SISTEMAS INFORMÁTICOS
Curso 2005-2006

Creación, gestión y uso de “objetos de aprendizaje” en un entorno web

Sheng Fang
José Antonio Villegas López
José Corbacho Gil

Dirigido por:

Alfredo Fernández-Valmayor Crespo
Dpto. Sistemas Informáticos y Programación

Facultad de Informática
Universidad Complutense de Madrid

Índice

AGRADECIMIENTOS

Nos es grato haber contado con la colaboración de nuestro director de proyecto Alfredo Fernández-Valmayor y del profesor Antonio Navarro por el tiempo que nos han dedicado y por su asesoramiento técnico.

Resumen

El proyecto “Creación, Gestión y Uso de objetos de Aprendizaje en un Entorno Web” se ha desarrollado en la asignatura Sistemas Informáticos. Tiene como objetivo el análisis de la versión 2.0, actualmente en producción, del Museo Virtual de Arqueología de la Facultad de Geografía e Historia de la Universidad Complutense de Madrid.

Este proyecto se centra en 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 Web” has been developed into the subject “Sistemas Informáticos”. The goal is to analyze the 2.0 version, currently in production, of the “Virtual Archaeology Museum” located at the “Facultad de Geografía e Historia” of the Complutense University of Madrid.

This project focuses in the construction of a tool to be useful in the creation and management of educational resources, *Learning Objects*, modular Objects that they may be used by various learning systems.

Palabras Clave

E-Learning; Museos Virtuales; Objetos de Aprendizaje; PHP; ZNF; XML; AJAX; IMS; LOM

1.- Introducción

El presente documento ofrece una visión sobre las actividades realizadas en la asignatura “Sistemas Informáticos, Curso 2005 - 2006”, la cual ha estado enfocada en el proyecto “Creación, Gestión y Uso de Objetos de Aprendizaje en un entorno Web”, enmarcado dentro del área e-learning.

Este proyecto se basa en los trabajos realizados en el curso anterior, en la cual se creó una herramienta que permite crear y gestionar recursos educativos (en adelante “Objetos de Aprendizaje”), de forma que estos puedan ser utilizados por otros sistemas de enseñanza. Los “Objetos de Aprendizaje” creados y gestionados mediante dicha herramienta, corresponden con los objetos físicos del museo Arqueología de la Facultad de Geografía e Historia. La herramienta creada fue bautizada con el nombre de CHASQUI.

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 educativos así como poder ser importados desde otras aplicaciones (siempre y cuando cumplan éstas un determinado estándar).

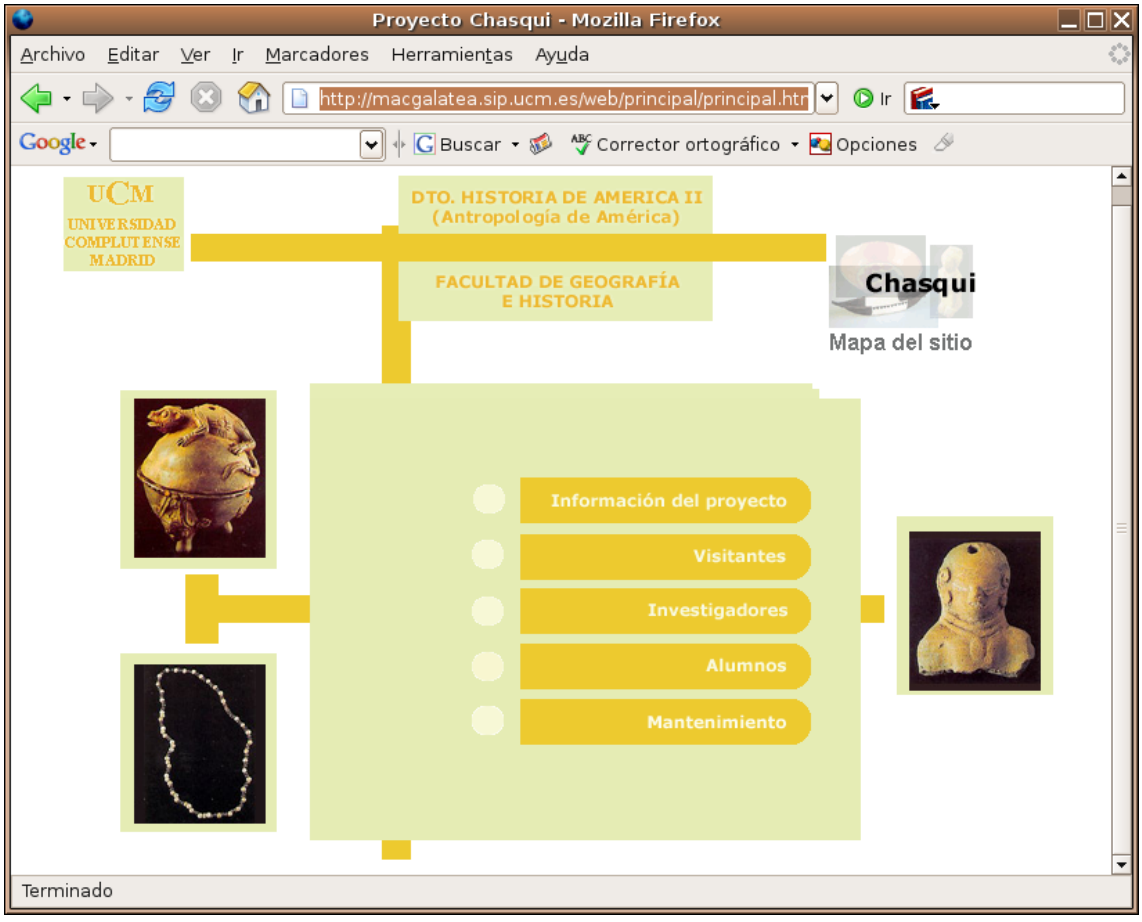
El objetivo actual del proyecto se centra en el análisis de Chasqui v.2, sistema el cual estaba pensado que se pondría en producción a lo largo de este curso. Actualmente todavía se encuentra en producción la versión 1. No obstante, nuestro trabajo se ha centrado en la versión 2, con la finalidad de analizar la estructura y diseño de este sistema, de identificar las limitaciones que presenta el modelo actual y ofrecer una nueva arquitectura a modo de prototipo para superar dichas limitaciones. Debido a la gran densidad del sistema, la nueva arquitectura solo cubrirá una parte funcional de la aplicación, la cual servirá de base para los trabajos a realizar en el proyecto en cursos posteriores.

2.- Chasqui

Con el concepto de "*Objeto de Aprendizaje*" (digitalización objetos reales) nace en el año 2002, el Museo de Arqueología de la Facultad de Geografía e Historia, bautizado como Chasqui. 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 del Grupo de Ingeniería del Software e Inteligencia Artificial del departamento de Sistemas Informáticos y Programación.

Este museo arqueológico, 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 de Arqueología.

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

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

3.1.- Utilización de estándares en el aprendizaje en línea

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 necesarios unas especificaciones y 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.

3.2.- Herramientas de Referencia

Este proyecto está basado en el concepto de Objeto de Aprendizaje (learning objects) promovido por iniciativas como IEEE/LTSC (Learning Technology Standards Comittee) [1], IMS [2], ADL (Advanced Distributed Learning). SCORM (Sharable Content Object Reference Model) [3], promovida por ADL, es una especificación para la construcción de objetos de aprendizaje complejos a partir de otros más sencillos.

Según L. Mortimer [4] algunos de los aspectos que caracterizan a un objeto de aprendizaje son:

- *Contenido:* contenido y actividades que deben soportar un aprendizaje objetivo, y su evaluación debe de estar orientada a conseguir dicha objetividad.

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

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

- *Etiquetado y almacenado*: El contenido se describe mediante un conjunto de etiquetas normalizadas (metadatos).

- *Construcción incremental*: Los objetos de aprendizaje complejos (con una estructura bien definida) se construyen utilizando otros objetos de aprendizaje

previamente construidos partiendo de objetos de aprendizaje más básicos (hasta llegar a los objetos atómicos).

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

Uno de los conceptos más importantes sobre el que se asienta este proyecto es el concepto de Objeto Virtual (OV), un tipo específico de Objeto de Aprendizaje (Learning Object). Un Objeto Virtual puede ser definido como "*objeto digital que sirve para agrupar con fines educativos toda la información relacionada con un determinado objeto real*". Así pues la estructura de datos de un Objeto Virtual está especificada como:

- ***Datos***: Permiten organizar y describir de forma extensible las características del objeto.

- ***Recursos***: Constituyen un conjunto de elementos informativos asociados al objeto. Dichos recursos pueden ser considerados de tres clases:

    - *Propios*: Archivos digitales asociados de forma primaria a un objeto virtual.

    - *Ajenos*: Archivos digitales asociados de forma primaria a otros objetos del almacén pero que guardan algún tipo de relación con el objeto actual.

    - *Otros Objetos Virtuales*: son considerados de forma unitaria y asociados al objeto virtual estudiado mediante una relación describiendo las propiedades que pueden incorporarse al objeto.

- ***Metadatos***: Describen el contenido, calidad, condiciones y otras características de los datos, permitiendo así a una persona ubicar y entender los datos. Cada uno de estos objetos puede ser descrito mediante un modelo de metadatos, y en este caso están basados en la propuesta de LOM (Learning Object Metadata) [5], cuyo modelo permite disponer de una forma estándar de clasificación de la información y establecer las posibles relaciones existentes entre los diferentes Objetos Virtuales.

Además de las características de los objetos de aprendizaje ya mencionadas, también se ha dado gran importancia durante el desarrollo del proyecto a los estándares educativos que a continuación se introducen.

SCORM [3], iniciativa de ADL, ha sido la principal referencia para este proyecto. Otras iniciativas que también nos han servido de referencia han sido: ARIADNE, IEEE/LTSC e IMS. En SCORM [3] se define un "modelo de agregación de contenido" (content agregation model), el cual debe de ser pedagógicamente neutro. El modelo de agregación de contenido propuesto por SCORM [3] está formado básicamente por tres elementos: el modelo de contenido, los metadatos, y el modelo de empaquetamiento. Así

mismo, el modelo de contenido se compone a su vez de tres elementos: assets (los contenidos más básicos), SCOs (Sharable Content Object) conjunto de assets que pueden ejecutarse en el entorno de enseñanza del sistema y finalmente las estructuras de agregación (content agregation) las cuales son capaces de organizar diferentes recursos de aprendizaje hasta formar una unidad didáctica coherente.

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*.

Por otro lado, existen diferentes grupos como Dublín Core [6] y ARIADNE [7] que han ajustado sus esquemas de metadatos a LOM, pero con menos nivel de detalle.

La aproximación que utilizamos para desarrollar este proyecto tiene como base los estándares antes descritos y el concepto de *Objeto Virtual*. El dominio de este proyecto son los objetos que se encuentran en el museo de 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.

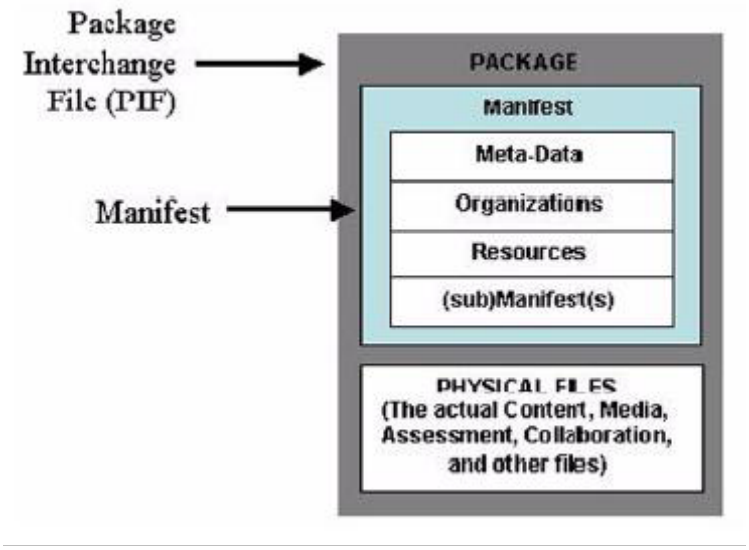
3.3.- IMS Content Packaging (IMS-CP)

Es la especificación seguida para el empaquetamiento de objetos virtuales en Chasqui. El IMS-CP describe las estructuras de datos que son usadas para proporcionar la interoperabilidad de contenidos basados en internet con herramientas de creación de contenidos, sistemas de gestión de aprendizaje (LMS), y entornos de ejecución.

El objetivo de IMS-CP es definir un conjunto estandarizado de estructuras que puedan ser usadas para intercambiar contenidos. Estas estructuras proporcionan la base para las uniones de datos estandarizadas que permiten crear aplicaciones que se comuniquen con herramientas de creación de contenidos, sistemas de gestión de aprendizaje (LMS), y entornos de ejecución que han sido implementados por separado por distintos desarrolladores de software.

El alcance de la especificación IMS-CP está centrada en la definición de la interoperabilidad entre sistemas que importan, exportan, añaden y borran paquetes de contenidos.

El Modelo conceptual se especifica de la siguiente manera:



Modelo IMS-CP.

- **Archivo de Intercambio de Paquetes (PIF)**: Es un archivo (.zip, .jar …), que incluye en su capa más alta, un archivo tipo "manifiesto" llamado "imsmanifest.xml" y otros archivos físicos identificados por el manifiesto. Un PIF es un conciso formato de entrega Web y medio de transportar información relacionada y estructurada.
- **Paquete**: directorio lógico, que incluye un archivo XML, en el cual, cualquier documento XML de control es directamente referenciado (como un archivo DTD ó XSD), y contiene los recursos reales físicos. Los recursos físicos pueden ser organizados en subdirectorios:
    - *Manifiesto de alto nivel*: un elemento obligatorio XML que describe el Paquete. Puede contener opcionalmente submanifiestos. Cada manifiesto contiene las secciones siguientes:
        - Meta datos: un elemento XML que describe un manifiesto en totalidad.
        - Organizaciones: un elemento XML que describe cero, una, o múltiples organizaciones del contenido dentro de un manifiesto.
        - Recursos: un elemento XML que contiene referencias a todos los recursos y elementos necesitado para un manifiesto, incluyendo

meta-datos que describen los recursos, y referencias a cualquier archivo externo;

- Sub-manifiestos: uno o varios, opcional.

- *Archivos de recursos*: estos son los elementos reales, archivos de texto, gráficos, y otros recursos descritos por el manifiesto. Los recursos físicos pueden ser organizados en subdirectorios.

4.- Análisis del sistema Chasqui (v.2)

4.1.- Instalación y puesta en marcha del sistema

Ya que nuestro trabajo se iba a centrar en mejorar la versión 2 de Chasqui, en un principio, nuestros esfuerzos se centraron en conocer el sistema Chasqui, principalmente, 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 íbamos a trabajar un primer paso fue instalar el sistema en nuestros ordenadores personales. A continuación se detallan los procesos de instalación que se han seguido para tener disponible el sistema Chasqui tanto en entorno Windows como en entorno Linux.

4.1.1.- Instalación en entorno Linux

En el entorno Linux, se ha utilizado las siguientes versiones de software para montar Chasqui v.2:

- Apache 2.0.54
- MySQL 4.1.12
- PHP 4.4.0-3

Modificaciones necesarias en la de configuración de Apache 2:

El fichero de configuración de Apache 2 se encuentra en la ruta /etc/apache2/apache2.conf.

