Desarrollo de componentes para Joomla 1.5

Development components for joomla 1.5

Un developpement de composants pour joomla 1.5

Recepción: 18-05-2010
Evaluación: 23-07-2010
Aceptación: 18-08-2010
Artículo de Investigación Científica

*Jorge Gabriel Hoyos Pineda

Resumen
En este documento se realiza una descripción del proceso de desarrollo de un componente para Joomla 1.5. Para ilustrar el proceso se utiliza un ejemplo de sistema académico que administra información sobre los estudiantes. El modelo de desarrollo de componentes utilizado es el patrón de diseño Modelo-Vista-Controlador (MVC) y hace uso de las clases proporcionadas por el API de Joomla para ese modelo.

Palabras clave: Componente, MVC, Desarrollo Web.

Abstract
This document is a description of the development process of a component for Joomla 1.5. To illustrate the process it is used an example of the academic system that manages information about students. The component development model used is the design pattern Model-View-Controller (MVC) and uses the types provided by the API of Joomla for this model.

Key words: Component, MVC, Web Development

Résumé
Dans ce document on réalise une description du processus de développement d'un composant pour Joomla 1.5. Pour illustrer le processus on utilise un exemple de système académique qui administre l'information sur les étudiants. Le modèle de développement de composants utilisé est le patron de dessin(conception) Modelo-Vista-Controlador (MVC) et utilise des classes fournies par l'API de Joomla pour ce modèle.

Mots-clés: Le Composant, MVC, un Développement Web.

Introducción

La masificación de la Internet ha permitido la aparición de nuevos espacios y nuevas formas de relación entre los usuarios de la red. Uno de los fenómenos más interesantes es el surgimiento de un nuevo enfoque en el desarrollo de sitios Web que ahora incorporan elementos que posibilitan a sus usuarios un mayor protagonismo y el desarrollo de nuevos mecanismos de comunicación y colaboración con otros usuarios que comparten intereses comunes (Hoyos Pineda, 2009). Es en esta nueva visión de la web en la que se apalanca la llamada Web 2.0.

Este tipo de sitios es desarrollado en la actualidad mediante la utilización de los llamados Sistemas de Gestión de

* Magíster en Ciencias de la Información y las Comunicaciones. Docente Facultad de Ingeniería de Sistemas, Universidad Santo Tomás, Seccional Tunja. jhoyos@ustatunja.edu.co

Fuente Fotográfica: *http://www.themza.com/template-photos/Ski+Club-16.jpg*

Contenidos (Content Manager System, CMS) (Robertson, 2003), como Joomla, tal vez el más conocido y utilizado. Pero más allá de la gestión de información, que es su tarea básica, un CMS debe ser compatible con las tecnologías más usadas en el mundo de la Web 2.0 y permitir la incorporación de un gran número de componentes desarrollados para obtener funcionalidades específicas.

El desarrollo de componentes para Joomla 1.5 tiene como finalidad incorporar nuevos elementos y funcionalidades a los portales desarrollados con este CMS, ya que incluso se puede llegar a desarrollar una aplicación completa que queda embebida en el mismo.

Desarrollo

En la primera parte se describen los principios del modelo vista controlador (MVC), que sirve de base para el modelo de desarrollo de componentes propuesto por la comunidad Joomla, y los elementos principales del API que posibilitan el desarrollo de componentes compatibles con este framework. En el resto del documento se describe la forma como dicho modelo es aplicado a la creación de un componente que responda a necesidades específicas.

