eliminar el paquete XAMPP.

XAMPP es gratuito y su compilación está hecha bajo licencia GPL. XAMPP está pensado para ayudar a los desarrolladores a introducirse en el mundo de entornos de desarrollo web basados en PHP y MySql. Sin embargo XAMPP no es seguro para usarlo en un entorno productivo. Nunca debe utilizarse para ningún sistema que se encuentre en producción.

Para usar XAMPP con un sistema operativo Windows basta con descargarse el paquete XAMPP y descomprimirlo. Para configurarlo es necesario ejecutar el archivo *setup_XAMPP.bat* (esto no es necesario solo si XAMPP no se sitúa en un directorio raíz de una partición), luego es necesario arrancar los servicios PHP y MySql, para ello basta con ejecutar el archivo *XAMPP_start.exe.* A partir de este momento ya se tiene el entorno preparado para desarrollar la aplicación web. Para evitar posibles problemas en la utilización de XAMPP, conviene dejar el Document Root de Apache como viene y situar el directorio raíz de nuestra aplicación en ../XAMPP/htdocs.

Existen versiones de XAMPP para los siguientes sistemas operativos:

- Linux: XAMPP Linux 1.5.3a (Apache 2.2.2, MySQL 5.0.21, PHP 5.1.4 & 4.4.2).
- Windows: XAMPP 1.5.3a (Apache 2.2.2, MySQL 5.0.21, PHP 5.1.4).
- MacOS X: XAMPP MacOS X 0.5 (Apache 2.0.55, MySQL 5.0.15, PHP 4.4.1, PHP 5.0.5).
- Solaris: XAMPP Solaris 0.8.1 (Apache 2.2.0, MySQL 5.0.18, PHP 5.1.1)

Toda la información relativa al paquete XAMPP se puede encontrar en la página web http://www.apachefriends.org/en/XAMPP.html.

Importación de la base de datos

Toda la información contenida en la base de datos se puede salvar a un fichero de texto mediante la aplicación *MySQL Dump*. Posteriormente el contenido de dicho fichero puede ser importado a otros servidores MySQL albergados en otras máquinas.

Al restaurar la base de datos mediante el fichero anterior, para evitar problemas con las tablas InnoDB y con las restricciones sobre claves externas, hay que realizar las siguientes modificaciones:

Al principio del fichero agregar:

```
SET AUTOCOMMIT=0;
SET FOREIGN_KEY_CHECKS=0;
DROP DATABASE IF EXISTS `chasqui2pr`;
CREATE DATABASE `chasqui2pr`;
USE chasqui2pr;
```

Al final del fichero agregar:

```
SET FOREIGN_KEY_CHECKS=1;
COMMIT;
SET AUTOCOMMIT=1;
```

Una vez realizadas dichas modificaciones en el fichero, la importación de la base de datos se realiza desde el cliente *mysql* mediante el siguiente comando:

mysql –u nombre_usuario –p password < chasqui.sql

2.5. Arquitectura Modelo vista controlador (MVC)

El modelo vista controlador (MVC) es el patrón de diseño recomendado para aplicaciones interactivas. MVC organiza una aplicación interactiva en tres módulos separados:

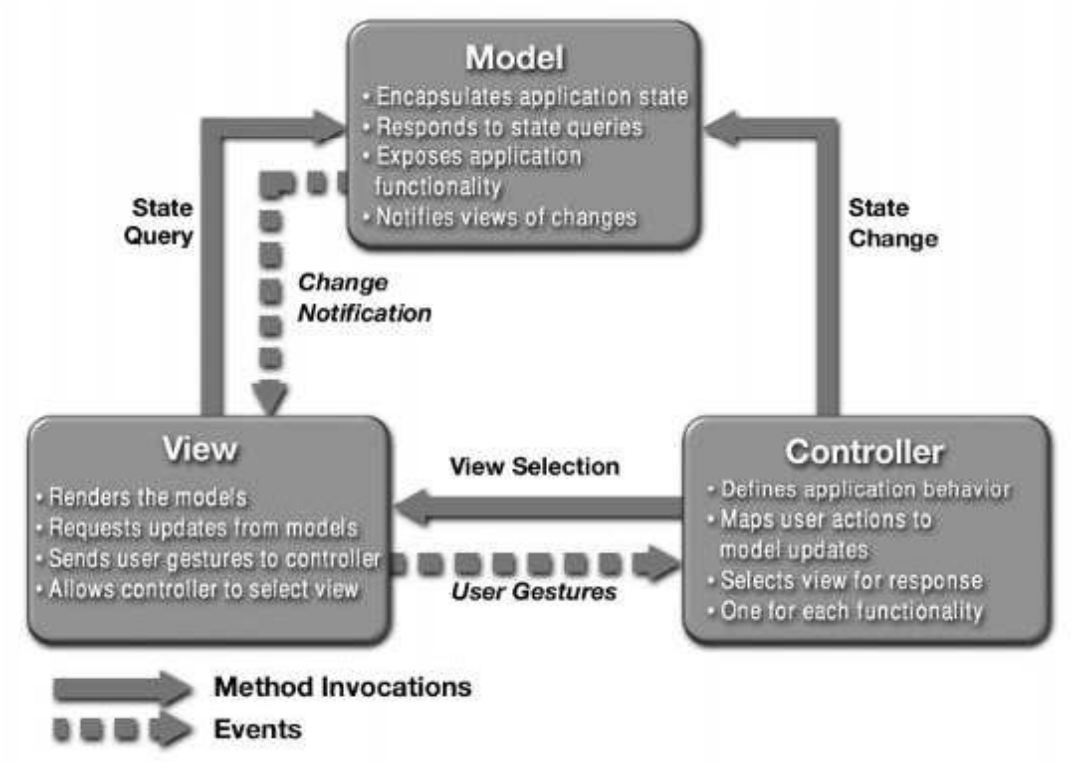
- El primero, para el modelo de la aplicación, con su representación de datos y la lógica de negocio.
- El segundo, para las vistas que proveen de la presentación de los datos y la entrada de usuario.
- El tercero, para el controlador que envía las peticiones y lleva el control del flujo de la aplicación.

La mayoría de los frameworks MVC a nivel de web usan alguna variación del patrón de diseño MVC. El patrón de diseño MVC provee gran cantidad de beneficios de diseño. MVC separa asuntos de diseño (persistencia de datos y comportamiento, presentación y control) decrementa la duplicación de código, centraliza el control y hace que la aplicación sea más fácilmente modificable.

El diseño del MVC permite centralizar el control de prestaciones de una aplicación tales como la seguridad, el inicio de sesión y el flujo de pantallas. En una

aplicación MVC es fácil agregar nuevas fuentes de datos creando código que adapte la nueva fuente de datos al código existente. De manera similar, nuevos tipos de clientes pueden ser agregados para operar con la vista del MVC.

MVC define claramente la responsabilidad de cada clase, haciendo que los errores sean más fáciles de detectar y eliminar.



Hay dos modelos de la arquitectura MVC:

El modelo 1 es una arquitectura en la cual, desafortunadamente muchos desarrolladores basan el diseño de sus aplicaciones web. Consiste en que el navegador accede directamente a las páginas PHP. Las páginas PHP acceden a los datos que representan el modelo de la aplicación y la siguiente vista a mostrar es determinada por el hipervínculo seleccionado en el documento fuente o por los parámetros de la petición. En control en una aplicación basada en el modelo 1 es descentralizado debido a que la página que actualmente está siendo mostrada determina cual será la siguiente página a mostrar. Adicionalmente, cada página PHP o clase procesa sus propias entradas (parámetros obtenidos a través de GET o POST).

El modelo 2 es una variante web del patrón MVC, el cual difiere del original porque la vista no tiene la capacidad de "escuchar" al modelo en orden de recibir las

notificaciones de los cambios de estado. De hecho en las aplicaciones web, las vistas son generadas bajo demanda y no en tiempo real como sucede en las aplicaciones GUI. La arquitectura del modelo 2 difiere de la del modelo 1 porque introduce una clase controlador entre el navegador y el contenido que está siendo entregado. El controlador centraliza la lógica y envía las peticiones a la siguiente vista basándose en la petición URI, en los parámetros de entrada y en el estado de la aplicación. El controlador además ayuda a la selección de la vista, lo cual permite desacoplar las páginas PHP y las clases unas de la otras. Las aplicaciones basadas en el modelo 2 son fácilmente mantenidas y ampliadas, porque las vistas no se refieren directamente. La clase del controlador del modelo 2 provee de un único punto de control para seguridad e inicio de sesión y frecuentemente encapsula los datos de entrada en un formulario usable por el back-end del modelo MVC. Por estas razones, la arquitectura del modelo 2 es recomendada para la mayoría de las aplicaciones interactivas.

2.6. Framework elegido: ZNF

Después de analizar diferentes frameworks y de valorar sus pros y contras, se decidió utilizar el framework ZNF basado en el modelo 2 del patrón MVC, debido a la alta documentación, a la filosofía de trabajo basada en Yakarta Structs y por apoyarse en otras tecnologías ya consolidadas en el mercado con el uso de PEAR::DB y Smarty.

De aquí en adelante intentaremos dejar lo más claro posible la estructuración y funcionamiento del framework ZNF.

3. Framework ZNF

3.1. Introducción

El objetivo de este proyecto es proporcionar un framework de código abierto para la construcción de aplicaciones web PHP5 para empresas. La filosofía de implementación y de trabajo está tomada de Jakarta Struts (algo muy positivo) y se basa en el modelo 2 del patrón MVC. El núcleo del framework está basado en XML (al igual que Struts) y PHP 5.