Generalmente en todas las distribuciones de Linux, Apache viene configurado para trabajar con PHP. Para asegurarnos debemos comprobar que en fichero de configuración tenemos las siguientes entradas (o similares):

LoadModule php4_module ….…
AddModule mod_php4.c
AddType application/x-httpd-php .php
DirectoryIndex index.html index.htm index.php

Debemos indicar a Apache donde van a residir los ficheros de nuestra aplicación web. En este caso debemos modificar la siguiente línea en el fichero /etc/apache2/sites-available/default:

DocumentRoot '/home/jose/web_chasqui'

Finalmente, debemos asegurarnos que el usuario con el que se está ejecutando Apache (generalmente www-data) tiene permisos para leer los ficheros albergados en el DocumentRoot. Una posible manera de hacerlo sería ejecutar el siguiente comando:

chmod –R go+rx /home/jose/web_chasqui

Modificaciones necesarias en la configuración de PHP 4:

La ruta del fichero de configuración de PHP 4 es: /etc/php4/apache2/php.ini
En dicho fichero hay que hacer las siguientes modificaciones :

register_globals = On
upload_max_filesize = 4M

Modificaciones necesarias en la configuración de PHP 5:

Además se hicieron pruebas con la versión PHP 5.0.5-2. En el caso de la versión 5 de PHP, el fichero de configuración se encuentra en la ruta: /etc/php5/apache2/php.ini
En dicho fichero hay que realizar las mismas modificaciones descritas anteriormente para PHP 4.

Si se necesita configurar Apache para tener soporte de PHP5 hay que crear los siguientes enlaces simbólicos desde la ruta /etc/apache2/mods-enabled:

ln -s ../mods-available/php5.conf
ln -s ../mods-available/php5.load

Modificaciones necesarias en la configuración de MySQL:

El gestor de base de datos MySQL diferencia entre mayúsculas y minúsculas en el caso de que el sistema operativo lo haga. Este es el caso de los sistemas UNIX / LINUX. En el caso de sistemas Windows y Mac no existe dicha distinción.
Se puede obtener más información acerca de esto en la siguiente URL del sitio web oficial de MySQL: http://dev.mysql.com/doc/refman/5.0/en/name-case-sensitivity.html

Debido a que las consultas realizadas en el código fuente del sistema Chasqui se hacen en mayúsculas, y las tablas están almacenas en minúsculas, en los sistemas UNIX / LINUX se debe agregar dentro de la sección *[mysqld]* del fichero de configuración (/etc/mysql/my.cnf) la siguiente línea

lower-case-table-names=1

Hay que establecer la contraseña para el usuario root introduciendo el siguiente comando:

mysqladmin -u root password root

Finalmente hay que ejecutar el cliente *mysql* e introducir el siguiente comando:

SET PASSWORD FOR root@localhost = OLD_PASSWORD('root');

Esta última parte se aclara en el apartado "4.1.3.- Comunicación MySQL 4 – PHP 4" que aparece más adelante en este documento.

4.1.2.- Instalación en entorno Windows

En el entorno Windows, se ha utilizado las siguientes versiones de software para montar Chasqui v.2:

- Apache 1.3.34
- MySQL 4.1.16
- PHP 4.4.1

Modificaciones necesarias en la de configuración de Apache 2:

El fichero de configuración de Apache 2 se encuentra en la ruta C:\Archivos de programa\Apache Group\Apache\conf\httpd.conf, en el cual se deben hacer las siguientes modificaciones:

- Al final de la seccion LoadModule agregar:
  LoadModule php4_module "c:/php-4.4.1/php4apache.dll"

- Al final de la seccion AddModule agregar:
  AddModule mod_php4.c

- Agregar la siguiente línea dentro del condicional <IfModule mod_mime.c>
  AddType application/x-httpd-php .php

- Modificar la siguiente línea dentro del condicional <IfModule mod_dir.c>
  DirectoryIndex index.html index.htm index.php

- Modificar la siguiente línea para que apunte al directorio donde residen los ficheros con el código fuente del sistema Chasqui
  DocumentRoot "D:/Mis documentos/web_chasqui"

Configuración de PHP 4:

Descargar el archivo *php-4.4.1-Win32.zip* de la web oficinal de php y realizar los siguientes pasos:

Descomprimir el fichero zip en el directorio C:\php-4.4.1
Crear la carpeta C:\php-4.4.1\uploads
Crear la carpeta C:\php-4.4.1\sessions
Copiar el fichero C:\php-4.4.1\php4ts.dll en c:\windows\system32
Copiar el fichero C:\php-4.4.1\dlls\iconv.dll en c:\windows\system32
Copiar el fichero C:\php-4.4.1\php.ini-recommended c:\windows y cambiar el nombre del fichero copiado a php.ini
Copiar el fichero C:\php-4.4.1\sapi\php4apache.dll en C:\php-4.4.1

Posteriormente hay que hacer las siguientes modificaciones en el fichero de configuración de PHP 4 (c:\windows\php.ini)

register_globals = On
upload_tmp_dir =C:\php-4.4.1\uploads
upload_max_filesize = 4M
session.save_path = C:\php-4.4.1\sessions
extension_dir = "C:/php-4.4.1/extensions/"
extension=php_domxml.dll
extension=php_zip.dll

Configuración de MySQL 4.x:

Hay que establecer la contraseña para el usuario root a través del cliente *mysql* mediante el siguiente comando:

mysql -u root -p password root

Finalmente hay que ejecutar el cliente *mysql* e introducir el siguiente comando:

SET PASSWORD FOR root@localhost = OLD_PASSWORD('root');

Esta última parte se aclara en el apartado “4.1.3.- Comunicación MySQL 4 – PHP 4” que aparece más adelante en este documento.

4.1.3.- Comunicación MySQL 4 – PHP 4

En el caso de tener instalado MySQL 4 y intente establecer conexiones a la base de datos con clientes viejos, se pueden tener problemas de conexión al servidor de MySQL.

Este problema se manifiesta de la siguiente forma:

> *Warning: mysql_connect(): Client does not support authentication protocol requested by server; consider upgrading MySQL client.*

Dicha situación es reproducida en el caso de que se tenga instalado MySQL 4.x y se tenga PHP configurado en la sección mysql con la version 3.x del *Client API*, este es el caso de PHP 4.

Para averiguar la versión del *Client API*, se puede ejecutar el un fichero php con el contenido *< ? phpinfo(); ?>* el cual devolverá la configuración actual de PHP.

El problema reside en que la versión 4 de MySQL ha mejorado el sistema de almacenamiento de contraseñas, y si se está intentando establecer una conexión a un servidor MySQL 4 desde un cliente 3 (PHP 4 incorpora dicha versión de cliente), la conversión que hace el cliente 3.x de la contraseña para pasársela a MySQL no coincide con la que espera recibir el servidor.

La solución reside en cambiar la forma en que MySQL 4 almacena las contraseñas para que use la antigua forma. Para ello hay que ejecutar el cliente *mysql* e introducir el siguiente comando:

SET PASSWORD FOR some_user@some_host = OLD_PASSWORD(new_passwd');

4.1.4.- Importación de la base de datos

Chasqui es una aplicación web que almacena su información en el gestor de base de datos MySQL. Dicha información va actualizándose a medida que los usuarios de Chasqui van agregando nuevos contenidos.

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 –ppassword < chasqui.sql

4.1.5.- Verificación de la instalación

Para comprobar que está funcionando todo el software necesario para el sistema Chasqui, hay que crear dos ficheros PHP y ubicarlos en la raíz del sitio web (DocumentRoot).

Contenido del fichero phpinfo.php

```
<html>
    <head>
    </head>
    <body>
        <?php phpinfo(); ?>
```

```
        </body>
</html>
```

Contenido del fichero db.php

```
<?php
        $link = mysql_connect('localhost', 'root', 'root');
        if (!$link) {
          die('No se ha podido conectar: ' . mysql_error());
        }
        echo 'conexion correcta';
        mysql_close($link);
?>
```

Ambos ficheros deben ser invocados desde un explorador web de la siguiente manera:

http://localhost/phpinfo.php
http://localhost/db.php

4.1.6.- 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 raiz 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 .

4.2.- Estudio de la arquitectura actual

Una vez realizada la instalación de Chasqui en nuestros ordenadores, pasamos a comprender la funcionalidad de la aplicación mas 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 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.

Para poder decidir una de estas dos opciones, realizamos un análisis exhaustivo del sistema, especialmente del diseño, de la implementación y de la base de datos. De esta forma también podíamos identificar posibles defectos o carencias del sistema.

A continuación se presentan la documentación relativa a este estudio y la decisión final que tomamos en base a dicho estudio.

4.2.1.- Abuso de "register_globals" en PHP

Como se ha mostrado anteriormente, para el correcto funcionamiento del sistema Chasqui actual, PHP debe estar configurado con la directiva *register_globals* activada (*register_globals = On*). A continuación se demuestra como la activación de esta directiva puede acarrear problemas de seguridad en el sistema Chasqui.

Hasta hace no mucho, todas las versiones de PHP venían, por defecto, con la opción de "register_globals" activada en su configuración. Su utilidad era forzar a que cualquier variable pasada al script, como entrada de usuario (vía GET, POST o cookie), fuera tratada como una variable global. Esto simplificaba la tarea del programador. Sin embargo, esto podía traer problemas en la seguridad. Obsérvese el código PHP ilustrado en la siguiente figura:

```
<?php
    if ($username==”root”) { // puede ser falsificado por un usuario mediante
                             // get/post/cookies
        $good_admin_login = 1;
    }
    if ($good_admin_login == 1) { // también puede ser falsificado
```

```
        fpassthru("/daots/altamente/sensibles/index.html");
    }
?>
```

El programa tiene dos partes bien diferenciadas: una primera donde realiza una primera comprobación (para ver si el usuario es o no “root”), en función de la cual asigna o no un valor a una segunda variable ($good_admin_login); y otra donde comprueba precisamente esta última variable.

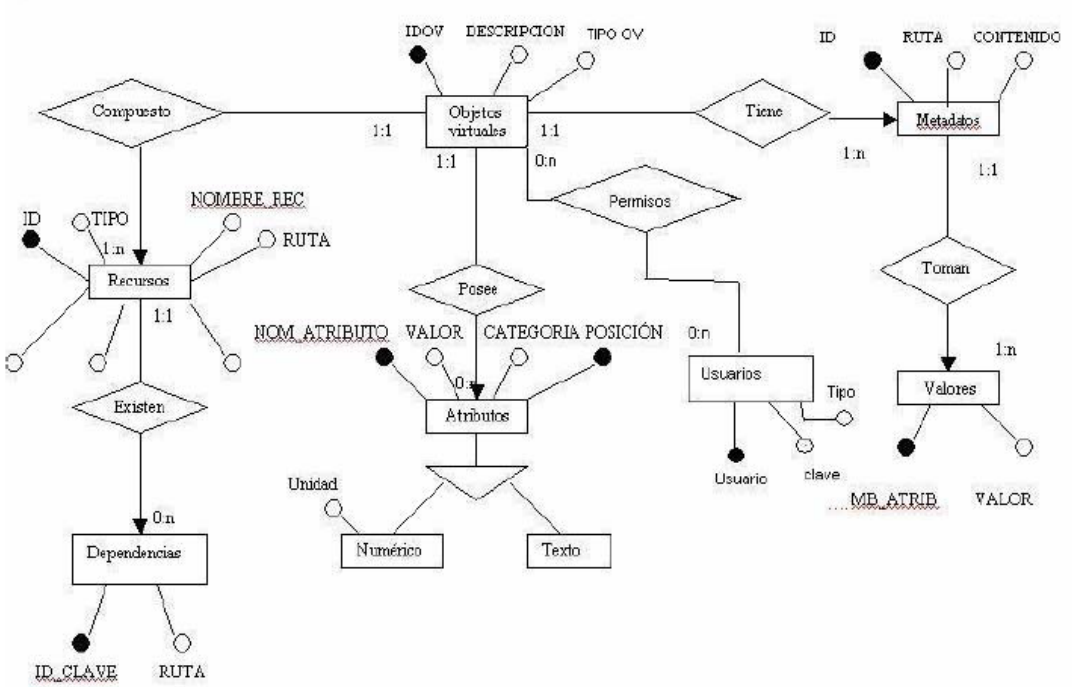
El problema está en esta segunda parte: si la primera comprobación no es correcta, la variable $good_admin_login quedará sin asignar. ¿Qué ocurriría si el atacante realiza esta asignación por su cuenta? Con la opción “$register_globals = On” (por defecto en la mayoría de versiones de PHP) podremos pasarle al script la variable anterior, y definirla a 1, de forma que conseguiríamos burlar la protección (la segunda comprobación).

El fallo ha sido la alteración (o “envenenamiento”) de una variable que se suponía interna del programa y en ningún momento debía haber estado al alcance del atacante (i.e., no se debería de poder modificar externamente). No ha sido así, por culpa de esta peligrosa opción: “register_globals”.

La versiones de PHP más recientes traen esta opción deshabilitada, de forma que dentro del script PHP se distingue entre una variable global corriente y una variable GET (o POST o cookie). Para ello, el programador debe referenciar explícitamente el tipo de variable. Por ejemplo, para que el programa anterior funcionara adecuadamente sería necesario sustituir $username por lo siguiente: $_GET[‘username’]. Si así lo hacemos, el programa aceptara y leerá esta variable de una petición GET. Existen formas de referenciación similares para variables de tipo POST y Cookie.

4.2.2.- Análisis de la 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. Hay seis tablas fijas para la navegación, que se corresponden con los valores del metadato Sección/Sección. Estas tablas son: **material_documental, arqueología, etnología, reproducciones, material_docente y agha**. Si se sube un nuevo objeto virtual, y se le pone como valor del metadato Sección/Sección uno distinto de los 6 anteriores, se crearía una nueva tabla cuyo nombre fuese el valor introducido.

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.

4.2.3.- 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 quieren 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.

4.2.4.- Viabilidad modificación sistema actual

Chasqui está organizado en una serie de directorios. Esta organización no es muy clara y no está orientada a un modelo vista controlador. Los directorios que presenta el chasqui, dentro del directorio raíz de la aplicación son los siguientes:

- **Cabecera**: Contiene un archivo php con la cabecera que llevan la mayoría de las páginas de la aplicación y que es incluida por estas.

- **Clases**: Contiene las clases php: BuscarDatos, Recursos, Navegación, Filtro, Usuarios y BaseDatos las cuales realizan accesos a la base de datos de la aplicación.

- **CSS**: Incluye una hoja de estilo que es incluida en el diseño de algunas páginas.

- **Fichas**: Contiene plantillas para las fichas de los objetos virtuales que son vistas en el acceso a la base de datos, tanto siendo visitante, como habiéndo iniciado sesión en la aplicación.

- **Images**: Directorio que contiene algunas de las imágenes de la aplicación.

- **Login**: Contiene la página de login en la aplicación y la página que verifica la corrección del nombre de usuario y contraseña.

- **Privado**: Contiene todas las páginas necesarias para todas las acciones que puede realizar un usuario que se ha registrado en la aplicación. Estas acciones pueden ser, crear, eliminar, modificar, subir o bajar objetos virtuales. También contiene las páginas necesarias para realizar la gestión de usuarios. Además tiene la lógica necesaria para realizar la serialización/deserialización de los objetos virtuales.

- **Publico**: Contiene las páginas necesarias para la sección de alumnos de la página de inicio de la aplicación.