• *Joomla y el Modelo Vista Controlador (MVC)*

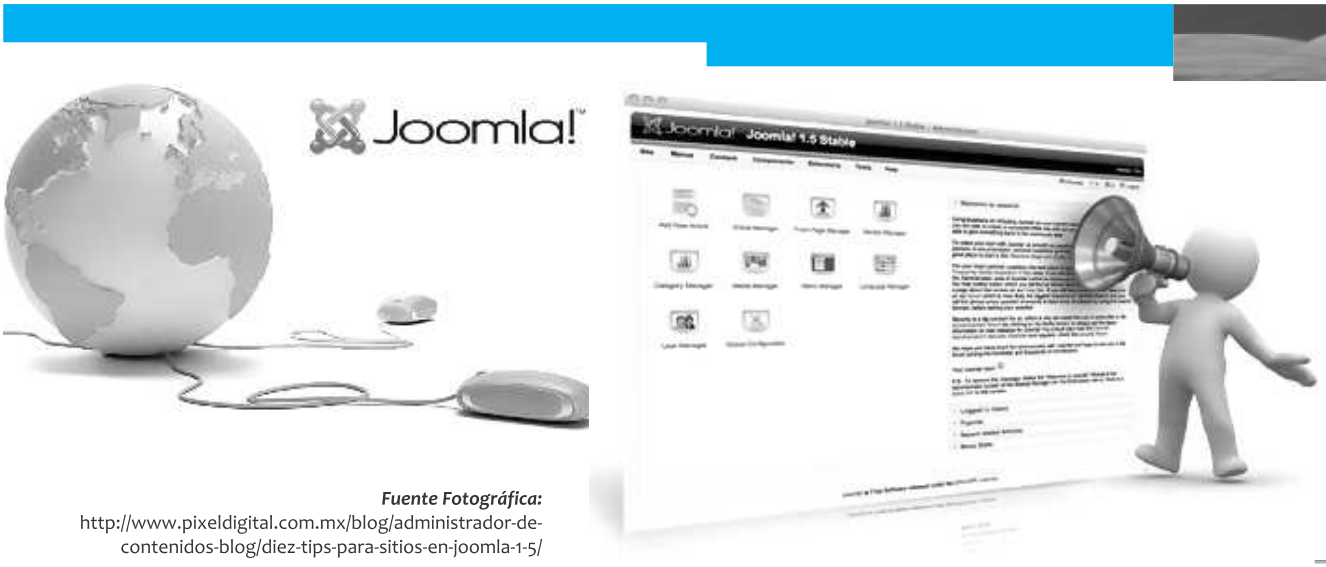
Como ejemplo se propone la construcción de un componente que facilite las labores de administración de la información académica de los estudiantes.

• *El Modelo MVC*

El Modelo Vista Controlador, también conocido como MVC es tal vez el más utilizado en desarrollo web. Este patrón de diseño consiste en la división del desarrollo en tres capas de aplicación. La Primera conocida como Modelo, es la capa encargada de la lógica de negocio y administración de los recursos de datos. La segunda llamada Controlador, que sirve como receptor de las peticiones realizadas por el usuario a través de una interfaz gráfica de usuario y como reorientador de las mismas hacia las unidades específicas del modelo encargadas de procesar o atender la petición del usuario. La última capa referida como Vista es la encargada de proporcionar una interfaz que permita la interacción del usuario con el sistema y la visualización de información proporcionada por el modelo.

Las tres capas están conformadas por un conjunto de clases que proporcionan una funcionalidad bien definida (Bascon, 2009).

In Vestigium Ire - No. 3 - Vol. 1 - Año 2010 - pp. 141 - 152 - ISSN 2011-9836



Fuente Fotográfica:
http://www.pixeldigital.com.mx/blog/administrador-de-contenidos-blog/diez-tips-para-sitios-en-joomla-1-5/

Fuente:
http://www.somoslibres.org/imagenes/joomla15.jpg

El modelo representa la información presente en el mundo real relativa al ámbito de la aplicación. Por ejemplo para el caso de estudio el modelo debería contener la representación de las entidades estudiantes, asignaturas, cursos, docentes.

Las vistas se encargan de presentarle al usuario la información proporcionada por el modelo. Una vista está asociada a un modelo, y varias vistas pueden estar asociadas a un mismo modelo. La vista se actualiza en respuesta a cambios en el modelo gracias a un sistema de notificaciones generadas por el mismo (Deacon, 2009).

Por su parte, el controlador, como su nombre lo dice debe controlar el flujo de eventos provenientes de la interfaz de usuario, y de convertir esos eventos en operaciones que son solicitadas y procesadas por el modelo. Además tiene la responsabilidad de administrar las vistas, es decir puede decidir qué vista utilizar y cuándo las mismas se cierran o se abren. En la Figura 1 se muestra un esquema de la relación existente entre las tres capas.

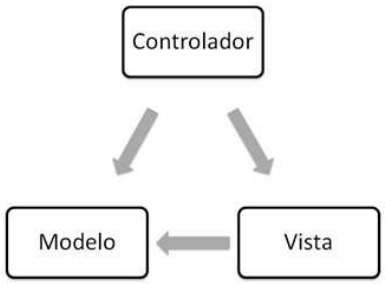


FIGURA 1. ESQUEMA DEL PATRÓN DE DISEÑO MODELO-VISTA-CONTROLADOR
Fuente: Autor

Por último, hay que mencionar que aunque el uso del patrón MVC proporciona algunos beneficios, como la facilidad para aplicar cambios a las interfaces y la posibilidad de definir diferentes vistas para un mismo modelo, también trae consigo algunas desventajas como el aumento de la complejidad y del nivel de acoplamiento entre la vista y el modelo (Merlino, 2009).