Dispone de un controlador propio personalizable, el cual realiza el manejo de eventos mediante configuraciones en XML (esto proviene de Struts). El modelo puede interactuar con estándares de acceso a base de datos tipo PEAR::DB. La vista trabaja con Smarty y transformaciones XSL

El tipo de licencia es GNU/LGPL (menos restrictiva incluso que GNU/GPL)

La API ha sido creada con phpDocumentor y está completamente documentada

Existe un amplio manual de referencia en inglés.

El proyecto está disponible al público desde Abril de 2005 y cada mes (más o menos) han ido sacando una nueva versión corrigiendo errores y agregando funcionalidades.

La versión utilizada en este proyecto es la 0.7.6, disponible desde octubre de 2005. Adicionalmente la versión en desarrollo siempre se encuentra disponible a través del sistema Subversión.

La última versión es la 0.7.9, y salió el 9 de abril del 2007, pero no contiene cambios importantes y hemos decidido no actualizar este año de momento.

A diferencia de otros frameworks de PHP, ZNF lee la configuración de un archivo XML, creando un array asociativo con los parámetros de configuración y almacenándolos en disco. En la primera petición se parsea el XML y se crea el fichero en disco, de tal forma que las siguientes peticiones leen el fichero escrito, siempre y cuando el XML no haya sido alterado. De esta forma se consigue un mayor rendimiento (En Struts debido al servidor de aplicaciones las configuraciones leídas al arrancar la aplicación son mantenidas en memoria y no hace falta este sistema de "cacheado").

La codificación del framework sigue el estándar de codificación de PEAR (http://pear.php.net/manual/en/standards.php).

En diciembre del año 2005 ha sido incorporado oficialmente este framework a la distribución Gentoo (Linux), algo bastante positivo ya que indica que hay una comunidad interesada en este proyecto.

La web http://www.zeronotice.org está disponible en inglés, francés, italiano y ruso, aunque la documentación y la API están solo disponible en ingles (algo lógico, ya que hay muy pocos proyectos donde exista la documentación en varios idiomas).

3.2. Requisitos

Comenzamos a recoger los requisitos para el framework completo, para capturar todas las restricciones que el sistema debe contemplar.

0.1. El framework se basa en el patrón de diseño MVC.
  0.1.1. El flujo de la aplicación es mediado por un controlador central.
  0.1.2. El comportamiento del framework puede ser modificado por los componentes de la aplicación.
0.2. La parte del controlador se centra en la recepción de peticiones del cliente, normalmente un usuario que utiliza un navegador web.
  0.2.1. Analiza un archivo de configuración, que contiene un sistema de emparejamientos que definen la ruta que encaja con la petición de la URI.
  0.2.2. Asegura que el usuario que realiza la petición tiene autorización.
  0.2.3. Encarga la filtración y validación de los datos de entrada a un filtro de entrada.
  0.2.4. Encarga el flujo de control y la gestión de errores a un controlador de flujo.
  0.2.5. Encarga la presentación de contenidos a la apropiada vista.
0.3. El modelo representa o encapsula la lógica o el estado de la aplicación.
  0.3.1. Puede ser dividido en dos subsistemas importantes: el estado interno del sistema y la función que se puede ejecutar para cambiar ese estado.
  0.3.2. El controlador de flujo actúa como adaptador entre la petición y el modelo.
  0.3.3. La arquitectura del framework proporciona la ayuda para cualquier intento de acceder al modelo.

0.4. La parte de la vista es responsable de producir la siguiente fase de la interfaz grafica de usuario.
  0.4.1. Proporciona funcionalidades para presentar contenido estático o dinámico.
  0.4.2. Proporciona la ayuda para la internacionalización.
  0.4.3. Presenta posibles errores.

3.3. Clases principales

En esta sección se describen las clases más importantes del framework ZNF. Si deseas una descripción más detallada puedes consultar el API, que se encuentra en el directorio *doc/api* dentro de la aplicación.

3.3.1. Capa del Controlador

ZNF

Proporciona el controlador principal en el patrón de diseño Modelo-Vista-Controlador (MVC) para aplicaciones web que es conocido comúnmente como Modelo 2.

ZNF_Config_AppConfig

Contiene la colección de informaciones estáticas de configuración del módulo. Devuelve la referencia del objeto *ZNF_Config_AppConfig* con un patrón singleton. Analiza y valida el archivo de configuración, por razones de rendimiento implementa un mecanismo de cacheado que serializa el archivo de configuración en una estructura interna.

ZNF_Config_ModulesConfig

Contiene la colección de informaciones estáticas de configuración del módulo. Devuelve la referencia del objeto *ZNF_Config_ModulesConfig* de un módulo especificado con un patrón singleton. Analiza y valida el archivo de configuración, por razones de rendimiento implementa un mecanismo de cacheado que serializa el archivo de configuración en una estructura interna.

ZNF_Action_RequestProcessor

Contiene el flujo lógico de proceso, proporciona un solo punto de control para la seguridad y registra y encapsula los datos entrantes en un *ZNF_Action_ActionForm* utilizable por el modelo MVC *back-end (retroceso-fin)*. Las subclases pueden modificar la petición de proceso pasando por alto el comportamiento de los métodos para los cuales se desea proporcionar una funcionalidad modificada.

ZNF_Action_ActionMapping

Representa las informaciones que el controlador *ZNF_Action_RequestProcessor* conoce sobre la correspondencia. La correspondencia se considera de una petición particular a una instancia de una clase particular *ZNF_Action_Action*. La instancia *ZNF_Action_ActionMapping* usada para seleccionar un *ZNF_Action_Action* particular se pasa a ese *ZNF_Action_Action*, de tal modo que proporciona el acceso a cualquier configuración a medida tomada del archivo de configuración del módulo y se incluye en el objeto *ZNF_Action_ActionMapping*.

ZNF_Action_ActionErrors

Encapsula los mensajes de error que son reportados por el método ZNF_Action_ActionForm→validate(). Cada error individual es descrito por una clave y un mensaje relativo.

ZNF_Action_Action

Es un adaptador entre el contenido de una petición HTTP entrante y la correspondiente lógica de negocio. *ZNF_Action_Action* se debe ejecutar para procesar la petición. El controlador *ZNF_Action_RequestProcessor* selecciona un apropiado *ZNF_Action_Action* para cada petición, crea una instancia y llama al método *ZNF_Action_Action→execute()*. Cuando un *ZNF_Action_Action* es creado el controlador requiere todas las clases para la acción solicitada. Las subclases deben anular el método *ZNF_Action_Action→execute()* para implementar el mecanismo de envío para la acción.

ZNF_Action_ActionForward

Representa la destinación a la cual el controlador *ZNF_Action_RequestProcessor* pudo ser direccionado. La redirección es un resultado de procesar actividades de una clase *ZNF_Action_Action*.

ZNF_Actions_DispatchAction

Es un abstracto *ZNF_Action_Action* que envía a un método público que sea nombrado por el parámetro de la petición cuyo nombre es especificado por la propiedad del parámetro del *ZNF_Action_ActionMapping* correspondiente. Esta acción es útil para los desarrolladores que prefieren combinar muchas acciones similares en una sola clase *ZNF_Actions_DispatchAction*, para simplificar su diseño de la aplicación.

3.3.2. Capa del Modelo

ZNF_Config_DBConfig

Contiene la colección de informaciones estáticas de configuración del módulo. Devuelve la referencia del objeto *ZNF_Config_DBConfig* con un patrón singleton. Analiza y valida el archivo de configuración, por razones de rendimiento implementa un mecanismo de cacheado que serializa el archivo de configuración en una estructura interna.

ZNF_Business_DB

La clase *ZNF_Business_DB* extiende de la clase DB de PEAR. Conecta con la base de datos usando la configuración tomada del archivo de configuración de la base de datos. Devuelve la referencia del objeto *ZNF_Business_DB* con un patrón singleton.

3.3.3. Capa de la Vista

ZNF_Action_ActionForm

Chequea y filtra todas las propiedades que vienen del formulario antes de que el método *ZNF_Action_Action→execute()* sea llamado. Se asocia a unos o más

ZNF_Action_ActionMapping. Primero las características deben ser fijadas, entonces si en el archivo de configuración del módulo el atributo *validate* del elemento *action* es fijado a *true* el método *ZNF_Action_ActionForm→validate()* será llamado por el controlador *ZNF_Action_RequestProcessor*, que da una ocasión de verificar que las propiedades enviadas son correctas y válidas. Si este método encuentra errores, añade mensajes de error a un objeto *ZNF_Action_ActionErrors*. En otro caso, el método *ZNF_Action_ActionForm→validate()* devuelve un objeto *ZNF_Action_ActionErrors* vacío, indicando que todo es aceptable y el método correspondiente *ZNF_Action_Action→execute()* debe ser llamado. Las subclases deben proporcionar los métodos accesores (*getter)* y modificadores (*setter)* apropiados para todas las propiedades que se desean exponer, más la invalidación de los métodos públicos o protegidos para los cuales se desean proporcionar una funcionalidad modificada.

ZNF_Presentation_Translation

Maneja las traducciones. Devuelve secuencias traducidas mediante un índice numérico. Analiza y valida los archivos de la traducción, por razones de rendimiento implementa un mecanismo de cacheado que serializa el archivo de configuración en una estructura interna.

ZNF_Presentation_Template

Extiende la clase Smarty del paquete Smarty. Transforma los datos usando plantillas del paquete o del tema.

ZNF_Presentation_XSLT

Implementa la transformación con *PHP XSLTProcessor*. Transforma los datos XML usando el paquete o el tema XSLT.

3.3.4. Utilidad

ZNF_Util_Utility

Proporciona multitud de métodos útiles para ayudar a implementar diversas funcionalidades.

3.3.5. ¿Clases a usar?

¿Es obligatorio utilizar todas las clases de ZNF para desarrollar una aplicación? La respuesta es NO. Para la capa del controlador se recomienda extender las clases del controlador de ZNF, para las capas del Modelo y la Vista, el ZNF ofrece clases de ayuda, puedes utilizarlas o continuar usando tus tecnologías preferidas.