- **InfoProyecto**: Páginas relativas a la parte de información del proyecto.

- **Investigadores**: Contiene todas las páginas relativas a la sección de investigadores de la página principal de la aplicación.

- **Principal**: Contiene la página principal del chasqui.

- **Recursos**: Directorio que contiene todos los recursos de los objetos virtuales. Hay un subdirectorio por cada objeto virtual en el que están los recursos asociados a ese objeto tales como imágenes, páginas web, etc.

- **Visitantes**: Este directorio no se utiliza.

Para analizar la posible viabilidad de mejorar la estructura de sistema Chasqui actual, obteniendo una organización similar a la del patrón modelo-vista-controlador, se han analizado los diferentes ficheros PHP que hay en los directorios más significativos

de los mencionados anteriormente. El objetivo es separar los ficheros PHP en ficheros de vista para los usuarios de la aplicación o bien en ficheros cuyo contenido referencia al modelo de datos de la aplicación.

El análisis de dichos ficheros ha sido el siguiente:

- **Principal**: Contiene una única página html que es la página principal de la aplicación. Solo tiene código html con lo cual es una página que se incluiría en la vista.

- **InfoProyecto**: Contiene todas las páginas que aparecen en la sección de información del proyecto de la página principal. Son todas páginas que contienen código html, no realizan accesos al modelo de datos de la aplicación. Contienen imágenes de objetos virtuales que al pinchar sobre ellas redirigen a la página *publico/control.php* desde la que se muestra la ficha del objeto virtual sobre el que se ha pinchado. Por tanto, todos los ficheros php incluidos en este directorio podrían formar parte de la vista de la aplicación.

- **Clases**: En este directorio hay varios ficheros en cada uno de los cuales hay una clase. Dichas clases son las que realizan los accesos a la base de datos de la aplicación. Son usadas por otras páginas para mostrar los datos a los usuarios. No contienen nada de código html. Estos ficheros formarían parte del modelo en un modelo-vista-controlador.

- **Recursos:** Solo hay imágenes que se corresponden con los recursos de cada objeto virtual. No hay ficheros de código fuente. Con lo cual, el contenido de este directorio formaría parte de la vista.

- **Images:** Solo contiene archivos de imágenes. Formaría parte de la vista.

- **Login:** Contiene dos ficheros LOGIN.php y verificaLogin.php. El primero de ellos es la página de la aplicación que permite iniciar sesión a usuarios registrados en el museo. Únicamente contiene código html y utiliza verificaLogin.php para comprobar la corrección del nombre del usuario y la contraseña. Por tanto, LOGIN.php formaría parte de la vista de la aplicación. Por su parte verificaLogin.php solo tiene código php y JavaScript para verificar la corrección del login. Utiliza la clase BaseDatos y además realiza consultas a la base de datos. No puede formar parte de la vista. Tampoco puede formar parte del modelo ya que mezcla código php y JavaScript.

- **Publico:** Contiene varios archivos que únicamente tienen código html:
  - error.php que es una página de gestión de errores.
  - mapa.php es la página del mapa web de la aplicación
  - presentación.php que es la página de presentación de la parte de alumnos.
  - acccesoGlobal.php que es la página para acceder a la base de datos de forma clasificada, solo contiene enlaces a la página accesoGlobal.php.

  Todas estas páginas se podrían clasificar como parte de la vista.

- parteIzq.php contiene mezcla de código html y php para mostrar el árbol de la parte izquierda, además hace una llamada a una función de la clase Navegación.
- control.php es la que contiene el acceso a la base de datos a través de cada uno de los objetos virtuales. Está construida completamente en php utilizando las funciones de la clase BaseDatos. El diseño de la página se realiza incluyendo a otras páginas.
- accesoGlobalClasificacion.php es la que permite realizar el acceso a los OV a través de la clasificación en base a sus metadatos. Es una página que tiene 973 líneas de código mezclando html y php. Tiene mucha lógica condicional en php. Utiliza las clases BaseDatos y Navegación.

Por tanto las páginas control.php, parteIzq.php y accesoGlobalClasificacion.php contienen tanto parte de vista como parte de modelo. No se pueden clasificar en ninguna de las capas del MVC.

- **Investigadores:** Contiene ficheros para la parte de los investigadores. Las páginas son similares a la de la parte pública. Hay varios archivos que solo contienen código html como presentacion.php y accesoGlobal.php, las cuales podrían formar parte de la vista. Pero al igual que en **Publico** existen las páginas accesoGlobalClasificacion.php y parteIzq.php que mezclan código php y html, utilizando dentro de ellas las funciones de las clases que forman el modelo de datos. Por tanto estos dos archivos no se pueden clasificar en ninguna de las capas del MVC y separar el código sería muy complicado ya que constan de muchas líneas de código.

- **Privado:** Contiene todas las páginas de la parte del mantenimiento de la aplicación, es decir, gestión de los objetos virutales, serialización/deserialización de objetos virtuales, tareas de gestión de usuarios, etc. La página principal es control.php que es una página con código php que tiene una casuística sobre todas las acciones que puede realizar un usuario registrado. Utiliza las funciones de las clases BaseDatos, Recursos y Usuarios. La vista se realiza mediante la inclusión de otras páginas. Tiene demasiado código php para ser la vista principal de los usuarios registrados.

  En el subdirectorio gestionUsuarios están las páginas para realizar operaciones sobre los usuarios (crear, modificar, eliminar). Estas páginas contienen mezcla de código php y html, realizan accesos a la base de datos mediante invocaciones a la clase Usuarios. No se pueden clasificar ni como parte de la vista ni como parte del modelo.

  El subdirectorio formularioOV contiene paginas necesarias para crear nuevos objetos virtuales. La página principal es objeto.php la cual contiene mucho código php relativo al manejo de ficheros. Es un fichero de más de 500 líneas que mezcla la vista de la página con el tratamiento de ficheros al crear objetos virtuales. Tampoco se puede clasificar como vista ni como modelo. Además están las páginas fichaX.php y fichaXcaja.php una por cada una de las fichas que existen para cada objeto virtual. Sirven para rellenar los recursos, metadatos y atributos de un OV. Son páginas que constituyen una vista para el usuario. Pero además de html contienen código php con accesos a base de datos

mediante las funciones de la clase Filtro. No deberían tener accesos a base de datos al tratarse de formularios para el usuario.

El subdirectorio modificarOV contiene páginas necesarias para modificar los objetos virtuales ya existentes. La página principal es objeto.php la cual contiene mucho código php relativo al manejo de ficheros y además realiza accesos a base de datos mediante las funciones de la clase BaseDatos. Es un fichero de más de 500 líneas que mezcla la vista de la página con accesos a base de datos. Tampoco se puede clasificar como vista ni como modelo. Además están las páginas fichaX.php y fichaXcaja.php una por cada una de las fichas que existen para cada OV. Sirven para poder modificar los recursos, metadatos y atributos de un OV. Son páginas que constituyen una vista para el usuario. Pero además de html contienen código php con accesos a base de datos mediante las funciones de la clase Filtro y BaseDatos. No deberían tener accesos a base de datos al tratarse de formularios a rellenar por el usuario. Imposible clasificar las páginas de este subdirectorio como parte de la vista o del modelo.

4.2.5.- Conclusiones sobre el sistema actual

Base de datos

Aunque anteriormente se han expuesto posibles mejoras que se podrían reflejar en la base de datos, esto supondría tener que hacer grandes modificaciones a nivel de SQL. Dichas modificaciones a nivel SQL también se deberían reflejar a nivel de PHP. Por tanto el hacer dichos cambios supondría perder la compatibilidad con el sistema actual en producción de los datos almacenados en la base de datos.

Por otro lado, estas posibles mejoras no son muy significativas, y lo que es el núcleo principal de la base de datos es correcto con respecto a la funcionalidad que debe proporcionar el sistema.

Logica de negocio

La idea del modelo vista controlador (MVC) es separar la lógica de la aplicación de la vista que se presenta a los usuarios, de tal forma que las páginas que se visitan a través de internet solo contengan código HTML y el mínimo código posible de PHP, Java u otro lenguaje. Desde la vista se accedería al modelo de datos a través del controlador. Es en el modelo de datos donde debe estar toda la funcionalidad de la aplicación y los accesos a la base de datos. En la vista únicamente se presenta a los usuarios de la aplicación, el contenido del modelo de datos.

En el chasqui, la mayoría de las páginas que son visitadas por los usuarios mezclan código html con php. Son páginas con muchas líneas de código en las que hay demasiada lógica condicional e incluso se realizan accesos a base de datos. Únicamente se pueden distinguir dos partes bien diferenciadas: por un lado las páginas de la información del proyecto Chasqui y las páginas de presentación para usuarios e investigadores las cuales solo contienen código html y podrían formar parte de la vista.

Por otro lado, el directorio clases contiene clases php en las que se realizan funciones con accesos a base de datos, estas clases formarían parte del modelo de datos.

Sin embargo, las páginas más importantes de la aplicación, tales como las de navegación, las de visita de los distintos objetos virtuales y las de creación o

modificación de dichos objetos virtuales son páginas cuyo código es mezcla de html, JavaScript y php, con lo que resultan imposibles de clasificar.

En muchas de estas páginas no se respetan mayúsculas y minúsculas a la hora de hacer las consultas a la base de datos, con lo cual dificultada la portatibilidad de la aplicación a entornos UNIX / LINUX. Se hacen muchas comparaciones con cadenas de caracteres en vez de estar dichas cadenas definidas como constantes para facilitar posibles modificaciones del código. Finalmente no existe un único punto de acceso a la base de datos, estando en múltiples ficheros el nombre de usuario y contraseña para acceder a la base de datos, lo cual dificultaría cambiar periódicamente dichas credenciales de autenticación para fomentar la seguridad del sistema.

Debido a todo lo anteriormente comentado, no se puede crear en el sistema Chasqui actual un MVC como tal. Otra posibilidad sería realizar una organización por directorios, pero sería muy débil ya que solo una pequeña parte de la aplicación es clasificable en alguna de las capas del MVC. Las partes más importantes de la aplicación no se pueden clasificar en ninguna de las capas.

Además realizar una reestructuración de estas páginas es muy complejo debido a la gran dificultad que supone comprender el código al no estar suficientemente documentado, y la aplicación seguiría siendo poco mantenible.

Finalmente, la opción tomada ha sido 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 estará mejor estructurada, siendo más clara y más mantenible, adaptándose fácilmente a las tecnologías más modernas del mercado.

Interfaz

Debido a que se ha optado por la opción de rehacer la aplicación por completo, se han rebajado las funcionalidades de la aplicación debido a la carencia de tiempo. Esto no implica que dichas funcionalidades no puedan ser abordadas posteriormente puesto que el modelo de datos, como se ha comentado anteriormente, va a permanecer constante. De esta forma el modelo de datos será compatible tanto con la versión actualmente en producción de Chasqui, como con la nueva versión.

Por otra parte, al hacer uso de un framework MVC, tenemos la vista completamente separada de la lógica de negocio con lo cual podremos tener distintos formatos de la salida que ofrecerá la aplicación. En esta nueva versión, el sistema ofrecerá salida tanto en HTML, XML, como en XML parseado mediante una hoja XLS ofreciendo aspecto HTML.

5.- Desarrollo de un nuevo sistema

5.1.- Arquitectura Modelo vista controlador (MVC)

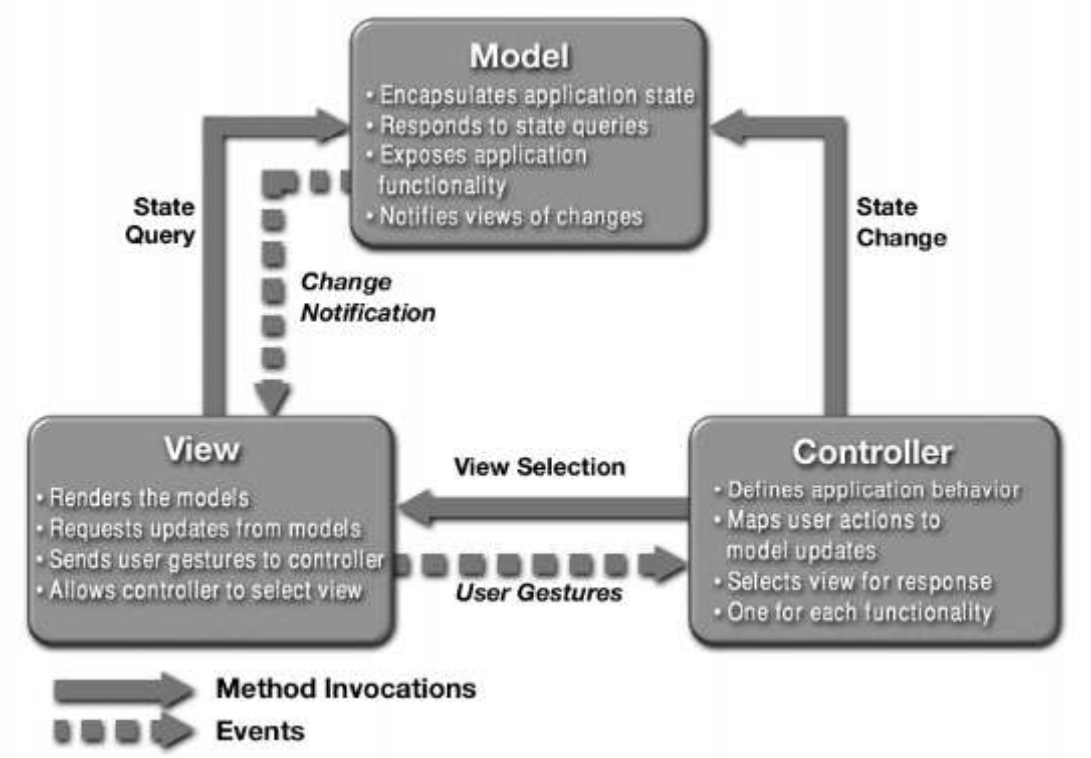
El modelo vista controlador (MVC) es el patrón de diseño recomendado para aplicaciones interactivas. MVC organiza una aplicación interactiva en tres módulos separados:

- 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.



La versión original de MVC nació en los laboratorios Seros y fue pensada para trabajar en aplicaciones gráficas para el entorno de Smaltalk.

5.1.1.- Modelo 1

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).

5.1.2.- Model 2

El modelo 2 es una variente 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.

5.2.- Análisis de los frameworks MVC existentes en el mercado

En este punto lo primero fue hacer un estudio de los distintos frameworks que existen en el mercado, comparando sus características para elegir el mejor framework en el que realizar la implementación del nuevo Chasqui.

5.2.1.- FRAMEWORK WACT

Nombre: Web Application Component Toolkit
Website: http://www.phpwact.org/
Licencia: Open LGPL Licence (http://www.gnu.org/copyleft/lesser.html)

Descripción general:

Es un proyecto open-source disponible en sourceforge.net, en el cual actualmente hay nueve personas en el grupo de desarrollo. Solo existen dos versiones alpha publicadas, 0.1alpha y 0.2alpha. Siendo versiones alpha, todavía contienen algunos errores conocidos y probablemente habrá errores que se desconocen.

Es un framework basado en patrones modulares, que ayuda a implementar el patrón MVC. En la página web del framework, existe una gran cantidad de información sobre la implementación de MVC en PHP.

Tiene una amplia colección de clases diseñadas para el patrón MVC en su API, clases como Controlador principal, Controlador de botones, de páginas, Vista, Vista de formulario, y el Modelo de respuesta. Además, proporciona una serie de clases básicas para completar un MVC.