- *El API de Joomla*

Joomla es sinónimo de software libre y trabajo en equipo. Es importante saber que Joomla está construido sobre un API comúnmente conocido como el "Joomla-Framework" (Framework - Joomla! Documentation, 2009). Este framework está compuesto por un conjunto de clases y funciones agrupadas en paquetes que hacen posible que un gran número de

Fuente:
http://www.themesandmods.com/uploads/2010/03/Joomla-1.5-SEO.jpg

componentes, módulos y plantillas funcionen en forma armónica. Este mismo framework es el que hace posible que cientos o quizás miles de personas hayan desarrollado componentes para atender necesidades específicas, y que los mismos puedan ser integrados en forma transparente con este núcleo central.

Adicionalmente, proporciona un conjunto de funcionalidades, como por ejemplo un esquema de seguridad que protege la aplicación de accesos indebidos, razón por la cual no es necesario que el desarrollador se preocupe por ese aspecto cuando está diseñando un nuevo componente.

Para el caso específico de la implementación del patrón MVC, el API proporciona tres clases que facilitan la construcción de la arquitectura propuesta de tres capas, como son JModel, JView y JController, las cuales se encuentran ubicadas en el paquete joomla.application.component.

a) *JModel*. Funciona como factoría o fábrica de los objetos requeridos por la aplicación y proporciona un buen conjunto de funciones de soporte. Sirve como clase base para la implementación de las clases que representan el modelo en una implementación MVC.
b) *Jview*. Proporciona los métodos necesarios para el despliegue de los datos sobre una interfaz. Sirve como clase base para la implementación de las clases que representan la vista en una implementación MVC.
c) *JController*. Proporciona la funcionalidad básica y es la encargada de la administración de las vistas. Sirve como clase base para la implementación de las clases que representan el controlador en una implementación MVC.

• *Desarrollo del Componente*

Joomla es mucho más que un gestor de contenidos, aunque éste sea el uso más frecuente que se le da a este CMS. Una de las características más importantes de Joomla es que permite la integración con aplicaciones complejas, podría decirse que cualquier aplicación escrita en PHP puede ser convertida en uno de sus componentes.

Joomla proporciona dos ambientes de trabajo diferentes conocidos como "front end" y "back end". El "front end" está representado por la interfaz del sitio web que puede ser accedida por cualquier usuario en forma anónima o mediante autenticación con un nombre de usuario y clave por un usuario registrado. El "back end" está representado por la interfaz que permite al usuario administrador ejecutar tareas de configuración, mantenimiento, generación de contenidos, y otras propias de dicho rol (Graf, 2008).

En el caso de los componentes es usual definir las dos partes, "front end" y "back end", aunque su estructura de archivos es

In Vestigium Ire - No. 3 - Vol. 1 - Año 2010 - pp. 141 - 152 - ISSN 2011-9836

Foto: http://www.themza.com/template-photos/Naturaleza+Verde-21.jpg

muy similar. La diferencia principal es la ubicación de la carpeta del componente. En el caso del "front end" la carpeta estará ubicada dentro de la carpeta *components* del sitio web, mientras que la carpeta correspondiente al "back end" estará ubicada dentro de la carpeta *administrator /components* del sitio.

En la tarea de implementar un nuevo componente, lo primero que hay que observar es la convención de denominación para el componente, según la cual el componente se debe nombrar con el sufijo "com_" seguido de un carácter de subrayado y el nombre distintivo del componente. Como ejemplo si el componente va a manejar información académica podría llamarse "com_ academico". A su vez este mismo nombre debe utilizarse como nombre del directorio que va a contener el conjunto de archivos que conforman el componente, tanto en el "front end" como en el "back end". A partir del directorio principal del componente se crean el directorio "models" y el directorio "views", para contener las clases representativas del modelo y de las vistas respectivamente.

Desarrollo del Back End del componente

Aunque el "back end" puede incluir una interfaz de administración del componente, ésta no será necesaria a menos que se requiera configurar parámetros u otros elementos aplicables a todo el componente. Aquí resulta interesante determinar el lugar dónde van a ser almacenados los datos que utiliza el componente.