3.3.6. Variables reservadas

Son variables reservadas las que casen con la máscara $_REQUEST['znf*'], como por ejemplo: $_REQUEST['znfAction'], $_REQUEST['znfModule'], $_SESSION['znf'].

3.4. Instalación

El framework ZNF es independiente del sistema operativo. Requiere una versión del servidor web Apache 2.0.30 o superior y además requiere la versión 5.0 o superior de PHP. El framework ZNF incluye las últimas versiones del paquete PEAR:DB el cual permite realizar operaciones en PHP contra la base de datos sin necesidad de especificar el nombre de del gestor de base de datos usado. Esto permite cambiar el gestor de base de datos sin necesidad de modificar nada en el modelo de datos.

También incluye el motor de plantillas Smarty el cual permite construir páginas HTML dinámicamente si necesidad de utilizar código PHP.

La instalación del framework es muy sencilla. Se puede realizar de dos formas:

- Instalación con el paquete principal: Se descarga del paquete bz2 o el paquete zip y se descomprime en el directorio Document Root indicado en la configuración del servidor web Apache.

- Instalación con el paquete PEAR: Ésta instalación requiere un básica familiaridad con el paquete PEAR y es la manera más fácil y rápida de bajarse y utilizar ZNF. Para instalarlo hay que bajarse el paquete de ZNF "*ZNF-0.7.6.tgz*". Se instala ZNF desde línea de comandos usando PEAR con el siguiente comando:

```
#> pear install ZNF-x.x.x.tgz
```

3.4.1. Estructura de directorios después de la instalación

La estructura de directorios del framework es la siguiente:

```
+--cache-----------+
|
+--config----------+
|           +--znf-app-config.xml
|            +--znf-db-config.xml
|           +--znf-<module>-config.xml
|
+--docs------------+
|           +--api-------------+
|           |
|           +--src-------------+
|           |           +--latex-----------+
|           |            |
|           |           +--uml-------------+
|           |
|           +--AUTHORS
|           +--ChangeLog
|           +--COPYING
|           +--README
|           +--TODO
|           +--znf-x.x.x.pdf
|           +--Makefile
|
+--lang------------+
|           +--<language>.xml
|
+--lib-------------+
|
+--<PackageName>---+
|           |
|           +--docs------------+
|           |
|           +--lang------------+
|           |           +--<language>.xml
|           |
|           +--sql-------------+
|           |           +--<PackageName>.ddl.sql
|           |           +--<PackageName>.dml.sql
|           |           +--<PackageName>.drl.sql
|           |
|           +--templates-------+
|           |           +--<language>------+
|           |                       +--css-------------+
|           |                       +--icons-----------+
```

```
|       |               +--images----------+
|       |               +--js--------------+
|       |               |
|       |               +--<templateName>.tpl
|       |               +--<xsltName>.xsl
|       |
|       +--Action----------+
|       |               +--<ActionClassName>.php
|       |
|       +--Business--------+
|       |               +--<BusinessClassName>.php
|       |
|       +--Presentation----+
|       |               +--<PresentationScriptName>.php
|       |
|       +--<PackageName>.php
|
+--resources-------+
|
+--themes----------+
|       +--default---------+
|       |               +--<language>------+
|       |               |               +--css-------------+
|       |               |               +--icons-----------+
|       |               |               +--images----------+
|       |               |               +--js--------------+
|       |               |               |
|       |               |               +--<templateName>.tpl
|       |               |               +--<xsltName>.xsl
|       |               |
|       |               +--packages--------+
|       |                               |
|       |                               +--<PackageName>---+
|       |                                               +--<language>------+
|       |                                                       +--css-------------+
|       |                                                       +--icons-----------+
|       |                                                       +--images----------+
|       |                                                       +--js--------------+
|       |                                                       |
|       |                                                       +--<templateName>.tpl
|       |                                                               +--<xsltName>.xsl
|       |
|       +--<ThemeName>-----+
|                        +--<language>------+
|                       |               +--css-------------+
|                       |               +--icons-----------+
|                       |               +--images----------+
|                       |               +--js--------------+
```

```
|                       |                 |
|                       |                 +--<templateName>.tpl
|                       |                  +--<xsltName>.xsl
|                       |
|                       +--packages--------+
|                       |
|                       +--<PackageName>---+
|                                          +--<language>------+
|                                          +--css-------------+
|                                          +--icons-----------+
|                                          +--images----------+
|                                          +--js--------------+
|                                          |
|                                          +--<templateName>.tpl
|                                          +--<xsltName>.xsl
|
+--ZNF-------------+
|         +--lang------------+
|         |                  +--en.xml
|         |                  +--it.xml
|         |
|         +--xsd-------------+
|         |                  +--znf-app-config.xsd
|         |                  +--znf-db-config.xsd
|         |                  +--znf-module-config.xsd
|         |                  +--znf-translation.xsd
|         |
|         +--Action----------+
|         |                  +--Action.php
|         |                  +--ActionErrors.php
|         |                  +--ActionForm.php
|         |                  +--ActionForward.php
|         |                  +--ActionMapping.php
|         |                  +--RequestProcessor.php
|         |
|         +--Actions---------+
|         |                  +--DispatchAction.php
|         |
|         +--Business--------+
|         |                  +--DB.php
|         |
|          +--Config---------+
|         |                  +--AppConfig.php
|         |                  +--DBConfig.php
|         |                  +--ModulesConfig.php
|         |
|         +--Presentation----+
|         |                  +--Render.php
```

```
|       |                   +--Template.php
|       |                   +--Translation.php
|       |                   +--XSLT.php
|       |
|       +--Util------------+
|       |                   +--Utility.php
|       |
|       +--ZNF.php
|
+--autopackage.php
+--index.php
+--Makefile
```

A continuación se muestran las carpetas más importantes de nuestra aplicación después de instalar el ZNF:

- **config:** En este directorio están los ficheros de configuración del framework. Hay un fichero de configuración de la aplicación, un fichero de configuración de la base de datos y un fichero de configuración por cada módulo de la aplicación.

- **<PackageName>:** Habrá un directorio por cada módulo de la aplicación. Es la parte principal sobre la cual se desarrolla la aplicación web. Dentro de este directorio se implementan cada una de las capas del modelo-vista-controlador. Las clases que forman parte del modelo de datos se implementan dentro del directorio Business. Las clases que forman parte del controlador se implementan dentro del directorio *Action*. Los ficheros que forman parte de la vista se implementan dentro del directorio *Presentation*. Además para construir las páginas html se utilizan plantillas que se encuentran dentro del directorio *templates*.

- **ZNF:** En este directorio está implementado el framework propiamente dicho. Tiene los mismos subdirectorios que los directorios de los módulos de la aplicación. Implementa las clases del framework, algunas de las cuales es necesario redefinirlas cuando se implementan clases en algunos módulos de la aplicación, sobre todo en las clases que implementan el controlador. Además hay un directorio llamado *util* que contiene algunas utilidades.

- **themes:** En este directorio hay plantillas que se pueden usar para construir todas las páginas html de la vista, tales como las cabeceras o pies de páginas.

- **docs:** Se pueden guardar documentación de la aplicación o por ejemplo un historial sobre el control de versiones (no es obligatorio). También hay documentación sobre el framework ZNF y su API.

3.4.2. Instalación de aplicación de ejemplo

Para tener una mejor comprensión del framework se puede instalar una aplicación de ejemplo disponible en la web, y estudiar su funcionamiento.

Descarga el paquete encargado de las noticias de: http://prdownloads.sourceforge.net/znf/znf-news-0.7.6.tar.bz2?download

Descomprime el paquete en un directorio de la raíz de tu servidor web.

Inicializa una nueva base de datos (substituir <username>, <password>, <host> y <directory> con tus valores.

- Crear la base de datos

  #> mysqladmin create znf

- Conceder privilegios al usuario

  mysql> GRANT ALL PRIVILEGES ON znf.* TO
  <username>@localhost IDENTIFIED BY ' <password>';

- Cargar las tablas del archivo DDL en el directorio sql de los paquetes News y User.

  $> mysql -u<username> -p<password> znf <
  <path>/News/sql/News.ddl.sql
  <path>/News/sql/News.dml.sql
  $> mysql -u<username> -p<password> znf <
  <path>/User/sql/User.ddl.sql
  <path>/User/sql/User.dml.sql

- Poner la información de autentificación en el archivo de configuración de la base de datos, znf-db-config.xml.

- Abrir la URI http://<host>/<directory>/ en tu navegador, ZNF debe mostrar el sistema encargado de las noticias, es decir, la aplicación ejemplo. Para loguearse en el área de administración se debe usar como username/password lo siguiente: admin/admin.

3.5. Uso

En este capítulo describimos cómo utilizar las características principales del framework.

3.5.1. Configuración

ZNF utiliza por lo menos dos archivos de configuración XML situados en el directorio *config*: un archivo de configuración de la aplicación, nombrado *znf-app-config.xml*, y uno para la configuración de cada módulo extra, nombrados típicamente con el *znf-<modulename>-config.xml*.

El archivo de configuración de la aplicación definía particularmente los valores prefijados que la aplicación tiene que utilizar y los módulos que están disponibles. El archivo de configuración de cada módulo inicializaba los recursos del framework, como *ZNF_Action_ActionForm* para recoger la entrada del cliente, *ZNF_Action_ActionMapping* para direccionar la entrada a las acciones del lado del servidor, y *ZNF_Action_ActionForward* para seleccionar y hacer las páginas de salida. Con estos archivos de configuración es posible manejar la aplicación sin la modificación del código de las clases.

Si deseas utilizar las clases de la capa del modelo, ZNF utiliza otro archivo de configuración para la base de datos, nombrado *znf-db-config.xml*, usado para conectar con una base de datos relacional.