Principales características:

Uno de los puntos fuertes de este framework, como ya viene indicado por su nombre, es el manejo de componentes. Tiene un paquete con una gran variedad de componentes web, para ser usado junto con su motor de plantillas, que facilita muchísimo la generación de páginas dinámicas.

El motor de plantillas, quizá la parte más trabajado del este framework, diferente de la de otros framework de PHP, WACT implementa totalmente, su propia gestión de plantillas, que incluye entre sus características, dos maneras diferentes para incrustar expresiones en plantillas (con etiquetas, y sintaxis corta usando {}).

WACT maneja las plantillas en dos escenarios: tiempo de compilación y tiempo de ejecución. Durante tiempo de compliación, WACT parsea las plantillas, crea una representación en árbol de componentes, (estos componentes son los que hemos mencionado anteriormente) y con estos componentes se crea la correspondiente plantilla compilada, que es el código en script de PHP, listo para ejecutar.

En el segundo paso, en tiempo de ejecución, se invoca a las plantillas compiladas usando los APIs de WACT para generar páginas WEB. En el fichero de configuración config.ini, podemos configurar que no se fuerce la compilación (FORCECOMPILE = FALSE), normalmente esta opción esta activa durante el desarrollo de la aplicación, una vez la aplicación entra en producción, las plantillas no se cambiará, por lo tanto, sólo habrá tiempo de ejecución.

Otro punto interesante de WACT, es que se ha hecho hincapié en la seguridad de las aplicaciones Web, proporcionando recursos de seguridad.

Documentación:

WACT usa una documentación inline(entre líneas de códigos) de API bastante completa, sigue el estándar phpDoc, casi todas las clases vienen documentadas. Y la web oficial con estilo Wiki contiene las informaciones más conceptuales y ejemplos simples de uso.

Requisitos:

La versión 0.2alpha de WACT, requiere PHP 4.1.2 o versiones posteriores, está testeado con las versiones 4.1.2, 4.3.8, 4.3.9, 5.0.1 y 5.0.2.

Aplicación:

En sourceforge, ha tenido más de 7.000 descargas de la versión 0.1, y más de 5.000 descargas de la versión 0.2. Adicionalmente existe otro framework de gestión de contenidos (Content Management Framework) llamado LIMB, que está basado en WACT.

Resumen:

Ventajas:

- Proyecto grande, con visión a largo plazo.
- Potente motor de plantillas.
- Buena base para completar el patrón MVC
- Facilidad de gestión de componentes (Páginas, formularios, botones, etc)
- Buena y completa documentación, incluye partes la API y los conceptos del framework.
- Recursos para la seguridad de aplicación.
- Configuración externa de los códigos.
- Paquete para gestión de base de datos.

Inconvenientes:

- Pocos ejemplos, sólo hay uno en la web, y es de cómo usar WACT con su motor plantillas.
- El patrón MVC es incompleto.
- Dudas de que si funciona bien con PHP 5. (Aunque hay una versión de LIMB que está hecho con PHP5 y WACT 0.2alpha)
- Versión inestable y experimental.

5.2.2.- FRAMEWORK WASP

Nombre: Web Application Structure for PHP 5
Website: http://wasp.sourceforge.net/
Licencia: LGPL Licence (http://www.gnu.org/copyleft/lesser.html)

Descripción general:

Es un framework basado en programación orientado a objetos con PHP 5, el proyecto está registrado en SourceForge desde 6-Jun-2005, es bastante reciente. En la actualidad, cuenta con un equipo de siete desarrolladores, y la última versión, WASP 1.2 Released, publicada el 1-Feb-2006 es una versión estable.

Todo el framework se puede instalar como un simple paquete de PEAR, para crear una aplicación, tendría que reconfigurar el fichero build.xml, y usando Phing para crear Controlador, Vistas, Modelo (Base de datos), según la configuración de build.xml.

Principales características:

Se empaqueta como un modulo de de PEAR. Facilidad de instalación.
Uso de PEAR:DB, para gestión contra deferentes tipo de base de datos.
Uso de PEAR:HTML_Template_Flexy, gestión de plantillas.

Uso de Phing (http://phing.info), es un sistema de contracción de proyectos (Basado en Apache Ant), tiene la misma funcionalidad que GNU make, usa un fichero XML para crear los ficheros y clases de "tareas" en PHP para el framework. Proporciona configuración automatizada, generación de objeto de modelo de datos, generación de controlador y vista. Su filosofía sigue las lineas de Ruby on Rails para facilitar la creación de proyectos.

Generación de MVC automatizado, según el fichero build.xml.

Documentación y manual de usuario:

En la web oficial, hay un manual de instalación y configuración muy completa, para Windows y Unix/Linux. La documentación de API generado por phpDocumentor, es un poco escasa y no tiene mucho comentario.

Dispone un manual de desarrollo de una simple aplicación con WASP en http://www.onlamp.com/pub/a/php/2006/01/19/wasp_intro.html?page=1,

Requisitos:

Instalación de PHP 5 con PEAR, (En Windows se recomienda XAMPP).
Instalación de Phing.

Aplicación:

En sourceforge, ha tenido más de 300 descargas de la versión 1.2. Se ha usado en una cantidad de soluciones de producción interna desarrolladas por la empresa PongoMedia.

Resumen:

Ventajas:

- Totalmente orientado a objetos.
- Framework con enfoque para proyectos grandes, con visión a largo plazo. (Hay planificación de 3 versiones futuras que incluirá interesantes avances).
- PEAR:HTML_Template_Flexy para plantillas.
- Generación automatizada del patrón MVC.
- Configuraciones externas en ficheros .ini y XML
- Manual de instalación y configuración.
- Versión estable, de producción.

Inconvenientes:

- No ofrece componentes de aplicación Web ya diseñados en PHP. (Se incluirá una arquitectura de componentes en la siguiente versión 1.3, que estará disponible en la primavera de 2006)
- Incertidumbre sobre el MVC generado automáticamente.
- La API poco documentada

5.2.3.- ZOOP FRAMEWORK

Nombre: Zoop Framework
Website: http://zoopframework.com
Licencia: Zoop License

Se trata de un framework de código abierto (open source) bajo una licencia propia denominada "Zoop License" (más información sobre dicha licencia en http://zoopframework.com/ss.4/7/license.html), donde la empresa Supernerd LLC tiene el copyright. Dicha empresa tiene un producto comercial denominado Supersite el cual se basa en el desarrollo de Zoop.

El grupo de desarrollo de Zoop está formado por personas relacionadas a la empresa Supernerd. El producto es anunciado como un framework con cinco años de madurez, lo mismo que su versión comercial Supersite.

El nivel de documentación de este proyecto es bastante bajo. En la página web del proyecto solo existen unos pocos tutoriales básicos para reflejar lo que se puede

hacer con Zoop, pero hay muy poca documentación sobre la filosofía que sigue en su implementación.

Respecto a la parte del controlador, en el modelo MVC, solo existe una página (http://zoopframework.com/ss.4/3/The_Undercarriage.html) donde comenta como son procesadas las URLs por el framework. En ella es donde se trata sobre la filosofía de trabajo del framework, haciendo alusión a las "zoop zones" como forma de estructurar las aplicaciones web a construir. Debido a la poca documentación existente, aún falta por documentar la aclaración de lo que son las "zoop zones".

La API existente ha sido generada con phpDocumentor a partir del código fuente del framework en la cual se pueden ver los atributos y métodos de las clases existentes, pero no existe ningún tipo de descripción acerca de la utilidad de un método o el propósito de una clase. Por tanto, se puede decir que aunque exista API, en la actualidad dicha API no sirve de ayuda al programador debido a que no está documentada.

La propia página del proyecto cuenta con un foro donde los usuarios del framework pueden encontrar ayuda, aunque en el momento actual no hay más de 30 entradas en el foro, con un nivel técnico bastante bajo, donde casi todas las entradas son respecto a la instalación del propio framework.

Fuera del propio sitio web, no se ha podido encontrar ningún tutorial o comentario sobre el uso de este framework.

Zoop se basa en: PHP, Pear, Smarty y Prototype (una librería con funciones JavaScript de propósito general).

Zoop corre sobre PHP 4.3 y sobre PHP 5.1, sin embargo con PHP 4 no soporta más de 10 conexiones simultáneas y recomiendan el uso de motores de cacheado (http://zoopframework.com/ss.4/6827/Programming_Tips.html). Sobre PHP 5 dicha limitación se ha subsanado, aunque indican que para obtener altos rendimientos haría falta motores de cacheado tales como APC for PHP5.

Dentro de sus funcionalidades tenemos:

- Tratamiento de errores (incluyendo registro)
- Controlador frontal basado en la URL
- Uso de Smarty para crear la vista mediante plantillas
- GuiControls: Se basa en la filosofía de los webcontrols de .net. Permiten crear rápidamente formularios, incluyendo validación de campos en el lado del cliente y en el lado del servidor. Se pueden crear nuevos GuiControls según las necesidades del usuario. Ej: crear un GuiControls para pedir direcciones de e-mail en un formulario incluyendo la validación del campo introducido, el cual luego pueda ser usado por ejemplo desde una plantilla sin tener que escribir el código HTML correspondiente (de esta forma se fomenta la reutilización de código). Esta característica es una de las más importantes del framework.
- Conectividad con base de datos a través de PEAR:DB
- El framework ofrece múltiples funciones para hacer consultas, modificaciones y actualizaciones contra el modelo de datos sin tener la

necesidad de introducir código SQL. Ofrece resultado con varios tipos de formato.
- Alta potencia a nivel de formularios: permite paginación y búsqueda de resultados mediante una única línea de código (llamada a la función correspondiente del framework que implementa dicha característica).
- Componente MAIL, con soporte mejorado sobre las funciones nativas de envío de correo de PHP.
- Posibilidad de mostrar la salida en formato PDF
- Sesiones almacenadas en base de datos (en vez de en ficheros como hace PHP)
- Uso de XML para describir secuencias de navegación.

Ventajas:
- Uso de Smarty como motor de plantillas
- Uso de PEAR:DB para el acceso a base de datos
- Tratamiento avanzado de errores
- GuiControls
- Bajo tiempo de desarrollo en la creación de formularios debido a las múltiples ayudas que ofrece el framework en la creación y tratamiento de formularios
- Validación de entradas de usuario por parte del framework, las cuales pueden ser redefinidas si es necesario. Objetivo: creación de aplicaciones web robustas y seguras

Inconvenientes:
- Desarrollo dependiente de intereses comerciales de la empresa Supernerd
- Comunidad de desarrolladores dependientes de la empresa Supernerd
- No está documentado de forma clara y detallada el funcionamiento interno del framework
- Falta de documentación de componentes y de la API.
- Fuera del sitio web del proyecto, no se encuentra información sobre dicho framework, por lo que se llega a la conclusión que es poco usado.

Conclusión final:

Debido a la falta de documentación, tanto en el propio sitio web como en fuentes externas al proyecto, sería muy arriesgado basar una aplicación en dicho framework.

Por otra parte, en el apartado "Ventajas" se han detallado las características más atractivas del framework, las cuales serían bastante interesantes que el framework finalmente elegido las contemplase, debido a su gran utilidad.

5.2.4.- Cake PHP FRAMEWORK

Nombre:Cake PHP Framework
Website: http://cakephp.org/
Licencia: MIT

Surge en 2005 mediante una mínima versión de un framework PHP el cual fue desarrollado por ichal Tatarynowicz, el cual se dio cuenta de que era un buen framework y lo publicó bajo la licencia MIT.

Está basado en el framework Ruby on Rails.

A partir del 20 de diciembre de 2005 el framework lo mantiene una comunidad de desarrolladores llamada The Cake Software Foundation la cual está fundada en las Vegas. Esta fundación tiene 3 miembros y además 3 desarrolladores. Son los encargados de actualizar la documentación de realizar el desarrollo para corregir los bugs del framework, de responder a preguntas.

Desde el día 13 de Enero han salido 6 releases del framework lo cual indica que actualmente está mantenido y que se puede informar ante el descubrimiento de posibles bugs. Además cada release lleva un fichero con información de los nuevos cambios que se han introducido con respecto a la versión anterior. Por otro lado se puede observar que el framework tiene algunos bugs en los que está trabajando la fundación. Dicha fundación espera que la siguiente release ya sea una versión estable del framework.

Actualmente las personas que se encargan de solucionar los bugs reportados por la gente que usa el framework son voluntarios, ya que los miembros de la comunidad no tienen tiempo para solucionarlos, con lo cual no es seguro del todo que sea resuelto brevemente.

Compatible con PHP4 y PHP5.

Tiene un despachador de peticiones basado en URL.

Trabaja desde cualquier subdirectorio de un sitio web con poca o ninguna configuración de Apache involucrada.

Existe un manual de referencia de unas 55 páginas que habla de la filosofía del framework, de cómo implementa el MVC y de las distintas características que tiene. También dispone de una API en la que informa de las diferentes clases y métodos.

Además en la parte CakePHPWiki hay varios tutoriales con ejemplos de aplicaciones utilizando el framework, las cuales permiten crear pequeños ejemplos para entender el funcionamiento del framework.

La fundación está desarrollando además del framework una serie de proyectos utilizando el framework.

La verdad es que el framework dispone de bastante documentación.

Para el MVC proporciona un núcleo con unas clases implementadas para cada una de las capas de dicho patrón. Todas los controladores o modelos nuevos que se introduzcan deben heredar de dichas clases ubicadas en un directorio llamado cake. Se recomienda no tocar el contenido de dicho directorio. Las clases del modelo deben extender la clase AppModel y las clases del controlador deben de extender la clase AppController.

Para el modelo proporciona algunas utilidades para crear querys más rapidamente. La filosofía es crear una clase por cada tabla de la base de datos.

Para el controlador se define un controlador por cada vista. La redirecciones se realizan mediante la URL. Existe un fichero de routing que a partir de una URL concreta se le asigna un controlador específico y una acción dentro de dicho controlador. Los datos son pasados del controlador a su vista asociada a través de un array llamado data, mediante una función llamada set.

Para la vista, proporciona una serie de herramientas llamadas Helper, las cuales permiten realizar más rápidamente formularios Web. Hay Helpers HTML, AJAX, JavaScript. Además los Helpers permiten ayudar a la corrección del formato de los datos de entrada de los usuarios.

La configuración del framework se realiza mediante ficheros de configuración los cuales vienen en el directorio llamado app, que es el que es modificado para introducir las partes de la aplicación. Existe un fichero de configuración para la base de datos y un fichero de configuración global.

Además, es necesario modificar algunas cosas en el fichero de configuración de nuestro servidor Apache.

5.2.5.- ZNF FRAMEWORK

Nombre: ZNF Framework
Website: http://znf.zeronotice.com/
Licencia: LGPL Licence (http://www.gnu.org/copyleft/lesser.html)

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 está en disponible en ingles, frances, 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).

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, donde describe la filosofía del framework, el patrón MVC, descripción de las clases que componen el framework, instalación, y un tutorial.

Existen un dos de listas de correo, una a nivel general y otra para desarrolladores, cuyo histórico está disponible en la web.

Se puede descargar una demo para ver como funciona el framework.

La última versión, disponible desde Octubre de 2005 ha sido descargada 1000 veces. Adicionalmente la versión en desarrollo siempre se encuentra disponible a través del sistema Subversión.

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.

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 estandar de codificación de PEAR (http://pear.php.net/manual/en/standards.php).