El primer paso es crear las tablas en la base de datos de Joomla que van a permitir almacenar la información de cada una de las entidades de datos definidas para el componente. Asumiendo que la entidad de datos que se va a trabajar es "*Estudiantes*" y que se aplica la convención de denominación para las tablas consistente en el prefijo de la base de datos (*jos_ por defecto en joomla*) más el nombre del componente y el nombre de la entidad de datos, separados por un carácter de subrayado, el primer paso es correr el script de creación de la tabla "*jos_academico_ estudiantes*" en la consola SQL o mediante otra interfaz de administración de la base de datos.

```
CREATE TABLE `ciu_academico_estudiantes`
(
 `id` int(11) NOT NULL auto_increment,
 `codigo` int(11) NOT NULL,
 `nombre` varchar(255) NOT NULL,
 PRIMARY KEY  (`id`)
);
```

Una vez creada físicamente la tabla en la base de datos, se necesita crear el mecanismo que permite al componente el intercambio de información con la base de datos. Para atender esta necesidad el API de Joomla incluye la clase JTable que proporciona la funcionalidad necesaria para crear, leer, actualizar y borrar registros sobre una tabla de la base de datos. Lo único que se requiere es implementar una clase derivada de la clase JTable, en cuya definición se debe guardar correspondencia entre el nombre de los atributos de la clase y el nombre de los campos de la tabla. En este caso también existen reglas de denominación que indican que el nombre

de la clase debe estar formado por el prefijo "*Table*" más el nombre de la entidad de datos. Adicionalmente el archivo contenedor de la definición de la clase debe estar ubicado en el directorio "*tables*", creado previamente en la estructura del back end del componente. Para el ejemplo se debe definir la clase "*TableEstudiante*" que debe estar almacenada en el archivo de nombre "*estudiante.php*" dentro del directorio */administrator/components /com_academico/tables*, y cuyo contenido es el siguiente:

```
<?php
defined('_JEXEC') or die('Restricted access');
class TableEstudiante extends JTable
{
var $id = null;
var $codigo = null;
var $nombre = null;
function __construct(&$db)
{
   parent::__construct( '#__estudiantes', 'id', $db );
}
}
?>
```

Fuente: *http://somosiphone.com/wp-content/uploads/home_top_welcome.png*

Como se puede observar en el segmento del código anterior, se trata de una definición típica de una clase en php. La línea *defined('_JEXEC') or die('Restricted access')* es utilizada en Joomla como mecanismo de seguridad que impide el acceso o ejecución directa desde una aplicación diferente a Joomla. Ya en la definición de la clase hay que mencionar que el parámetro del método constructor corresponde a una referencia a la base de datos vinculada al sitio, adicionalmente cuando se hace el llamado al constructor de la clase base se utilizan tres parámetros. El primero indica el nombre de la tabla (donde *#__* corresponde al prefijo de la tabla), el segundo el nombre de la llave principal y el tercero nuevamente la referencia a la base de datos.

En este punto la estructura de archivos del back end, será la mostrada en la Figura 2.

FIGURA 2. ESQUEMA DE LA ESTRUCTURA DE ARCHIVOS DEL BACK END
Fuente: Autor

• *Desarrollo del Front End del componente*

Existen diferentes propuestas sobre la forma de cómo se puede desarrollar un componente para ser integrado en Joomla. En (LEBLANC, 2007) se presenta un ejemplo de construcción de un componente sin seguir ningún modelo, por una parte construye la interfaz gráfica, que sería en este caso la vista, y otra que mezcla en un mismo sitio las operaciones propias del controlador con las definiciones del modelo. Posteriormente se recompone el diseño inicial para ajustarlo al modelo MVC.

In Vestigium Ire - No. 3 - Vol. 1 - Año 2010 - pp. 141 - 152 - ISSN 2011-9836

Un modelo similar es presentado en (KENNARD, 2007), donde se aplica el patrón MVC mediante la utilización de las clases del API de Joomla, JView, JModel y JController, y se hacen algunas recomendaciones para la definición de las clases derivadas de las anteriores, como son:

- En la construcción del modelo se debe tener en cuenta una convención especial utilizada para darle nombre a las clases que implementan el modelo, para garantizar que el modelo funcione en la forma correcta. Esta convención establece que el nombre de la clase debe estar formado por el nombre del componente, la palabra *"Model"* y el nombre del modelo, y que debe estar almacenada en un archivo cuyo nombre se corresponda con el del modelo y que debe estar ubicado en la carpeta *models*. Para el ejemplo el modelo se llamará *"AcademicoModelAcademico"*, y estará almacenado en un archivo de nombre "academico.php" en la carpeta *models* del componente.

```
<?php
defined('_JEXEC') or die('Acceso denegado');
jimport('joomla.application.component.model');

class AcademicoModelAcademico extends JModel
{
var $datos;

function consultar()
{
        if (empty($this->datos))
        {
                $query="SELECT * FROM #__academico_estudiantes";
                $this->datos=$this->_getList($query);
        }
        return $this->datos;
}
}
?>
```

En este caso el método *_getList* obtiene el conjunto de registros resultado de la sentencia SELECT contenida en la variable *query*, y lo almacena en la variable *datos* de la instancia del modelo.