Archivo de configuración de la aplicación

El elemento raíz del archivo de configuración de la aplicación debe ser *znf-app-config*, los posibles elementos hijos son *app-settings* y *modules*. Para los posibles módulos los elementos hijo son *module*.

Atributos del elemento app-settings

lang: Secuencia de dos caracteres usada para especificar el lenguaje por defecto. El lenguaje especificado se debe configurar para todos los paquetes y temas usados en la aplicación.

langAutodetect: Valor booleano, usado para solicitar la auto detección del lenguaje desde los ajustes del navegador del usuario.

locale: Secuencia de 2 caracteres + _ + 2 caracteres, usada para especificar la localización por defecto.

theme: Secuencia, usada para especificar el tema por defecto. El tema especificado debe estar presente en el directorio *theme*.

sessionId: Secuencia, usada para especificar el identificador de sesión.

Atributos del elemento module

default: Valor booleano, usado para seleccionar este módulo como modulo por defecto. Las acciones pertenecientes a ese módulo pueden ser seleccionadas sin especificar el módulo en la URI. La aplicación puede tener solamente un módulo por defecto. El atributo es opcional, y por defecto es falso.

name: Secuencia (sin _), usada para especificar un nombre único para el módulo. El nombre especificado se puede utilizar en la URI como un valor de la variable *GET* del *znfModule* para hacer una selección de una acción del módulo, o alternativamente como valor de la variable *GET* del *znfChangeModule* para hace una selección de un modulo persistente.

processor: Secuencia, usada para especificar el nombre completo de la clase del procesador de peticiones modificado para requisitos particulares. La clase especificada debe extender de *ZNF_Action_RequestProcessor* y será cargado en lugar de la clase del procesador de peticiones por defecto. Es útil cuando es necesario un flujo lógico de proceso modificado para requisitos particulares.

Atributos del elemento auth-config

path: Secuencia, usada para especificar un nombre de acción. La acción especificada será invocada en caso de que se encuentre la restricción de la autentificación o de la autorización. Es útil para redireccionar automáticamente al formulario de logueo.

Archivo de configuración de nuestra aplicación (znf-app-config.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<znf-app-config xmlns="http://znf.zeronotice.com/ZNF/znf-app-
config" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://znf.zeronotice.com/ZNF/xsd/znf-app-
config.xsd">
  <app-settings
    lang="es"
    locale="es_ES"
    theme="default"
    sessionId="ChasquiSessionID"/>
  <modules>
    <module
      default="true"
      name="objetoVirtual"
      config="znf-objetoVirtual-config.xml"/>
  </modules>
</znf-app-config>
```

Archivo de configuración del módulo

El elemento raíz del archivo de configuración del módulo debe ser *znf-module-config*, *global-forwards*, *action-mappings* y *form-beans*. Para *global-forwards* los posibles elementos hijo son *forwards*, para los *action-mappings* son *action* y para los *form-beans* son *form-bean*.

Atributos del elemento action

default: Valor booleano, usado para seleccionar esta acción como la acción por defecto para el módulo. La acción por defecto será invocada cuando no se especifica ninguna otra acción dentro del URI. El módulo puede tener solamente una acción por defecto. El atributo es opcional y por defecto su valor es falso.

forward: Secuencia, usada para especificar una ruta relativa de la aplicación para un script para la vista o alternativamente un nombre de una acción. En el primer caso el flujo de control será remitido al script de vista especificado en vez de cargar la clase de una acción, en el segundo caso el flujo de control será redireccionado a la acción especificada, así que la acción en sí misma es utilizada solamente como alias para la acción especificada. Los datos de entrada serán

filtrados y validados antes del *forward*. Los atributos *forward* y *type* son mutuamente exclusivos y exactamente uno debe ser definido.

input: Secuencia, usada para especificar un nombre de acción. El flujo de control retornara a la acción especificada si se encuentra un error de validación. Es posible referirse a acciones de distintos módulos usando la sintaxis *<modulename>//<actionname>*.

name: Secuencia, usada para especificar un nombre de *form-bean*. El *form-bean* especificado será utilizado para procesar, y eventualmente validar datos de entrada. El nombre especificado tiene que coincidir con el nombre de un elemento *form-bean* definido. El atributo es opcional.

nextPath: Secuencia, usada para especificar un nombre de acción. El nombre especificado será utilizado para actualizar automáticamente referencias en los scripts de vista, así que el flujo de control es completamente configurable en el archivo de configuración.

path: Secuencia (sin _), usada para especificar un nombre único para la acción. El nombre especificado ha sido utilizado en la URI como un valor de la variable *GET* del *znfAction* para hacer una selección de una acción.

parameter: Secuencia, usada como un parámetro de configuración de propósito general. El parámetro especificado se puede utilizar para pasar la información adicional a la clase de la acción cargada. El valor será ignorado por el procesador de peticiones y almacenado simplemente en la variable superglobal $_REQUEST.

roles: Secuencia (separada por ,), usada para especificar una lista de nombres que cumplen papeles de seguridad. Estas tareas de seguridad especificadas permitirán invocar esta acción. El procesador de peticiones verifica que el usuario tiene por lo menos uno de los papeles especificados antes de cargar una acción.

scope: Secuencia (petición o sesión), usada para especificar en qué ámbito de petición o de sesión tiene que estar situado el *form-bean*. El atributo puede ser definido solamente si el atributo *name* está presente.

type: Secuencia, usada para especificar el nombre completo de la clase de la acción. La clase especificada debe extender de *ZNF_Action_Action*.

Los atributos forward y *type* son mutuamente exclusivos y exactamente uno debe ser definido.

validate: Valor booleano, usado para solicitar la llamada del método *validate()* del *form-bean* especificado en el atributo *name*. El atributo es opcional y por defecto tiene el valor falso.

Atributos del elemento forward

name: Secuencia, usada para especificar un nombre único para el *forward*. El atributo es requerido.

path: Secuencia, usada para especificar una URI o nombre de acción. En el primer caso el flujo de control será remitido o redireccionado a la URI especificada, en el segundo caso la acción especificada será invocada. Es posible referirse a acciones de diversos módulos usando la sintaxis <*modulename*>//<*actionname*>. El atributo es requerido.

redirect: Valor booleano, usado para solicitar que el procesador de peticiones realice una redireccion en lugar de un *forward*. El atributo es opcional y por defecto tiene el valor falso.

Atributos del elemento form-bean

name: Secuencia (sin _), usada para especificar un nombre único para el *form-bean*. El atributo es requerido.

type: Secuencia, usada para especificar un nombre completo de la clase. La clase especificada debe extender de *ZNF_Action_ActionForm*. El atributo es requerido.

Archivo de configuración de nuestro módulo (znf-objetoVirtual-config.xml) (solo la parte relacionada con los usuarios, para no ocupar mucho espacio)

```
<global-forwards>
      <forward name="failure" path="resources/Failure.php" />
</global-forwards>
<action-mappings>
      <action path="login"
            forward="Usuario/Presentation/Login.php"
            nextPath="loginSubmit" />
      <action path="loginSubmit" type="Usuario_Action_Login"
            name="loginForm" input="login" validate="true">
            <forward name="loginSuccess" path="loginSuccess" />
            <forward name="loginSuccessXml"
                  path="loginSuccessXml" />
            <forward name="failure" path="login" />
      </action>
      <action path="loginSuccess"
            forward="Usuario/Presentation/LoginSuccess.php" />
      <action path="loginSuccessXml"
            forward="Usuario/Presentation/LoginSuccessXml.php" />
      <action path="logout" type="Usuario_Action_Logout">
            <forward name="success" path="welcome"
                  redirect="true"/>
      </action>
</action-mappings>
<form-beans>
      <form-bean name="loginForm"
            type="Usuario_Action_LoginForm"/>
      </form-beans>
</znf-module-config>
```

Archivo de configuración de la base de datos

El elemento raíz del archivo de configuración de la base de datos debe ser *znf-db-config*, el posible elemento hijo es *db*.

Atributos del elemento db

default: Valor booleano, usado para seleccionar esta base de datos como la utilizada por la aplicación. La aplicación puede tener solamente una base de datos por defecto. El atributo es opcional y por defecto el valor es falso.

dbms: Secuencia, usada para seleccionar el tipo de DBMS (Sistemas Manejadores de BD). El atributo es opcional y por defecto el valor es mysql.

username: Secuencia, usada para especificar el nombre de usuario de la base de datos. El atributo es requerido.

password: Secuencia, usada para especificar la contraseña de la base de datos. El atributo es requerido.

hostname: Secuencia, usada para especificar el *hostname* de la base de datos. El atributo es opcional y por defecto el valor es *localhost*.

dbname: Secuencia, usada para especificar el nombre de la base de datos. El atributo es requerido.

Archivo de configuración de nuestra base de datos (znf-db-config.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<znf-db-config xmlns="http://znf.zeronotice.com/ZNF/znf-db-
config"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://znf.zeronotice.com/ZNF/xsd/znf-
db-config.xsd">
      <db default="true"
            dbms="mysql"
            username="root"
            password="root"
            hostname="localhost"
            dbname="chasqui2pr" />
</znf-db-config>
```

3.5.2. Localización

El ZNF tiene ayuda incorporada para la localización, utiliza archivos XML para cargar secuencias traducidas. El comportamiento es muy simple: en la raíz de la aplicación y en cada paquete (módulo) tienes que crear un directorio llamado *lang*, en el cual tienes que crear por lo menos un archivo para la lengua por defecto, y un archivo para cada otra lengua que deseas para tu aplicación.

El directorio *lang* puesto en la raíz debe contener traducciones de texto genéricos de la aplicación, por ejemplo la palabra inglesa *"welcome"*. Por el

contrario, los directorios *lang* colocados en cada paquete deben contener la traducción del texto específico para cada módulo, por ejemplo la palabra inglesa *"login"* en el paquete *User*.