Las funciones del controlador son las siguientes:

1. El cliente hace una petición la cual es recibida por el controlador
2. Se procesa la petición
3. Se parsea el fichero de configuración
4. Se comprueba que el usuario esta autorizado para acceder a esta zona
5. Se parsean y validan las entradas de los usuarios
6. Se interactua con el modelo
7. Y por ultimo se redirecciona a la vista.

Mediante los diagramas de UML, descritos en el manual de referencia, queda completamente claro cual es el flujo de instancias de objetos que se dan ante una petición.

Finalmente este framework ha sido el que se ha decidido utilizar 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.

5.3.- Instalación del framework ZNF

ZNF framework 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
```

5.4.- Estructura de directorios de ZNF framework

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 partes más importantes del paquete 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).

5.5.- Configuración de ZNF framework

ZNF usa al menos dos ficheros de configuración XML localizados en el directorio *config*, un fichero de configuración de la aplicación llamado *znf-app-config.xml* y otro de configuración del módulo con el nombre *znf-<nombremodulo>-config.xml* . Se pueden crear más ficheros de configuración para otros módulos.

El fichero de configuración de la aplicación es usado para definir los aspectos generales de la configuración de la aplicación, tales como los valores por por defecto que debe tomar la aplicación y los módulos que forman parte de ella.

El fichero de configuración de un módulo se usa para inicializar los recursos del framework, es decir, se indica como recoger los datos de entrada del cliente, hacia que clases hay que enviar las peticiones para interactuar con el modelo de datos y que vista

se muestra al cliente en cada caso. Con estos ficheros de configuración es posible manejar la aplicación sin modificar el código o las clases.

Para poder usar las clases del modelo de datos, el framework maneja un tercer fichero de configuración llamado *znf-db-config.xml*, el cual permite realizar la conexión a una base de datos relacional. Este fichero XML posee atributos tales como el gestor de base de datos que utiliza, el nombre de usuario para conectarse a la base de datos, su contraseña, el hostname donde está la base de datos y el nombre de la base de datos. Estos parámetros solo aparecen en dicho fichero y no lo hacen en ninguna parte del código. Con dicho fichero lo que se consigue es centralizar los parámetros de conexión a la base de datos. Con lo cual se tiene la ventaja de que si queremos cambiar el nombre de la base de datos, el usuario con el que se accede, el password , e incluso el gestor donde se ha implementado la base de datos, basta con cambiarlo en este fichero de configuración evitando tener que modificar nada en el código de la aplicación.

5.6.- Requisitos y funcionalidades a implementar

Debido a que nuestro sistema se iba a implementar desde el principio, no se ha podido implementar toda la funcionalidad del Chasqui, si no que se ha hecho una parte del sistema. Se ha implementado solo algunas funcionalidades de Chasqui y se ha realizado el diseño basado en el patón MVC el cual sirve de base para terminar el futuro Chasqui, únicamente se han de implementar las funcionalidades que faltan siguiendo el diseño que se ha hecho.

Para el nuevo sistema se ha utilizado exactamente la misma base de datos que existe para el actual Chasqui. Además el aspecto de la interfaz gráfica ha pasado a un segundo plano no siendo muy vistosa, para de este modo poder realizar la máxima funcionalidad posible.

Listado de objetos

En un primer paso lo que se hizo fue mostrar un listado con todos los objetos virtuales, una funcionalidad muy sencilla con el propósito de entender mejor el funcionamiento del framework y de asegurarnos que realmente era válido para desarrollar el nuevo sistema.

Se creó un módulo llamado *objetoVirtual* y por consiguiente un fichero de configuración XML para este módulo llamado *znf-objetoVirtual-config.xml*. En ZNF para cada funcionalidad implementada en un módulo tiene una acción en su fichero de configuración, que indica el flujo que sigue la acción por las diferentes clases del framework. Para el listado de objetos virtuales:

```
<action
   path="objetosListado"
   type="ObjetoVirtual_Action_ObjetoVirtualListado">
   <forward
       name="show"
       path="ObjetoVirtual/Presentation/ObjetoVirtualListado.php"/>
</action>
```

Cuando se solicita el listado de objetos virtuales, se ejecuta la acción `objetosListado`, esta acción la procesa el disparador de peticiones y la envía a la clase

del controlador `ObjetoVirtual_Action_ObjetoVirtualListado` (situada en el directorio Action de ObjetoVirtual), la cual hereda de la clase `ZNF_Action_Action` implementada por el framework. En esta clase se ejecuta el método `execute` el cual interactúa con el modelo de datos mediante el uso de un objeto de la clase `ObjetoVirtual_Business_ObjetoVirtualDAO` (situada en el directorio Bussiness de ObjetoVirtual) . Esta clase opera con la base de datos y obtiene el listado de objetos virtuales mediante la función `getAllObjects()`. En la etiqueta forward del fichero de configuración se indica la página php a que el controlador controlador debe mostrar al usuario. Esta página es `ObjetoVirtualListado.php` (situada en el directorio Presentation de ObjetoVirtual). En esta página php se construye la vista que se presenta al usuario, para ello se utiliza el motor de plantillas Smarty. Esta página muestra al usuario el listado de objetos virtuales mediante la utilización de la plantilla `objVirtList.tpl` (situada en el directorio templates de ObjetoVirtual).

Login

La siguiente operación que se realizó fue la de iniciar sesión en la aplicación. Al igual que la anterior es una operación para familiarizarse con el comportamiento del framework, y se diferencia de la anterior en que el usuario introduce datos. Para ello se crea otro módulo llamado Usuario. La entrada de esta operación en el fichero de configuración es la siguiente:

```
    <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"/>


    <form-bean
      name="loginForm"
      type="Usuario_Action_LoginForm"/>
```

Cuando el usuario solicita hacer login se ejecuta la acción login y el disparador de peticiones redirige a la página `Login.php` (situada en el directorio Presentation del módulo Usuario). Aquí el usuario debe introducir el nombre de usuario y la contraseña

para registrarse en la aplicación. Además tiene una opción que puede marcar por si desea ver la salida en código XML en lugar de en HTML. Esta página php utiliza la plantilla `login.tpl` (situada en el directorio templates del módulo Usuario). En `nextPath` se indica que la siguiente acción a ejecutar es `loginSubmit`. En la acción `loginSubmit` hay 2 partes, por un lado se valida la entrada del usuario mediante `loginForm` y por otro lado se realiza la acción de loguearse en la aplicación mediante la clase `Usuario_Action_Login` (situada en el directorio Action del módulo Usuario). La clase `loginForm` hereda de la clase `ZNF_Action_ActionForm` implementada en el framework. Sirve para validar si el usuario ha introducido los datos obligatorios. Para ello utiliza las funciones `populate` para obtener los datos introducidos por el usuario, y `validate` para validar dichos datos. Si hay error se redirige a la página en la que se introducen los datos. En caso contrario se pasa a ejecutar la acción de loguearse. Para que se ejecute el método `validate`, es necesario que en el fichero de configuración el atributo `validate` sea igual a true. Como `loginForm` hereda de la clase `ZNF_Action_ActionForm`, es necesario que en el fichero de configuración (*znf-objetoVirtual-config.xml)* haya una entrada del tipo `form-bean` que indica cual es la clase del framework en la que se ha implementado la acción correspondiente al formulario `loginForm`.

Una vez que se ha validado la entrada del usuario se realiza la acción de loguearse mediante la clase del controlador `Usuario_Action_Login` (situada en el directorio Action de Usuario) que hereda de la clase `ZNF_Action_Action` implementada en el framework. Se ejecuta el metodo execute que utiliza un objeto de la clase `Usuario_Business_UsuarioDAO` (situada en el directorio Bussiness de Usuario). La clase `Usuario_Business_UsuarioDAO` pertenece al modelo y mediante el método `authUser` comprueba en la base de datos si el nombre de usuario y el password introducidos son correctos. Si el login es correcto, y el usuario no marcó la opción de ver la salida en XML, se ejecuta la acción `loginSuccess` que muestra al usuario la página `LoginSuccess.php` (situada en el directorio Presentation). Esta página utiliza la plantilla `loginSuccess.tpl` (situada en el directorio templates). Si el usuario marcó la opción de ver la salida en XML, y el login es correcto, se ejecuta la acción `loginSuccessXml` que muestra al usuario LoginSuccessXml.php con código XML.

En caso de que el login no sea correcto se ejecuta la acción `failure` que redirige a la página `Login.php` para que pueda volver a realizarse el login.

Navegación por objetos virtuales

Esta es una funcionalidad que existe en el Chasqui 2, y por tanto, fue la primera funcionalidad del antiguo Chasqui que se implementó en nuestro nuevo sistema. Esta funcionalidad consiste en navegar por los distintos objetos virtuales viendo sus atributos numéricos, atributos de texto, recursos y metadatos. Los objetos están ordenados por identificador y hay botones para poder ir al siguiente objeto, al anterior, al primero y al último.

La entrada de esta operación en el fichero de configuración es la siguiente:

```
<action
     path="objetosNavegar"
     type="ObjetoVirtual_Action_ObjetoVirtualNavegacion">
    <forward
           name="mostrarNavegar"
```

```
path="ObjetoVirtual/Presentation/ObjetoVirtualNavegacion.php"/>
</action>
```

Cuando el usuario solicita la navegación por objetos se ejecuta la acción `objetosNavegar`. Esta acción además lleva consigo una variable de tipo `REQUEST` llamada `posicion` que indica la posición del objeto virtual que queremos consultar. Una vez que se ha elegido navegar por un objeto virtual, el disparador de peticiones envía la petición a la clase `ObjetoVirtual_Action_ObjetoVirtualNavegacion` (situada en el directorio Action) la cual hereda de la clase `ZNF_Action_Action` que es una clase que viene implementada por el framework. En esta clase se ejecuta el método `execute`, y es dentro de este método donde se interactúa con el modelo de datos. Este método utiliza un objeto de la clase `ObjetoVirtual_Business_ObjetoVirtualDAO`, que es la clase que realiza operaciones con la base de datos. Mediante la posición que ocupa el objeto virtual, se realizan diferentes llamadas a métodos de esta clase para obtener los atributos numéricos, los atributos de texto, los recursos y los metadatos del objeto virtual correspondiente. Todos estos atributos se pasan a la vista mediante variables de tipo `REQUEST`. Posteriormente se ejecuta la acción `mostrarNavegar`, que muestra al usuario la página `ObjetoVirtualNavegacion.php` (situada en el directorio Presentation de ObjetoVirtual). Par mostrar los diferentes atributos del objeto virtual, esta página utiliza la plantilla `objVirtNaveg.tpl` (situada en el directorio templates).

Dentro de esta página es donde aparecen los botones de anterior, siguiente, primero y último que permiten realizar la navegación. Al pulsar uno de estos botones se vuelve a invocar a la acción `objetosNavegar`, modificando el valor de la variable `posicion`. De nuevo se llamada al método `execute` de la clase `ObjetoVirtual_Action_ObjetoVirtualNavegacion` y se vuelve a mostrar al usuario la página `ObjetoVirtualNavegacion.php` con las características del nuevo objeto virtual seleccionado.

Ir a un objeto virtual

La siguiente funcionalidad a implementar, también existente en el Chasqui 2, es la de ir a un objeto virtual. El usuario introduce el identificador del objeto virtual que desea ver, y si existe algún objeto con dicho identificador, se muestra. Además la página que muestra el objeto virtual es la misma que la de la navegación, de tal forma, que un vez encontrado el objeto virtual que se desea, se puede navegar por los siguientes objetos virtuales.

La entrada de esta operación en el fichero de configuración es la siguiente:

```
<action
  path="IrAObjeto"
  forward="ObjetoVirtual/Presentation/ObjetoBusqueda.php"
  nextPath="objetoBusquedaSubmit"/>

<action
  path="objetoBusquedaSubmit"
  type="ObjetoVirtual_Action_IrAObjeto"
  name="objBusquedaForm"
  input="IrAObjeto"
  validate="true">
  <forward
    name="success"
```

```
    path="ObjetoVirtual/Presentation/ObjetoVirtualNavegacion.php"/>
  <forward
   name="successXML"
   path="ObjetoVirtual/Presentation/ObjetoVirtualNavegacionXML.php"/>

   <forward
     name="error"
     path="IrAObjeto"/>
</action>

<form-bean
      name="objBusquedaForm"
      type="ObjetoVirtual_Action_ObjetoBusquedaForm"/>
```

Cuando el usuario solicita ir a un objeto virtual, se lanza la acción `IrAObjeto` y el disparador de peticiones redirige a la página `ObjetoBusqueda.php` (situada en el directorio Presentation del módulo ObjetoVirtual). En esta página el usuario introduce el identificador del objeto virtual al que quiere ir. También puede marcar una opción para ver la salida en XML en lugar de en HTML. Esta página php utiliza la plantilla `objIrAOV.tpl` (situada en el directorio templates del módulo ObjetoVirtual). En el atributo `nextPath` se indica que la siguiente acción a ejecutar es `objetoBusquedaSubmit`. En la acción `objetoBusquedaSubmit` hay 2 partes, por un lado se valida la entrada del usuario mediante `objBusquedaForm` y por otro lado se realiza la acción de buscar el objeto virtual en la base de datos mediante la clase `ObjetoVirtual_Action_IrAObjeto` (situada en el directorio Action del módulo ObjetoVirtual). La acción `objBusquedaForm` hereda de la clase `ZNF_Action_ActionForm` implementada por el propio framework. Sirve para validar si el usuario ha introducido los datos obligatorios. Para ello utiliza las funciones `populate` para obtener los datos introducidos por el usuario, y `validate` para validar dichos datos. Si hay error se redirige a la página en la que se introducen los datos (`ObjetoBusqueda.php`). En caso contrario se pasa a ejecutar la acción de buscar el objeto virtual con el identificador introducido por el usuario. Para que se ejecute el método `validate`, es necesario que en el fichero de configuración el atributo `validate` sea igual a true. Como la clase `objBusquedaForm` hereda de la clase `ZNF_Action_ActionForm`, es necesario que en el fichero de configuración (*znf-objetoVirtual-config.xml)* haya una entrada del tipo `form-bean` que indica cual es la clase del framework en la que se ha implementado la acción correspondiente al formulario `objBusquedaForm`. En este caso se puede ver que la acción `objBusquedaForm`, se implementa en la clase `ObjetoVirtual_Action_ObjetoBusquedaForm` (que está situada en el directorio Action del módulo ObjetoVirtual).

Una vez que se ha validado la entrada del usuario, pasa a ejecutarse la acción de buscar el objeto correspondiente en la base de datos. Esto se realiza mediante el método `execute` de la clase `ObjetoVirtual_Action_IrAObjeto` (situada en el directorio Action del módulo ObjetoVirtual). Esta clase hereda de la clase `ZNF_Action_Action` la cual está implementada por el framework. En este método `execute`, es donde se accede al modelo de datos, utilizando métodos de la clase `ObjetoVirtual_Business_ObjetoVirtualDAO` (situada en el directorio Bussiness del módulo ObjetoVirtual). Se comprueba si existe algún objeto con dicho identificador. Si no existe, se vuelve a mostrar al usuario la página `ObjetoBusqueda.php`. Si existe, se obtiene la posición que ocupa dicho objeto virtual, se obtienen los atributos, los recursos

y los metadatos del objeto mediante llamadas a métodos de la clase `ObjetoVirtual_Business_ObjetoVirtualDAO`. Posteriormente, si no se ha marcado la opción de ver la salida en XML, se muestra al usuario la página `ObjetoVirtualNavegacion.php`, que se corresponde con la misma vista que en la operación de navegar por objetos virtuales. En esta página además de los botones primeo, anterior, siguiente y último, se ha añadido un botón “Ir a un objeto” que al pulsarlo ejecuta la acción `IrAObjeto`.

Si por el contrario el usuario marcó la opción de ver la salida en XML, se muestra al usuario la página `ObjetoVirtualNavegacionXML.php` que muestra los atributos del objeto en formato XML.

Búsqueda de objetos virtuales

Esta funcionalidad también se ha implementado en el Chasqui 2. Consiste en buscar objetos virtuales rellenando una plantilla de búsqueda. Los criterios que se pueden utilizar para buscar objetos virtuales son los atributos de texto, los atributos numéricos, los recursos y los metadatos.

La entrada de esta operación en el fichero de configuración es la siguiente:

```
<action
      path="objetoBuscar"
      type="ObjetoVirtual_Action_ObtenerDatos">
      <forward
      name="FormularioBusqueda"
      path="ObjetoVirtual/Presentation/BuscarObjeto.php"/>
 </action>

 <action
      path="BuscaObjetoSubmit"
      type="ObjetoVirtual_Action_BuscarObjeto"
      name="BuscarObjetoForm"
      input="objetoBuscar"
      validate="true">
     <forward
        name="success"
        path="ObjetoVirtual/Presentation/ResultBusqueda.php"/>
     <forward
        name="error"
        path="objetoBuscar"/>
    </action>

    <action
      path="enlaceNavegar"
      type="ObjetoVirtual_Action_EnlaceNavegacion">
      <forward
        name="success"
        path="ObjetoVirtual/Presentation/ObjetoVirtualNavegacion.php"/>
      <forward
        name="error"
        path="ObjetoVirtual/Presentation/ResultBusqueda.php"/>
    </action>

<form-bean
      name="BuscarObjetoForm"
      type="ObjetoVirtual_Action_BuscarObjetoForm"/>
```

Cuando el usuario solicita realizar una búsqueda, se lanza la acción `objetoBuscar` y el disparador de peticiones ejecuta el método `execute` de la clase `ObjetoVirtual_Action_ObtenerDatos` (situada en el directorio Action del módulo ObjetoVirtual). Esta clase interactúa con el modelo de datos mediante la clase `ObjetoVirtual_Business_ObjetoVirtualDAO` (situada en el directorio Bussiness de ObjetoVirtual) . Mediante métodos de la clase mencionada anteriormente, se obtienen algunos valores para ser introducidos como parte de la plantilla de búsqueda. Estos valores son los distintos psoibles valores que existen para algunos metadatos, atributos numéricos y de texto o recursos. De esta forma en la plantilla de búsqueda, se puede seleccionar el valor de un comboBox en lugar de escribirlo manualmente. Posteriormente se muestra al usuario la plantilla de búsqueda que se corresponde con la página `BuscarObjeto.php`. Esta página php utiliza la plantilla `BusquedaOV.tpl` (situada en el directorio templates del módulo ObjetoVirtual). Después de que el usuario haya rellenado los criterios con los que desea realizar la búsqueda de objetos, se ejecuta la acción `BuscaObjetoSubmit`. En la acción `BuscaObjetoSubmit` hay 2 partes, por un lado se valida la entrada del usuario mediante `BuscarObjetoForm` y por otro lado se realiza la acción de buscar en la base de datos los objetos virtuales que cumplen los criterios introducidos mediante la clase `ObjetoVirtual_Action_BuscarObjeto` (situada en el directorio Action del módulo ObjetoVirtual). La acción `BuscaObjetoSubmit` hereda de la clase `ZNF_Action_ActionForm` implementada por el propio framework. Sirve para validar si el usuario ha introducido los datos obligatorios. Para ello utiliza las funciones `populate` para obtener los datos introducidos por el usuario, y `validate` para validar dichos datos. Si hay error se redirige a la plantilla de búsqueda (`BuscarObjeto.php`) y se muestra la causa del error. En caso contrario se pasa a ejecutar la acción de buscarlos objetos virtuales que cumplen con los criterios introducidos por el usuario. Para que se ejecute el método `validate`, es necesario que en el fichero de configuración el atributo `validate` sea igual a true. Como la clase que implementa `BuscarObjetoForm` hereda de la clase `ZNF_Action_ActionForm`, es necesario que en el fichero de configuración (*znf-objetoVirtual-config.xml)* haya una entrada del tipo `form-bean` que indica cual es la clase del framework en la que se ha implementado la acción correspondiente al formulario `BuscarObjetoForm`. En este caso se puede ver que la acción `BuscarObjetoForm`, se implementa en la clase `bjetoVirtual_Action_BuscarObjetoForm` (que está situada en el directorio Action del módulo ObjetoVirtual).

Una vez que se ha validado la entrada del usuario, pasa a ejecutarse la acción de buscar en la base de datos los objetos que cumplen las condiciones . Esto se realiza mediante el método `execute` de la clase `ObjetoVirtual_Action_BuscarObjeto` (situada en el directorio Action del módulo ObjetoVirtual). Esta clase hereda de la clase `ZNF_Action_Action` implementada por el framework. En este método `execute`, es donde se accede al modelo de datos, utilizando métodos de la clase `ObjetoVirtual_Business_ObjetoVirtualDAO` (situada en el directorio Bussiness del módulo ObjetoVirtual). Se buscan los objetos que cumplen los criterios introducidos mediante un llamada al método `BuscarObjetos` de la clase `ObjetoVirtual_Business_ObjetoVirtualDAO`. Posteriormente, tanto si se ha encontrado algún objeto, como si no, se muestra al usuario la página `ResultBusqueda.php` (situada en el directorio Presentation del módulo objetoVirtual). Esta página utiliza la plantilla `ResBusqueda.tpl` (situada en el directorio templates de ObjetoVirtual). En la página se muestran los identificadores y las descripciones de los

objetos virtuales encontrados. Si se pulsa con el ratón en el identificador de algún objeto se ejecuta la acción `enlaceNavegar` . Esto hace que se ejecute el método `execute` de la clase `ObjetoVirtual_Action_EnlaceNavegacion` (situada en el directorio Action). En esta método se obtienen los atributos, recursos y metadatos del objeto sobre el que se ha pinchado con el ratón. Esto se hace mediante llamadas a métodos de la clase `ObjetoVirtual_Business_ObjetoVirtualDAO`. Entonces se muestra al usuario la página `ObjetoVirtualNavegacion.php` que es la página de navegación por objetos virtuales, de la que ya hemos hablado en otras funcionalidades.

5.6.1.- Modificaciones en el framework

Han sido necesarias hacer ciertas modificaciones en el propio código fuente del framework ZNF para poder cumplir nuestros requisitos.

Todas las modificaciones que se han realizado han sido documentadas en un fichero de texto ubicado en el directorio /docs de nuestro código fuente. El objetivo de dicha documentación es facilitar el cambio a una nueva release del framework.

5.7.- Introducción y aplicación de la tecnología AJAX

5.7.1.- Introducción a AJAX