El archivo de la lengua se debe nombrar con la abreviatura de 2 dígitos de dicha lengua, por ejemplo *"en"* para el inglés e *"it"* para el italiano.

Archivo de la traducción

El elemento raíz del documento XML debe ser *znf-translation*, el posible elemento hijo es ítem.

Atributos del elemento ítem

code: Secuencia alfanumérica única, usada para referirse al elemento de traducción en la aplicación. El atributo es requerido.

text: El texto traducido en la lengua especificada.

Observar que se pueden agregar archivos de lengua con no todos los elementos traducidos, el framework cargara automáticamente el texto sin traducir de la lengua por defecto para los elementos que no estén traducidos.

Por defecto el framework es fijado a *"en"* para la lengua y a *"en_US"* para la localización. Si deseas cambiar el valor prefijado tienes que cambiar los valores del atributo *app-settings* en el archivo de configuración.

Puedes especificar si el *ZNF_Presentation_Translation* debe cargar las traducciones de un paquete específico o los globales, usadas normalmente en la aplicación entera. El primer parámetro del constructor de *ZNF_Presentation_Translation* te deja especificar el paquete a utilizar. Si pasas una secuencia como dicho parámetro el objeto *ZNF_Presentation_Translation* cargará los archivos de las traducciones del directorio *lang* del paquete especificado. Si no se pasa ninguna secuencia a este parámetro el objeto *ZNF_Presentation_Translation* cargará los archivos de las traducciones del directorio *lang* de la raíz de la aplicación.

Archivo para el lenguaje por defecto, el español (es.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<znf-translation lang="es"
xmlns="http://znf.zeronotice.com/ZNF/znf-translation"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://znf.zeronotice.com/ZNF/xsd/znf-
translation.xsd">

  <item code="welcome" text="Bienvenido a Chasqui"/>
  <item code="failure" text="Ha ocurrido un error"/>
  <item code="contacts" text="Contactar"/>
  <item code="mailingLists" text="Lista de correo"/>
  <item code="login" text="Iniciar sesion"/>
  <item code="loginSuccess"
      text="Sessión iniciada correctamente"/>
</znf-translation>
```

3.5.3. Presentación

El ZNF proporciona la clase *ZNF_Presentation_Smarty*, basada en el motor de plantillas Smarty, y la clase *ZNF_Presentation_XSLT*, que maneja transformaciones XSL, para que un desarrollador de PHP pueda utilizarlas para la presentación de contenidos. Observar que todas las clases cuya función es construir las páginas para mostrarlas al usuario deben extender de la clase abstracta *ZNF_Presentation_Render*.

Puedes encontrar la documentación del motor de plantillas Smarty en:

http://smarty.php.net/docs.php

y la documentación de las transformaciones XSL en:

http://www.w3.org/TR/xslt

Constantes

Observar que la variable *$znf* está reservada en ambas plantillas, Smarty y XSL. Esta variable contiene los siguientes atributos, que puedes utilizar para simplificar y acelerar la creación de la vista:

- *znf.baseHref*. Encadenamiento del *hostname* y de la ruta relativa del directorio público de scripts, usado como prefijo en hipervínculos para referirse a URIs correctas.

- *znf.actionIndex*. Nombre del atributo *GET* usado para seleccionar la acción a ejecutar.

- *znf.moduleIndex*. Nombre del atributo *GET* usado para seleccionar la configuración del modulo.

- *znf.themePath*. Ruta relativa del tema actual, usada como prefijo en hipervínculos (css, Javascript, imágenes…) para referirse a rutas correctas.

- *znf.packagePath*. Ruta relativa del paquete en uso, usada como prefijo en hipervínculos (css, Javascript, imágenes…) para referirse a rutas correctas.

- *znf.path*. Ruta de la actual acción.

- *znf.nextPath*. Ruta de la siguiente acción a solicitar.

- *znf.credits*. Créditos de ZNF.

Mecanismo de carga de plantilla

El ZNF soporta dos clases distintas de plantillas:

- Plantilla de paquete: asociada al paquete, entregada con el paquete para proporcionar un componente de trabajo listo para utilizar, contiene generalmente las pautas para el tratamiento y presentación de datos.

- Plantilla de tema: asociada a la aplicación en general, usada para construir la estructura y aspecto de la aplicación.

Si deseas cambiar el diseño de una plantilla de un paquete puedes poner una nueva, con el mismo nombre que la original, en el directorio *themes/packages* especificando el nombre del paquete y de la lengua.

Puedes especificar si el *ZNF_Presentation_Render* debe cargar la plantilla de un paquete específico o la de uno de los temas. El primer parámetro del constructor de *ZNF_Presentation_Render* te permite especificar el paquete a utilizar. Si pasas una secuencia a este parámetro el objeto *ZNF_Presentation_Render* cargará los archivos de la plantilla del directorio *template* del paquete especificado. Si no se pasa

ninguna secuencia a este parámetro el objeto *ZNF_Presentation_Render* cargará los archivos de la plantilla del tema de la aplicación.

3.5.4. Acciones remotas

El ZNF tiene soporte incorporado para acciones remotas. Esta ayuda sigue siendo muy experimental por lo que este apartado es meramente informativo sobre la posibilidad de utilizar dichas acciones en un futuro cuando estén bien documentadas y probadas. La ejecución de acciones remotas usando las facilidades del framework provoca que el flujo de proceso se dividida en 2 subflujos, uno para el cliente remoto (la aplicación con la cual el usuario interactúa usando un navegador) y otro para el servidor remoto (la ejecución de la aplicación ZNF en un host remoto que ejecuta acciones como servicios web).

El framework oculta al desarrollador la complejidad de la comunicación remota, dejando la configuración de la ejecución remota de la acción realmente simple. El cliente remoto envía la ruta que corresponde a la acción que se ejecutará y el estado serializado de la aplicación a un servidor remoto donde corre el ZNF y recibe una secuencia usada para procesar el reenvío. El servidor remoto recibe la ruta que corresponde a la acción que se ejecutará y el estado serializado de la aplicación del cliente remoto donde corre el ZNF y envía una secuencia usada para procesar el reenvío. La comunicación se hace con el protocolo SOAP.

Si deseas intentar el soporte remoto tienes que utilizar el paquete *Remote*, creando una nueva configuración del módulo tanto en el lado del cliente como en el lado del servidor.

El cliente tiene que configurar las acciones que se ejecutarán remotamente con el atributo *path* fijado a la ruta usada en el servidor remoto para la acción deseada y el atributo *parameter* fijado a la URI del host remoto, como *http://<remotehost>/<directory>*.

El servidor tiene que configurar una acción con el atributo *path* fijado a *webService* y el atributo *default* fijado a *true*. Esta acción es usada por el procesador de peticiones personalizado *Request_Action_RequestProcessorServer* para interceptar todas las peticiones SOAP que llegan desde el cliente.

4. Nuevas funcionalidades a implementar

El nuevo sistema Chasqui desarrollado usando el framework ZNF no disponía de todas las funcionalidades del sistema anterior. Principalmente se echaba en falta la ausencia de un acceso clasificado a la base de datos. La navegación estática del sistema anterior, presentaba una serie de problemas de actualización a la hora de introducir objetos con nuevos metadatos de clasificación. Por tanto nuestra tarea principal consistirá en elaborar, dentro del nuevo sistema, un acceso clasificado a la base de datos dinámico, que permita reflejar inmediatamente en la navegación los cambios realizados en la base de datos.

Otra funcionalidad que no estaba desarrollada es la inserción de objetos virtuales, que tan solo permitía introducir el usuario y la descripción del nuevo objeto. Nosotros hemos modificado esta inserción para poder añadir objetos con un modelo de fichas similar al que presentaba la antigua aplicación. Nos interesa especialmente la ficha que contendrá los metadatos de la clasificación del objeto, que será la que luego permita su aparición en el acceso clasificado a la base de datos.

4.1. Acceso clasificado a la base de datos

La navegación por clasificación que proponía el sistema antiguo, se basaba en una serie de accesos a seis de tablas fijas, que se creaban en la base de datos a partir de otras tablas (principalmente de **metadatos**). Cada tabla se encargaba de guardar los datos correspondientes a la navegación de los objetos con igual metadato **Sección/Sección**, cuyo valor daba también el nombre a la tabla. Estas tablas eran: **arqueología**, **reproducciones**, **etnología**, **agha**, **material_docente** y **material_documental**. El nombre de los campos de las mismas hacía referencia a los metadatos de clasificación de los objetos, pero sin tener en cuenta la ruta dentro de la navegación. Por ejemplo, un nombre que aparece en los campos de casi todas las tablas es **yacimiento**, pero no indica su secuencia dentro de la navegación, que es **Sección/País/Región/Yacimiento**. Esto se hizo así pensando en la construcción de un acceso clasificado a la base de datos estático, que no reflejase los cambios que se pudieran producir en la estructura interna de cualquiera de las seis tablas de navegación. Por tanto el diseño anterior no mostraría la inserción de una nueva subsección, ni contemplaría la creación de una nueva tabla de navegación, a causa de que se hubiera introducido un nuevo objeto con una sección distinta a las seis que se representan. Es más, sólo se encargaba de mostrar ciertas columnas de las tablas, dejando muchos metadatos de clasificación inservibles ya que nunca podían ser accedidos.

Para la nueva navegación, al principio pensamos en crear una sola tabla que almacenase los metadatos de clasificación de todos los objetos, construyéndola mediante un script PHP a partir de la tabla **metadatos**. En el nombre de los campos de esta nueva tabla, estaría reflejado el título de los metadatos de clasificación

recogidos, indicando su ruta dentro de la navegación. Pero nos encontramos con un problema, y es que al agregar un campo con un nuevo título (por ejemplo el mencionado antes: **Sección/País/Región/Yacimiento**), el servidor de MySQL no permitía su inserción a causa de contener caracteres del tipo "/". La solución que empleamos consistió en cambiar el formato de las cadenas que se leían de la tabla **metadatos**. Para los nombres de campo se optó por cambiar los caracteres "/" a "X", dejar todos los valores de la ruta en minúsculas, suprimir las tildes, y sustituir los espacios " " por "_". Para los valores correspondientes a las filas y columnas escogimos establecerlos todos a mayúsculas, suprimir igualmente las tildes, y cambiar los espacios " " por "_". Esto fue decidido así porque a la hora de realizar consultas a la base de datos MySQL usando PHP, se nos presentó otro problema. Consiste en que PHP distingue mayúsculas de minúsculas, y vocales sin tilde de vocales con tilde, y MySQL no, entonces para evitar accesos y resultados erróneos, decidimos establecer este formato explicado. Por ejemplo, si realizábamos una consulta sobre el **area_cultural** con el valor *ANDINA*, el servidor de MySQL nos devolvía las filas de los objetos con los valores *ANDINA* y *Andina*, y PHP los distinguía en los resultados como si perteneciesen a distintas áreas culturales, cuando en el fondo son de la misma. Según lo dicho, la cadena correspondiente al metadato **yacimiento**, que es **Sección/País/Región/Yacimiento**, quedaría así: **seccionXpaisXregionXyacimiento**. Y un valor correspondiente a ese metadato como es *Loma Alta*, quedaría transformado en *LOMA_ALTA*. La única cadena que no ha sido necesario modificar es *~Sin asignar*, que es siempre igual en todas sus apariciones en la base de datos, ya que según la estructura de las tablas de la misma, es el valor que se toma por defecto cuando no se le asigna ninguno al registro en cuestión.

La nueva tabla funcionaba muy bien al realizar consultas sobre ella, pero se desperdiciaba gran parte de su contenido. La mayoría de las secciones poseen algunos metadatos de clasificación específicos de la sección, y el hecho de almacenar todos los objetos juntos, obligaba a crear campos en la tabla que eran usados por muy pocos de ellos, ya que el valor de ese campo en los objetos de las demás secciones era *~Sin asignar*. Este hecho también complicaba la tarea de conseguir una navegación dinámica, ya que nosotros mostramos las opciones disponibles recogiendo el nombre de los campos de la tabla. Si por ejemplo deseábamos mostrar las opciones correspondientes a la sección *ETNOLOGIA* (que son **area_cultural** y **pais**), obteníamos muchas opciones que no pertenecían a la misma, como **privada**, **tema**, **tipo**, **destinatario**, además de las correctas.

Por tanto decidimos al final volver a usar una tabla individual para cada sección, pero a diferencia de las tablas fijas que había anteriormente, los nombres de los campos de estas nuevas tablas denotarían a los metadatos, y también a su ruta dentro de la navegación. Además, estas tablas son creadas de forma que sólo se añaden campos, si como mínimo algún objeto de la sección que da nombre a la tabla, tiene valor distinto de *~Sin asignar* en el campo añadido. Así es que todas las tablas tienen un conjunto de campos particular en los que no existe ninguno que contenga todas sus casillas con el valor *~Sin asignar*.

También creamos una tabla llamada **secciones**, que sencillamente contiene las distintas secciones que hay actualmente en la base de datos, para poder mostrarlas inmediatamente cuando se pincha sobre el enlace *Navegación por clasificación*. Al añadir un objeto con una sección distinta a las que aparecen en esta tabla, se inserta en la misma la nueva sección, y se procede a crear una tabla con su nombre, añadiendo los metadatos de clasificación correspondientes al nuevo objeto en ella. Si se va a añadir un nuevo objeto con una sección ya conocida, pero contiene metadatos de clasificación que no aparecen en los campos de la tabla de su sección, se crean estos nuevos campos añadiendo en ellos los valores pertenecientes al objeto. Evidentemente, también se añadirán los valores de los metadatos de clasificación del objeto, que ya eran conocidos anteriormente en la tabla.

Para elaborar una navegación dinámica nos hemos basado en el análisis de los nombres de los campos de las tablas de clasificación. Excepto el primer paso de la navegación (que es en el que se escoge la sección por la que se va a navegar), todos los demás pasos usan los nombres de los campos de la tabla en cuestión, para mostrar las opciones correspondientes al siguiente paso. Como ya hemos comentado antes, estos nombres son de la forma **seccionXsubseccion1Xsubseccion2X...**, donde cada subsección hace referencia a un nuevo avance dentro de la navegación.

El método que utilizamos para obtener las opciones a mostrar en un siguiente paso, consta de dos partes. La primera se encarga de recopilar todas las subsecciones de la tabla pertenecientes a un mismo paso, pero sólo elige las que representan la continuidad de la rama elegida en la navegación. Es decir, si vamos escoger las opciones del segundo paso, y estamos en la tabla **arqueologia**, se seleccionarían como valores a mostrar **area_cultural**, **pais** y **privada**, haciendo referencia a los metadatos **seccionXarea_cultural**, **seccionXpais**, y **seccionXprivada** respectivamente. Otros metadatos como **seccionXarea_culturalXsubarea_culturalXcultura** o **seccionXpaisXregion**, quedarían excluidos al pertenecer a otros niveles de la navegación, y ser la continuación además, de alguno de los caminos a escoger en el segundo paso. Pero nos hemos encontrado casos dentro de las tablas de navegación en donde no se respeta la continuidad del camino. También en la tabla **arqueología** se da el nombre de dos metadatos que muestran este caso. Son **seccionXtemaXsituacion_museo** y **seccionXtemaXconferencias**, los cuales no disponen del metadato **seccionXtema** (en la tabla **arqueologia**), que haga de puente en el camino de la navegación. Estos metadatos quedarían por tanto inaccesibles. Al principio pensamos en añadir todas las subsecciones pertenecientes a un mismo paso en la navegación, sin distinguir las que tuviesen continuidad o no. Pero los accesos a la tabla fallaban, a causa de que se buscaba el metadato que hacía de puente y no se encontraba. Por esta razón nos vimos obligados a desarrollar la segunda parte del método. Lo que realmente hace esta parte es obtener los metadatos que han quedado perdidos en el camino elegido, y ofrecerlos para su acceso directo en el siguiente paso, independientemente del nivel en el que nos encontremos, o se encuentren ellos. Volviendo al ejemplo anterior, los metadatos **seccionXtemaXsituacion_museo** y **seccionXtemaXconferencias** que antes no aparecerían, ahora lo harían junto con **area_cultural**, **pais** y **privada**, pero

se mostrarían como **seccion/tema/situacion_museo** y **seccion/tema/conferencias**, sustituyendo las "X" por "/" para mejor presentación. Mediante esta notación, los metadatos que rompan la continuidad de la navegación podrán ser detectados, ya que seguramente se deben a errores en la inserción de las características de clasificación de algunos objetos.

Seguidamente pasamos a mostrar los fragmentos más importantes del código comentado de este método en PHP. Pero antes hay que dejar claro que cuando estamos en un paso de navegación posterior al segundo, sólo se van seleccionando los campos que comparten las mismas subsecciones anteriormente elegidas, para poder continuar por el camino escogido. El código de esta distinción no se muestra para simplificar un poco la explicación de la estructura del método. Hay algunos detalles más que tampoco se muestran por la misma razón, pero se puede consultar la función completa en el fichero *ObjetoVirtualDAO.php* de la carpeta *Business*, ya que aparece con el nombre *buscarOpciones*. Comenzamos mostrando la primera parte del método:

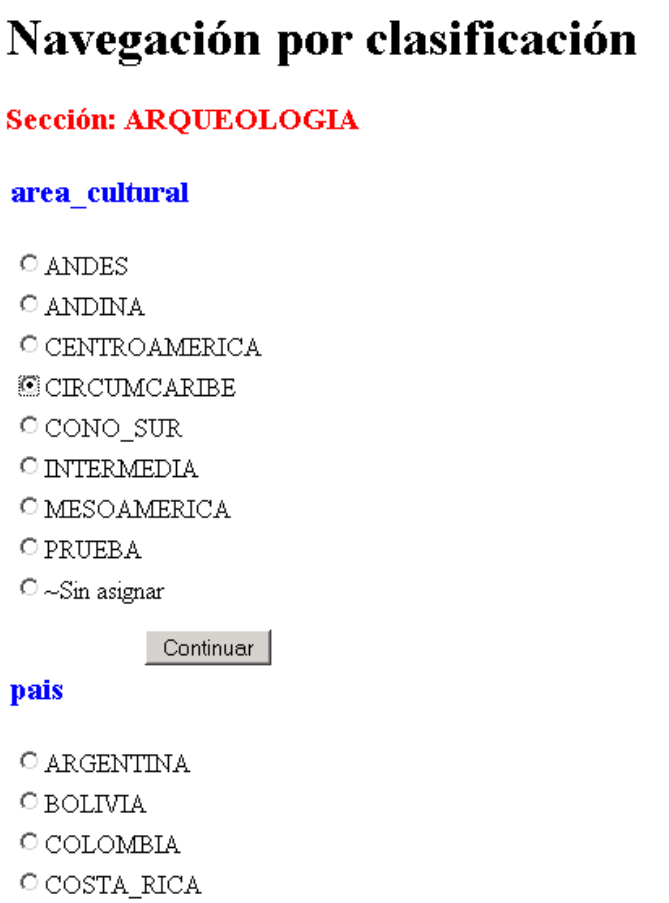
```
/*Guardamos la información de la tabla con nombre $seccion en
$infoTab. Utilizaremos la variable como listado con los nombres de
los campos de la tabla, aunque contiene mucha más información que
ahora no nos va a interesar.*/