AJAX, acrónimo de *Asynchronous JavaScript And XML* (JavaScript y XML asíncronos), es una técnica de desarrollo web para crear aplicaciones interactivas. Éstas se ejecutan en el cliente, es decir, en el navegador del usuario, y mantiene comunicación asíncrona con el servidor en segundo plano. De esta forma es posible realizar cambios sobre la misma página sin necesidad de recargarla. Esto significa aumentar la interactividad, velocidad y usabilidad en la misma.

AJAX es una combinación de tres tecnologías ya existentes: XHTML (o HTML) y hojas de estilos en cascada (CSS) para el diseño web; Document Object Model (DOM) accedido con un lenguaje de scripting como JavaScript para mostrar e interactuar dinámicamente con la información presentada. Se usa el objeto XMLHttpRequest para intercambiar datos asincrónicamente con el servidor web; XML es el formato más usado para la transferencia de vuelta al servidor, aunque cualquier otro formato puede funcionar, como JSON (*JavaScript Object Notation*), EBML (*Extensible Binary Meta Language*) o incluso texto plano.

Realmente, AJAX no constituye una tecnología nueva en sí, sino que es un término que engloba a un grupo de éstas que trabajan conjuntamente.

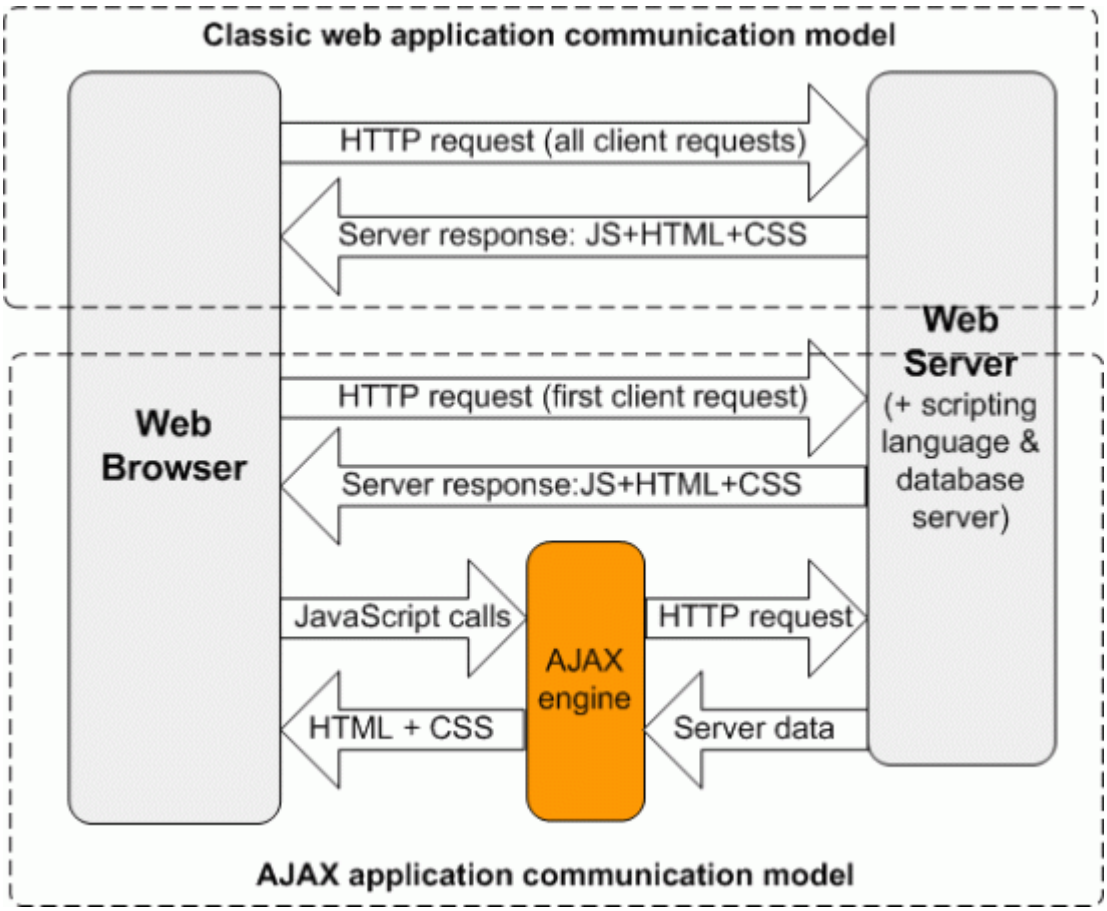
5.7.2.- Cómo funciona AJAX

Cuando una aplicación usa AJAX, se añade una nueva capa al modelo de comunicación. En una aplicación web clásica, la comunicación entre el cliente (el navegador) y el servidor web se realiza directamente: cuando un usuario solicita una página, el servidor le mandará el código HTML y CSS entero de una sola vez. Después

de que el usuario rellena un formulario y lo entrega, el servidor procesa la información, reconstruye la página, y vuelve a mandar la página entera al cliente (navegador).

Cuando se usa AJAX, la página se carga enteramente de una sola vez. Entre los códigos de HTML y CSS, también se descarga algunos ficheros de JavaScript: el motor de AJAX (librería de AJAX en JavaScript). Todas las solicitudes de datos que van contra el servidor serán enviadas como llamadas de JavaScript a este motor. Posteriormente, el motor AJAX solicita la información al servidor web asíncronamente. Después únicamente la parte mínima necesaria de la página será devuelta al navegador del usuario, y el motor AJAX la muestra sin recargar enteramente la página. Esto supone una interfaz mucho más sensible, ya que solo se transfiere la información necesaria entre el cliente y servidor, en lugar de la página entera. Esto produce la sensación de que la información se muestra inmediatamente, y hace que las aplicaciones web se comporten de manera más similar a las de standalone.

Para la ilustración de la comunicación entre el cliente (navegador) y el servidor remoto y la diferencia entre una aplicación web clásica y una con AJAX, se muestra el siguiente diagrama:



En el corazón de AJAX, está el motor AJAX, que no es nada más que códigos de JavaScript a los que se hace referencia y usa el objeto XMLHttpRequest. Este último es un objeto en JavaScript que permite enviar, recibir y procesar solicitudes HTTP con el servidor sin necesidad de refrescar la página entera.

En las aplicaciones de AJAX, las solicitudes HTTP para obtener datos pueden hacerse completamente en segundo plano, sin que el usuario note ninguna interrupción. Esto significa que el usuario puede continuar trabajando y usando la aplicación, mientras recibe la información solicitada desde el servidor. El objeto XMLHttpRequest está implementado como un objeto de ActiveX en Internet Explorer 6.x, y como un objeto nativo de JavaScript en la mayoría de navegadores modernos (FireFox, Safari, etc.)

Aunque añadir una capa extra a cualquier tipo de modelo puede significar un aumento del tiempo de respuesta, aquí se produce una excepción. En la nueva capa (el motor AJAX), el tiempo de respuesta es tan corto que la interfaz del usuario parece mucho más conectada a la lógica de la aplicación, cosa que solo ocurre en aplicaciones de tipo standalone. Además, el usuario ya no tiene que esperar hasta que se recargue la página entera.

5.7.3.- Aplicación de AJAX en Chasqui

La aplicación de AJAX en Chasqui, nos puede aportar ventajas tales como:

- Disminuir la carga del servidor: el tráfico de datos que se transmite entre el cliente y el servidor.
- Incrementar la usabilidad de Chasqui: nos permite implementar una interfaz más inteligente y rápida.
- Proporcionar más interacción entre el usuario y la aplicación.

Podemos aplicar AJAX en el menú de navegación de Chasqui, la navegación de objetos, donde se necesita un mayor grado de interacción y transferencia de datos. En nuestro prototipo, hemos implementado con AJAX una versión más pequeña de navegación por objetos que en Chasqui2. En esta implementación, se usa XML como el formato para la transferencia de los datos, XSLT para mostrar la información obtenida en XML dentro un diseño HTML, y finalmente unos métodos de JavaScript que se encargan de crear objetos XMLHttpRequest para gestionar los datos con el servidor.

La parte más importante de una aplicación AJAX es el motor AJAX, en este caso, hemos usado un motor (librería) simple llamado "net.js" que viene en el famoso libro "Ajax In Action". Es una librería de tamaño reducido que proporciona un método "cross-browser" de llamadas con XMLHttpRequest, es decir, la librería se encarga de hacer diferentes tratamientos con XMLHttpRequest dependiendo del navegador usado. Por ejemplo, ActiveX para IE6.x, y JavaScript nativo para los navegadores basado en el núcleo Gecko (Mozilla, FireFox, Camino, Safari, etc). Con lo cual, nos ahorra mucho tiempo en hacer compatible nuestra aplicación para diferentes navegadores, y podemos centrarnos en cómo tratar la información obtenida en XML.

En el directorio "js" de nuestro proyecto "chasqui_znf" está la librería "net.js", y tambien otro fichero JavaScript llamado "xsltAjax.js", que contiene siguientes métodos:

- *function LoadXMLXSLTDoc (urlXML, reloadXML, urlXSL, reloadXSL, elementID){ … }* :

Es el método que se encarga de solicitar un XML y un XSL en las URLs que se le indican, y luego lo recoloca en el nodo *elementID*. Los booleanos *reloadXML*