$infoTab=$this->_db->tableInfo($seccion);

/*Iniciamos el contador a 2 para que se salte los dos primeros
campos de la tabla, que son id e idov. No nos interesan ahora.*/

$i=2;

/*Array en el que iremos recogiendo los metadatos válidos.*/

$camposValidos=null;

/*Índice para el array.*/

$arrayIndex=0;

/*Bucle encargado de recorrer el listado con los nombres de los
campos. No termina hasta que no ha analizado todos.*/

while ($infoTab[$i]['name']!=""){

      /*Dividimos cada metadato en "$numCad" cadenas, usando la "X"
      como delimitador de las particiones.*/

      $arrayCampo=explode("X",$infoTab[$i]['name'],$numCad);

      /*Si la última partición obtenida es igual a la cadena vacía,
      significa que el metadato es apto para ser elegido, ya que
      demuestra así que se encuentra dentro del nivel adecuado en la
      navegación. Lo añadimos al array junto con el valor que se
      visualizará en el navegador.*/

      if ($arrayCampo[$numCad-1]==""){
            /*Valor que se mostrará por pantalla.*/
            $camposValidos[$arrayIndex][0]=$arrayCampo[$numCad-2];
            /*Valor que servirá para realizar las consultas.*/
            $camposValidos[$arrayIndex][1]=$infoTab[$i]['name'];
            $arrayIndex++; /*Incrementamos el índice.*/
      } /*Fin if.*/

      $i++; /*Incrementamos el contador.*/

} /*Fin while.*/
```

Mostramos ahora la segunda parte del método:

```
/*Volvemos a inicializar el contador a 2 por las mismas razones
expuestas en la primera parte.*/
$i=2;
/* Bucle encargado de recorrer el listado con los nombres de los
campos. No termina hasta que no ha analizado todos.*/
while ($infoTab[$i]['name']!=""){
      /*Dividimos cada metadato en "$numCad" cadenas, usando la "X"
      como delimitador de las particiones.*/
      $arrayCampo=explode("X",$infoTab[$i]['name'],$numCad);
      /*Declaramos un booleano que nos indicará que nuevos metadatos
      se añadirán a la solución. Se inicializa a FALSE.*/
      $salta=FALSE;
      $j=0; /*Contador del siguiente bucle.*/
      /*Bucle que señala (poniendo $salta a TRUE), los metadatos que
      no deben ser añadidos a causa de que comparten la misma
      subsección (en el paso en cuestión), con alguno de los
      metadatos insertados en la primera parte del método. A éstos
      se podrá acceder en los siguientes pasos de la navegación.*/
      while ((!$salta) AND ($camposValidos[$j][0]!="")){
            if ($arrayCampo[$numCad-2]==$camposValidos[$j][0]){
                  $salta=TRUE;
            }
            $j++;
      } /*Fin del while anidado.*/
      /*Si alguno de los metadatos no es señalado, es porque su ruta
      es distinta a la de los ya añadidos, y carece del metadato
      puente explicado anteriormente, por tanto se debe insertar
      para su posterior acceso directo. En el valor a mostrar por
      pantalla se representa su ruta completa, sustituyendo los
      delimitadores "X" por "/".*/
      if (!$salta){
            $k=0;
            $ruta=strtr($infoTab[$i]['name'],"X","/");
            $camposValidos[$arrayIndex][0]=$ruta;
            $camposValidos[$arrayIndex][1]=$infoTab[$i]['name'];
            $arrayIndex++;
      }
      $i++;
}
return $camposValidos;
```

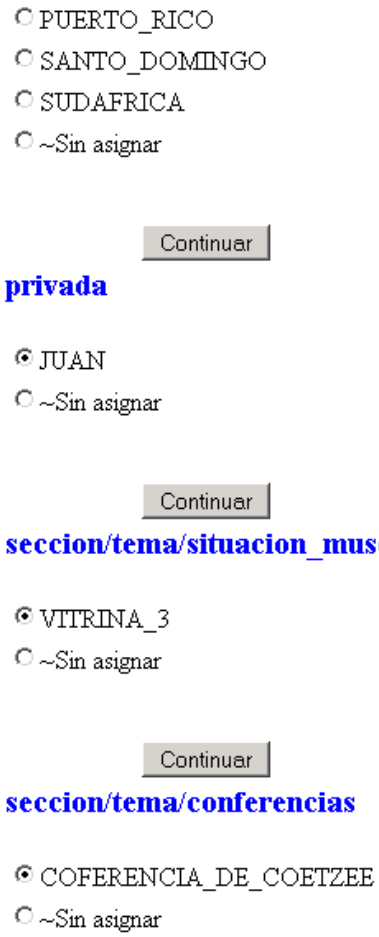
En cada paso de la navegación no sólo se muestran los metadatos a elegir, sino que también se ofrecen los valores disponibles correspondientes a esos metadatos. Al seleccionar uno de estos valores, se almacena implícitamente en una

variable el siguiente paso dentro del camino elegido. A continuación mostramos una imagen que nos ayudará a aclarar todo esto:



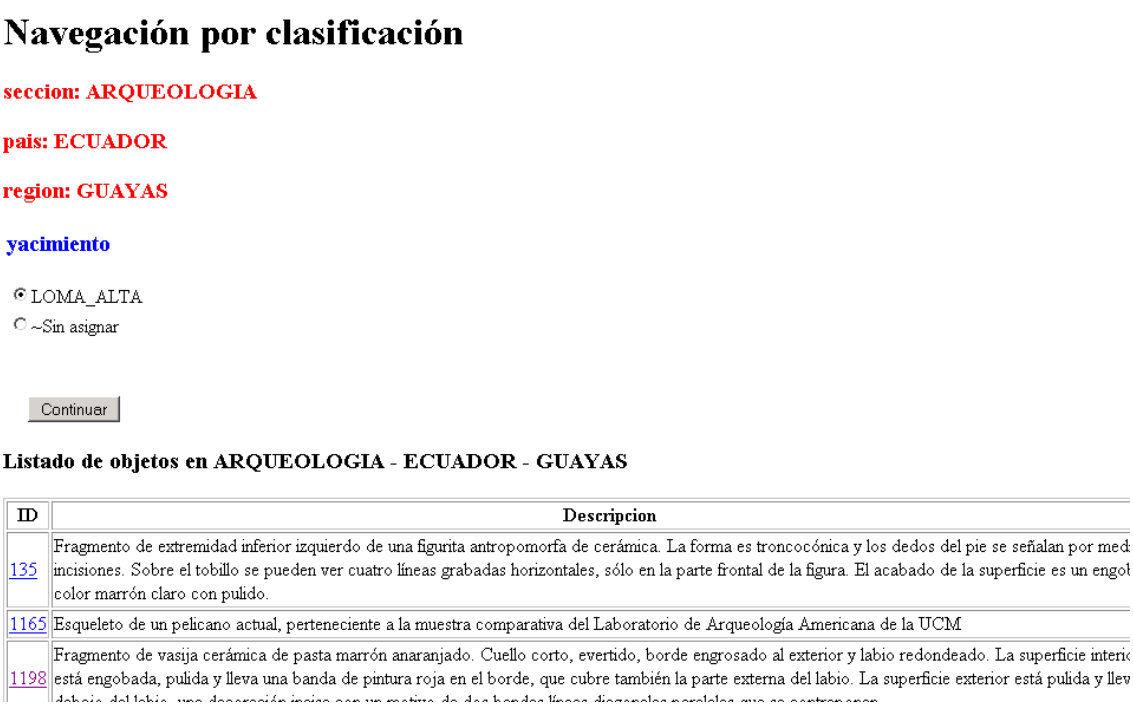
Nos encontramos en el segundo paso de la navegación, habiendo escogido en el primero la sección *ARQUEOLOGÍA*. En la imagen aparecen los metadatos **area_cultural** y **pais**, con algunos de sus respectivos valores. Si se selecciona la opción *CIRCUMCARIBE*, en el siguiente paso de la navegación ya sabremos que el metadato elegido es **area_cultural** con el valor *CIRCUMCARIBE*. Por tanto podremos continuar navegando por las subsecciones de **area_cultural**, teniendo en cuenta el valor que hemos escogido para este metadato.

Si quedaban dudas de cómo se realiza la presentación de los metadatos perdidos en el camino de la navegación, pasamos a mostrar una imagen que seguro las podrá resolver:



Al igual que en el ejemplo anterior, estamos en el segundo paso de la navegación, dentro de la sección *ARQUEOLOGÍA*. Los valores que se reflejan al principio (*PUERTO_RICO*, *SANTO_DOMINGO*, *SUDÁFRICA* y *~Sin asignar*), pertenecen al metadato **seccionXpais**, que aparece como **pais** según indica la imagen anterior a la que estamos comentando. De hecho esta imagen es una continuación de la anterior. Los metadatos que no respetan la continuidad de la navegación son **seccion/tema/situacion_museo** y **seccion/tema/conferencias**, acompañados de sus respectivos valores. Se facilitan con su ruta completa para que se sepa donde está el hueco en el camino elegido de la navegación, que en este caso sería el metadato **seccionXtema**.

En cada paso de navegación, también son mostrados los objetos que coinciden con los metadatos de clasificación seleccionados por el usuario. Dichos objetos podrán ser accedidos directamente pulsando sobre su índice. Se presentan junto con los valores de los metadatos del camino seguido, el cual se muestra completamente en la parte superior, antes de las opciones del siguiente paso. Pasamos a enseñar un ejemplo de todo esto:



Navegación por clasificación

seccion: ARQUEOLOGIA

pais: ECUADOR

region: GUAYAS

yacimiento

- (•) LOMA_ALTA
- ( ) ~Sin asignar

Continuar

Listado de objetos en ARQUEOLOGIA - ECUADOR - GUAYAS

ID	Descripcion
135	Fragmento de extremidad inferior izquierdo de una figurita antropomorfa de cerámica. La forma es troncocónica y los dedos del pie se señalan por medi incisiones. Sobre el tobillo se pueden ver cuatro líneas grabadas horizontales, sólo en la parte frontal de la figura. El acabado de la superficie es un engob color marrón claro con pulido.
1165	Esqueleto de un pelicano actual, perteneciente a la muestra comparativa del Laboratorio de Arqueología Americana de la UCM
1198	Fragmento de vasija cerámica de pasta marrón anaranjado. Cuello corto, evertido, borde engrosado al exterior y labio redondeado. La superficie interio está engobada, pulida y lleva una banda de pintura roja en el borde, que cubre también la parte externa del labio. La superficie exterior está pulida y llev

Ahora comentaremos brevemente como se ha adaptado esta navegación al framework ZNF, mostrando las entradas del fichero de configuración *znf-objetoVirtual-config.xml*, correspondientes a la misma:

```
<action
     path="objetosNavegarClas"
     type="ObjetoVirtual_Action_ObjetoVirtualNavegacionClas">
     <forward
          name="secciones"
          path="ObjetoVirtual/Presentation/ObjetoVirtualNavegacionCl
          as.php" />
</action>
<action
     path="seccionSubmit"
     type="ObjetoVirtual_Action_BuscarSeccion"
     name="NavegacionForm"
     input="objetosNavegarClas"
     validate="true">
     <forward
          name="success"
          path="ObjetoVirtual/Presentation/ResultSeccion.php" />
      <forward
          name="error"
          path="objetosNavegarClas" />
</action>
<action
      path="sigOpSubmit"
      type="ObjetoVirtual_Action_BuscarSigOp"
      name="NavegacionSigOpForm"
      input="seccionSubmit"
      validate="true">
      <forward
          name="success"
          path="ObjetoVirtual/Presentation/ResultSigOp.php" />
      <forward
          name="error"
          path="objetosNavegarClas" />
</action>
```

El primer módulo **action** solamente nos muestra las secciones que podemos elegir. Utiliza para ello la clase *ObjetoVirtual_Action_ObjetoVirtualNavegacionClas*, que es la que realiza la acción de consultar la base de datos mediante el método *getSecciones*, del fichero de la carpeta *Business* (parte encargada de interactuar con la base de datos), titulado *objetoVirtualDAO.php*. Para la presentación se usa el archivo *ObjetoVirtual/Presentation/ObjetoVirtualNavegacionClas.php* que a su vez se apoya en la plantilla *objVirtNavClas.tpl*, que es realmente enseña los datos por pantalla.

El segundo módulo **action** se encarga de mostrar las opciones correspondientes a la sección elegida en el módulo anterior. Usa la clase

ObjetoVirtual_Action_BuscarSeccion, la cual realiza sendas llamadas a las funciones *buscarOpcionesSeccion* y *buscarOpcionesListSeccion* del fichero *objetoVirtualDAO.php*. Estas funciones devuelven los metadatos válidos del siguiente paso (tarea de *buscarOpciones*), y los listados con sus correspondientes valores (trabajo de *buscarOpcionesList*). En la presentación se recurre al fichero *ObjetoVirtual/Presentation/ResultSeccion.php*, que visualizará los datos usando la plantilla *resSeccion.tpl*.

El último módulo **action** funciona de la misma manera que el módulo anterior, y es el que será utilizado en los posteriores pasos de la navegación. Sólo se diferencia en que los métodos a los que llama la clase *ObjetoVirtual_Action_BuscarSigOp*, son *buscarOpciones* y *buscarOpcionesList* también de *objetoVirtualDAO.php*. Estos métodos son un poco más complejos que los mencionados en el módulo **action** anterior pero su estructura es básicamente la misma.

Para la obtención del listado de objetos virtuales se usan los métodos *buscarObjetosSeccion* y *buscarObjetosSigOp*, para el segundo y el tercer módulo **action** respectivamente. Evidentemente los dos son se encuentran en el archivo *objetoVirtualDAO.php*.

Sólo cabe añadir que los ficheros *NavegacionForm.php* y *NavegacionSigOpForm.php*, se usan para poder conservar las variables correspondientes a la ruta escogida, a lo largo de los pasos de la navegación. También nos alertan cuando no hemos seleccionado ninguna opción, mostrándonos el correspondiente mensaje de error.

4.2. Inserción de nuevos objetos virtuales

El desarrollo de esta funcionalidad, se ha dado a causa de la necesidad de comprobar como se actualiza la navegación, a la hora de insertar objetos con nuevas secciones y nuevos metadatos de clasificación. No comprende todas las funciones que aparecían en la inserción de objetos del sistema anterior, ya que nos hemos centrado principalmente en las que nos interesan a nosotros. Cabe resaltar que no permite añadir un número indefinido de nuevos valores en cualquiera de las fichas, y no se realizan las acciones de empaquetar los objetos para subirlos a la base de datos, sino que se insertan directamente. Tampoco contempla la acción de añadir los ficheros que representan a los recursos del objeto, aunque si se pueden añadir los metadatos correspondientes a los mismos. Estas tareas se dejan como futuro trabajo de los cursos posteriores, junto con la modificación de objetos virtuales, en la que parte del código usado aquí podrá ser reutilizado.

En lo que nos concierne, lo fundamental de esta inserción es que mantiene la navegación actualizada, modificando las tablas de navegación o creando una nueva si es necesario, como ya hemos comentado en la explicación de la nueva navegación. Las funciones que se encargan de esto son *creaTablaNuevaSeccion* y *actualizaTabla*, de *objetoVirtualDAO.php*, fichero encargado de llamar a la base de datos.

Su adaptación y funcionamiento dentro del framework ZNF se muestra a continuación, usando para ello el fichero de configuración *znf-objetoVirtual-config.xml*, como en el caso anterior. Las entradas del mismo en relación a la inserción de objetos son:

```
<action
     path="objetosAgregar"
     forward="ObjetoVirtual/Presentation/ObjetoInsert.php"
     roles="admin"
     nextPath="objetoInsertSubmit" />
<action
     path="objetoInsertSubmit"
     type="ObjetoVirtual_Action_Objeto"
     roles="admin"
     name="objInsertForm"
     input="objetosAgregar"
     validate="true"
     parameter="method">
     <forward
          name="success"
          path="ObjetoVirtual/Presentation/ObjetoInsertSubmit.php"/>
</action>
```

El primer módulo **action**, simplemente se encarga de mostrar por pantalla el formulario que recoge los datos referentes a las fichas de los metadatos. Recurre para ello al fichero de presentación *ObjetoVirtual/Presentation/ObjetoInsert.php*, que a su vez utiliza la plantilla *objInsert.tpl*. Es ésta la que realmente muestra el formulario en la pantalla, además de indicar al usuario que debe rellenar obligatoriamente los campos *usuario*, *descripción* y *sección* del objeto. Esto se consigue mediante la función *checkfield* incluida en la plantilla. Sólo se permite la inserción de objetos a usuarios logueados.

El segundo módulo **action**, es el que inserta directamente en la base de datos el objeto, añadiendo los nuevos valores a la tablas **objeto_virtual**, **metadatos**, y las de navegación. Utiliza para ello la clase *ObjetoVirtual_Action_Objeto*, que efectúa varias llamadas a los métodos de inserción en la base de datos, creados en el fichero *objetoVirtualDAO.php*. Los metadatos de clasificación se añaden a las tablas de navegación mediante los métodos *creaTablaNuevaSeccion* y *actualizaTabla*,

también llamados desde *ObjetoVirtual_Action_Objeto*. El archivo *ObjetoInsertForm.php*, es el que se encarga de obtener las variables que guardan el contenido de los campos de texto del formulario. Si los mismos son rellenados correctamente se mostrará por pantalla el mensaje de aceptación del nuevo objeto. Se recurre para ello al fichero *ObjetoVirtual/Presentation/ObjetoInsertSubmit.php*, el cual se apoya en la plantilla *index.tpl*, para ir a la página principal del sistema mostrando el mensaje de correcta inserción.

5. Conclusiones y líneas de trabajo futuro

A pesar de que la elección de crear un nuevo sistema desde cero limitó en años anteriores la funcionalidad del mismo respecto al sistema actualmente en producción, ha quedado demostrado que dicho cambio supone tener un nuevo sistema mucho más potente gracias a la división en capas de la aplicación, la cual permitirá fácilmente en líneas de trabajo futuro completar tanto las funcionalidades omitidas hasta ahora como una mejora de la vista para mejorar la usabilidad del interfaz del sistema cara al usuario, así como la implementación de nuevas funcionalidades de una forma más rápida, clara y estructurada.

Como líneas de trabajo futuro, como se acaba de comentar, la más importante sería dotar al nuevo sistema de todas las funcionalidades de las cuales dispone actualmente la versión 2 de Chasqui.

Posteriormente se podría dotar al sistema de la capacidad de interconexión con otros sistemas mediante el uso de *"web services".* Debido a que el intercambio de información entre los sistemas que utilizan *"web services"* es XML y el nuevo sistema ofrece salida en dicho formato, gran parte del trabajo ya estaría hecho.

6. Bibliografía

1. IMS: www.imsglobal.org
2. LOM: http://ltsc.ieee.org/wg12/
3. Apache Struts: http://struts.apache.org
4. PHP: http://www.php.net
5. PEAR: http://pear.php.net/
6. XML: http://www.w3.org/XML
7. MySQL: http://www.mysql.com
8. Smarty: http://smarty.php.net/
9. XSLT: http://www.w3.org/TR/xslt

Se autoriza a la Universidad Complutense de Madrid a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria, como el código, la documentación y/o el prototipo desarrollado".

Miguel Ángel Alonso Pajuelo

Jose Luis Rojo Rojo