y *reloadXSL* indican si es necesario re-solicitar el XML o XSL, porque puede haber casos en que sólo necesitemos solicitar uno de estos dos.

- *function onXMLLoad (){ …} :*

Es el método que se encarga de solicitar/cargar el XML deseado.

- *function onXSLLoad (){ …}:*

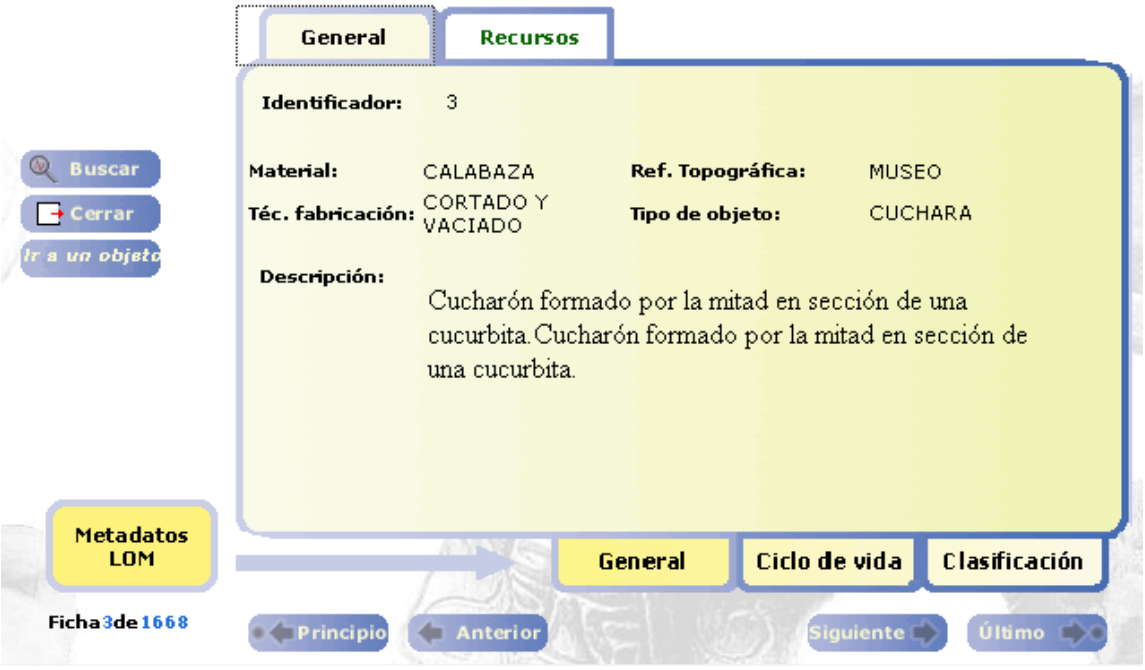
Es el método que se encarga de solicitar/cargar el XSL deseado.

- *function doXSLT() { …}:*

Es el método que muestra los datos obtenidos en XML con la plantilla XSL, y finalmente lo recoloca en el nodo HTML que se le había indicado anteriormente. Este método es invocado después cargar los XML y XSL correspondientes.

Estos métodos son los usados principalmente en nuestro "chasqui_znf", en particular, el primero *LoadXMLXSLTDoc*, ya que los otros tres son llamados desde éste.

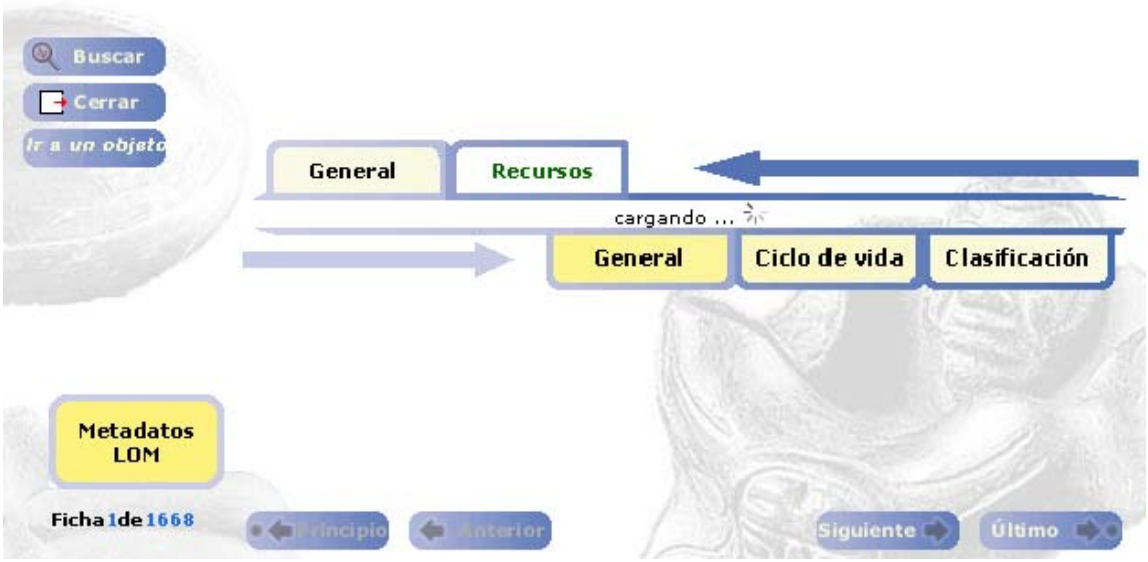
A continuación, vamos a explicar con un ejemplo (una operación) concreto, en el cual hemos aplicado AJAX con los métodos anteriores de JavaScript. En nuestra aplicación "chasqui_znf", se puede seleccionar una opción de navegación por objetos con AJAX. Si se selecciona, aparecerá la siguiente pantalla:



En esta página de arriba, hay cuatro operaciones (botones) implementadas con AJAX, que son las fichas "General" y "Recursos" que están situados encima del cuadro de información (el cuadro que está en el medio). Cuando pinchamos uno de estos botones, se invocará el método *LoadXMLXSLTDoc* mencionado anteriormente con los parámetros correspondientes, luego se solicitará al servidor los datos pedidos, y una vez

obtenidos todos, los muestra en el cuadro que está en el medio, sin refrescar la página completamente.

Además, para hacerse una idea más cercana del efecto de AJAX, hemos metido un icono y un mensaje mientras se está solicitando los datos al servidor en segundo plano. Véase la imagen siguiente:



Con esta misma filosofía, podemos extender el uso de AJAX en nuestro Chasqui. Por ejemplo, el menú de navegación, las fichas de datos de los objetos virtuales, así como en todos los sitios que interviene la comunicación masiva de datos entre el cliente y el servidor, sin la necesidad de tener que refrescar la página entera.

5.7.4.- Otras aplicaciones con AJAX y sus compatibilidades

AJAX ha sido una revolución para las aplicaciones web, con él se puede implementar aplicaciones web muy ricas, incluso desarrollar aplicaciones web que tengan efectos similares que una aplicación standalone, por ejemplo, el efecto drag&drop. Actualmente, ya existe una gran variedad de aplicaciones implementadas con AJAX que son muy útiles y populares, entre los más famosos están Google Maps y Gmail.

Existe un amplio grupo de navegadores que soportan AJAX, aunque este soporte dependerá de las características que el navegador permita. La mayoría de navegadores que soportan *JavaScript*, también soportan directamente AJAX. Con la excepción de que, en Internet Explorer 6.0, se necesita el ActiveX activado, ya que en este navegador el objeto *XMLHttpRequest* está implementado junto con ActiveX.

Para implementar la aplicación "*cross-browser*" con AJAX, se recomienda usar alguna librería de *JavaScript* que soporte ya previamente a varios navegadores, tal como ***Prototype*** (una librería de código abierto), ***xajax*** para *PHP*, etc. En nuestra

implementación, hemos usado una pequeña librería (*net.js*) disponible en el libro de "AJAX EN ACTION".

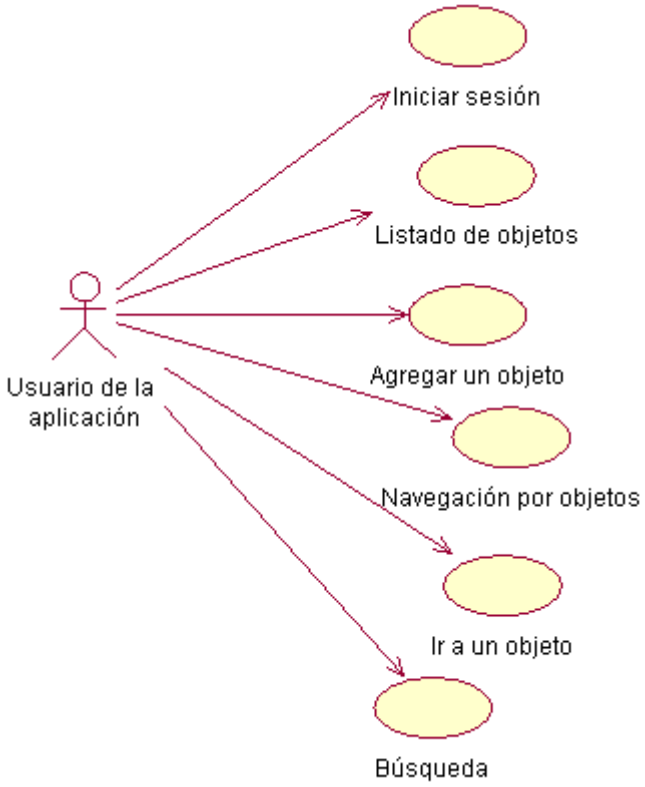
Navegadores que permiten usar AJAX:

- Microsoft Internet Explorer para Windows versión 5.0 y superiores, así como los navegadores basados en él.
- Navegadores basados en Gecko como Mozilla, Mozilla Firefox, SeaMonkey, Camino, Flock, Epiphany, Galeon y Netscape versión 7.1 y superiores.
- Navegadores con el API KHTML versión 3.2 y superiores, incluyendo Konqueror versión 3.2 y superiores, Apple Safari versión 1.2 y superiores, y el Web Browser for S60 de Nokia tercera generación y posteriores.
- Opera versión 8.0 y superiores, incluyendo Opera Mobile Browser versión 8.0 y superiores.

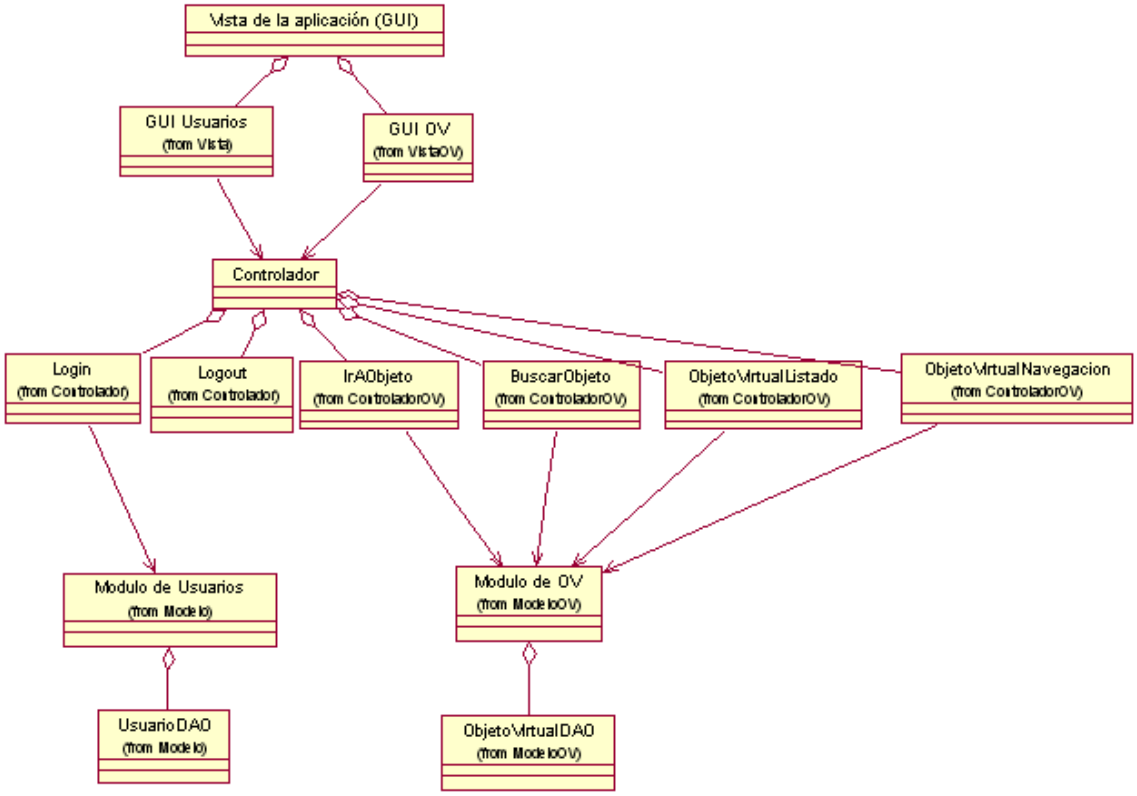
5.8.- Diagramas UML

A continuación se muestran los diagramas UML de casos de uso, de clases y de secuencia relativos a las funcionalidades implementadas en la nueva aplicación, que son las que se han explicado en el apartado 5.6.

Diagrama de casos de uso de la aplicación

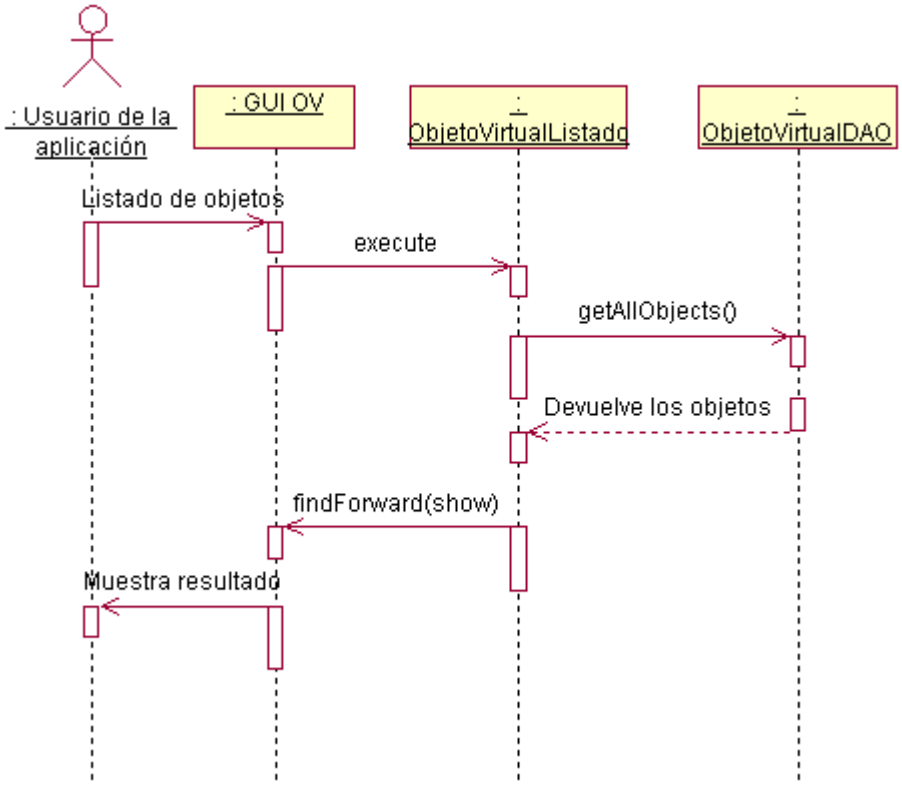


Diagramas de clases

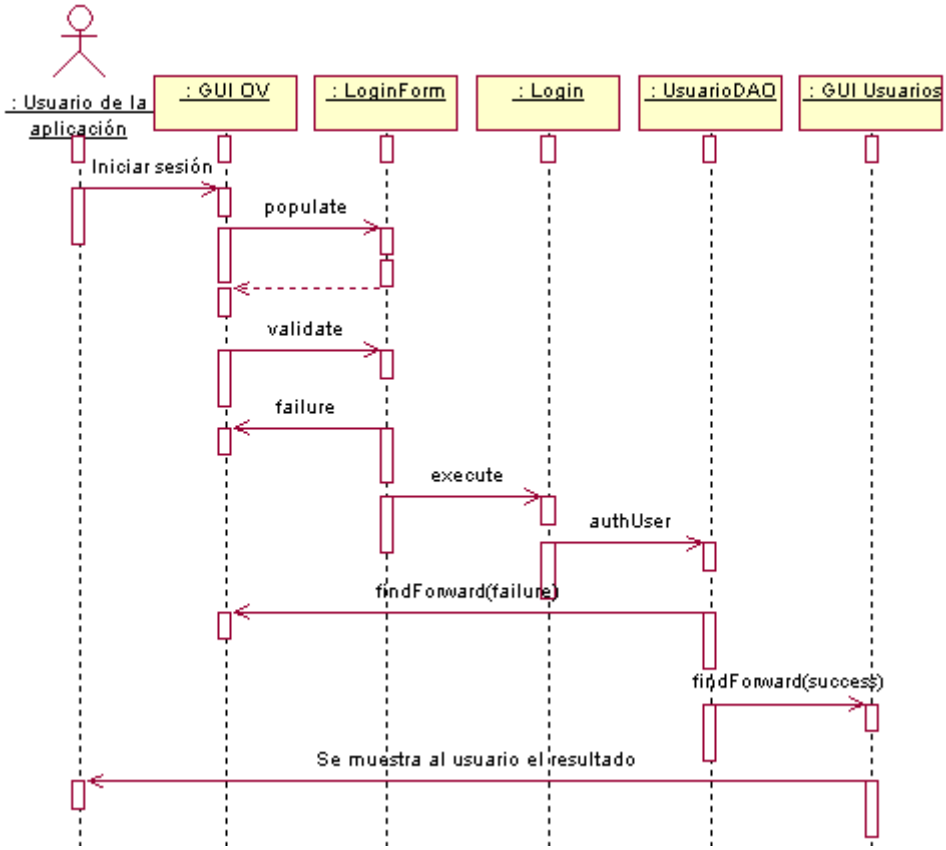


Diagramas de secuencia

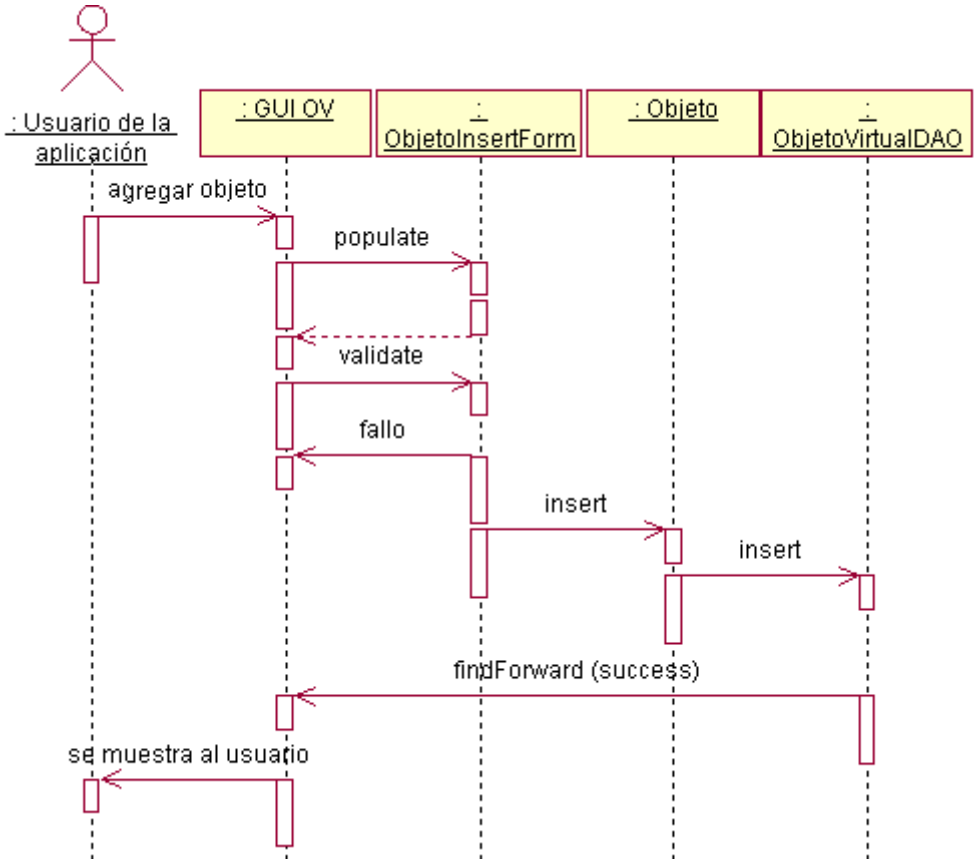
Listado de objetos



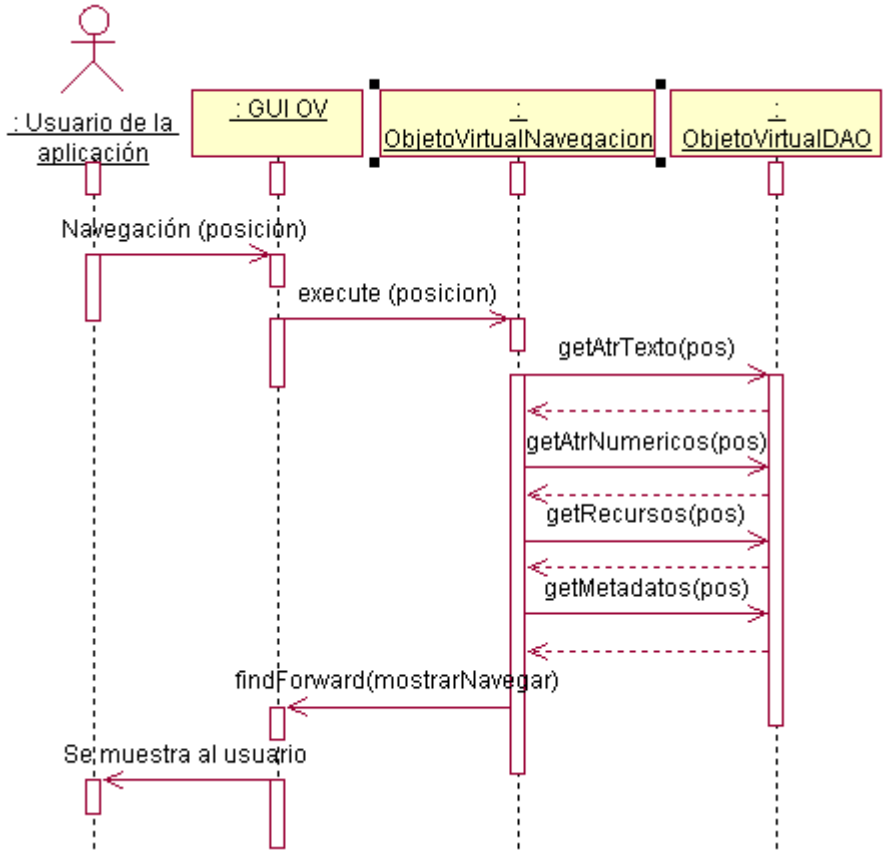
Iniciar sesión



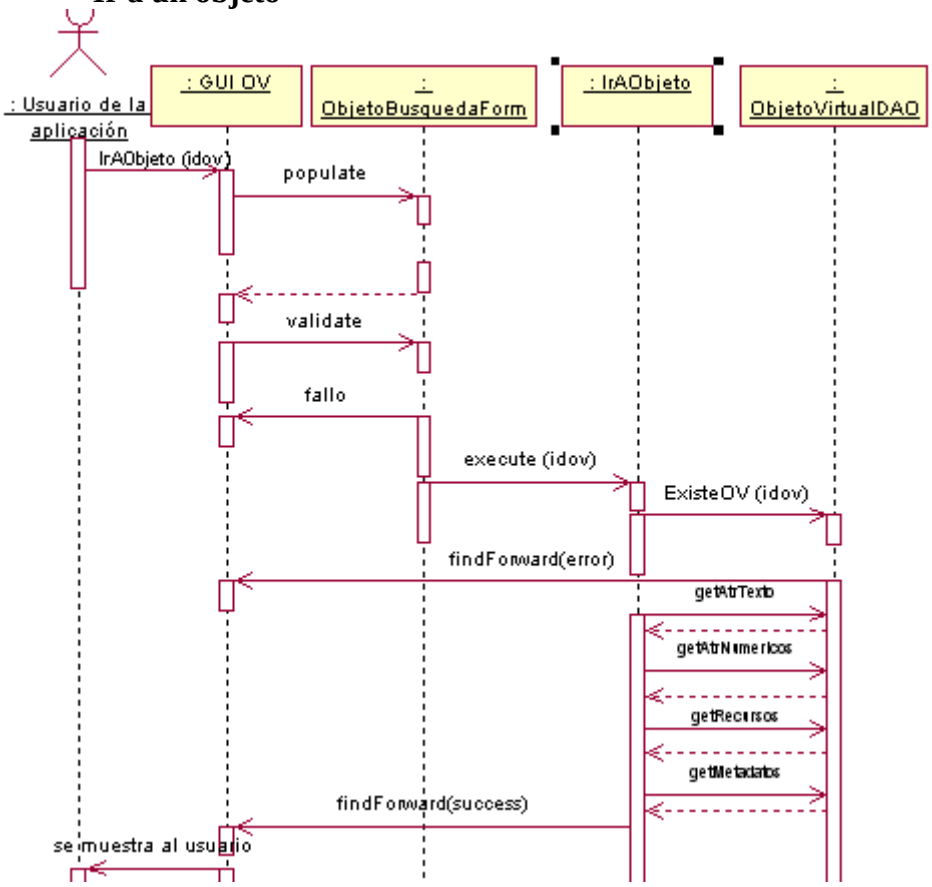
Insertar objeto



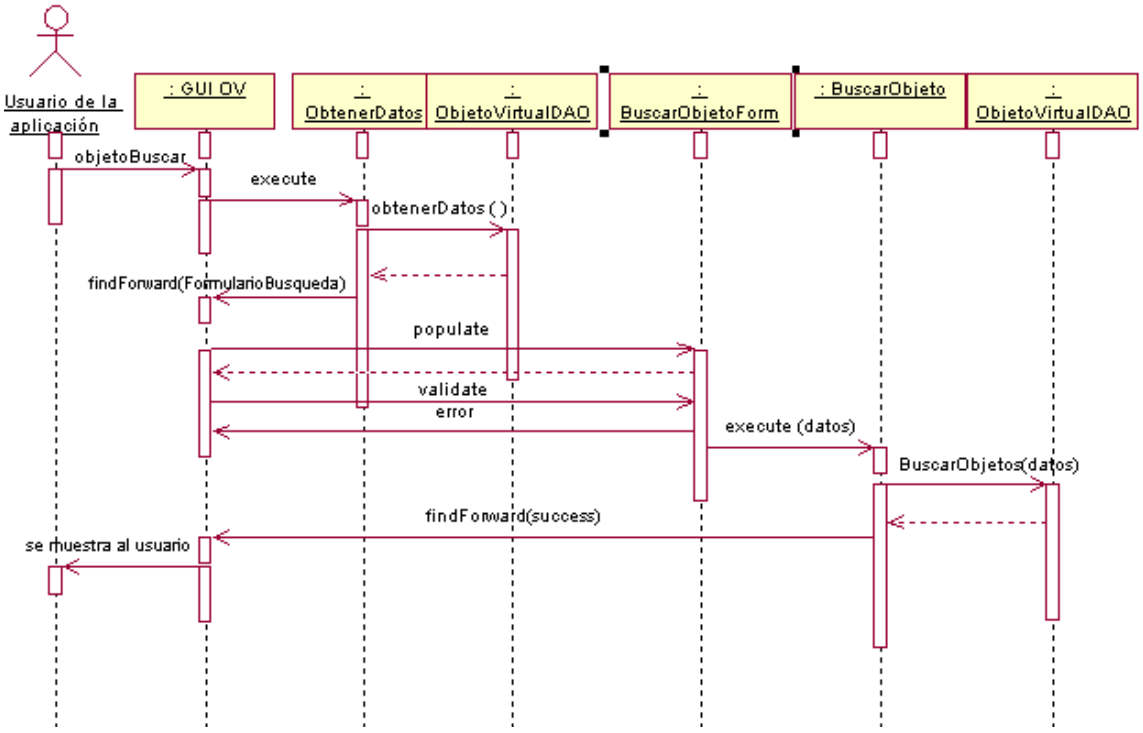
Navegación por objetos



Ir a un objeto



Búsqueda


Usuario de la aplicación
: GUI OV
: ObtenerDatos
: ObjetoVirtualDAO
: BuscarObjetoForm
: BuscarObjeto
: ObjetoVirtualDAO
objetoBuscar
execute
obtenerDatos ( )
findForward(FormularioBusqueda)
populate
validate
error
execute (datos)
BuscarObjetos(datos)
findForward(success)
se muestra al usuario

6.- Conclusiones y líneas de trabajo futuro

A pesar de que la elección de crear un nuevo sistema desde cero ha limitado 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 las funcionalidades omitidas en este primer prototipo así como la implementación de nuevas funcionalidades de una forma más rápida, clara y estructura.

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 de aprendizaje 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.

7.- Bibliografía

1. IEEE/LTSC: http://ltsc.ieee.org/
2. IMS: www.imsglobal.org
3. ADL: www.adlnet.org
4. L. Mortimer: http://www.learningcircuits.org/2002/apr2002/mortimer.html
5. LOM: http://ltsc.ieee.org/wg12/
6. Dublin Core: http://dublincore.org/
7. ARIADNE: http://www.ariadne.ac.uk/
8. Smarty: http://smarty.php.net/
9. PEAR: http://pear.php.net/

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”.

Sheng Fang

José Antonio Villegas López

José Corbacho Gil