Es importante mencionar que se pueden definir varios modelos, que normalmente van a estar asociados con las entidades de datos definidas para el componente. En la figura 3 se muestra el esquema de ejemplo.

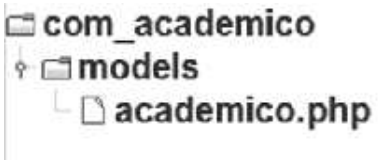


FIGURA 3. ESQUEMA DE LA ESTRUCTURA DE ARCHIVOS UTILIZADAS PARA LOS MODELOS
Fuente: Autor

- Por su parte las vistas requieren su propia carpeta dentro de la carpeta views del componente. La carpeta correspondiente a cada vista puede contener un archivo diferente por cada tipo de documento soportado por la vista. En este caso también se debe observar la convención especial para el nombre de la clase, el cual debe estar formado por el nombre del componente, la palabra *View* y el nombre de la vista, y es almacenado en un archivo de

Fuente: *http://www.scriptola.com/golf-club-template/*

Imagen: http://www.template-help.com/screenshots/19300/19316-joomb.jpg

nombre *"view.document Type.php"*, donde *documentType* corresponde al tipo de documento que puede ser FEED, HTML, PDF o RAW. Adicionalmente si la vista soporta el tipo de documento HTML se debe incluir una carpeta con nombre *tmpl*, la cual contiene los archivos *html* necesarios para desplegar la vista. Para el caso de ejemplo se contará con una vista principal que se llamará *"AcademicoView Académico"*, y estará almacenada en un archivo de nombre *"view.html.php"* en la carpeta *academico* que a su vez está contenida en la carpeta *views* del componente.

```
<?php
defined('_JEXEC') or die('Acceso restringido');
jimport('joomla.application.component.view');
class AcademicoViewAcademico extends JView
{
function display($tpl=null)
{
        $model=& $this->getModel();
        $datos= $model->consultar();
        $this->assignRef('datos', $datos);
        parent::display($tpl);
}
}
?>
```

Lo primero que realiza la función *display* es obtener una referencia al modelo asociado (en este caso el modelo *Académico*), luego ejecuta el método consultar definido en el modelo, el cual retorna un conjunto de datos que es almacenado en la variable *datos*, a partir de la cual se crea una referencia con el mismo nombre, que será utilizada posteriormente por el *template* o plantilla de visualización de los datos.

Adicionalmente en la carpeta *académico* se debe crear otra carpeta de nombre *"tmpl"* que contiene el archivo encargado de la visualización, para el ejemplo se llamará *"default.html"*. Al igual que en el caso de los modelos, también se pueden definir varias vistas, incluso con varios *templates* (esquemas de visualización), dependiendo de las necesidades de la aplicación.

```
<h1>Listado de Estudiantes</h1>
<form action="index.php" method="post">
<table border="0">
<tbody>
<?php
        $n = count($this->datos);
        for($i=0; $i<$n; $i++)
        {
        $row = & $this->datos[$i];
?>
<tr>
        <td><?php echo $row->codigo;
?></td>
        <td><?php echo $row->nombre;
?></td>
</tr>
<?php
        }
?>
</tbody>
</table>
</form>
```


In Vestigium Ire - No. 3 - Vol. 1 - Año 2010 - pp. 141 - 152 - ISSN 2011-9836

Como se puede observar este archivo contiene etiquetas comunes html, mezcladas con el código *php* que permite la generación dinámica de nuevas etiquetas.

En la figura 4 se muestra el esquema de ejemplo.

FIGURA 4. ESQUEMA DE LA ESTRUCTURA DE ARCHIVOS UTILIZADAS PARA LAS VISTAS
Fuente: Autor

- En el caso de los controladores, normalmente se encuentra un único controlador para el componente almacenado en un archivo de nombre *controller.php*, aunque es posible definir varios de ellos. Los controladores usan un mecanismo mediante el cual asocian el nombre de una tarea o *task*, con la ejecución del método del controlador que lleva el mismo nombre de la tarea. Es posible establecer el valor de la tarea mediante la variable *task* en una operación de tipo POST.

```
<?php
defined('_JEXEC') or die('Acceso Restringido');
jimport('joomla.application.component.controller');
class AcademicoController extends JController
{
function display()
{
        parent::display();
}
}
?>
```

Por último, es necesario crear en el directorio raíz del componente el archivo *academico.php* que es el encargado de crear una instancia del controlador e invocar el método "*redirect*" del mismo que es el encargado de cargar la vista correspondiente.

```
<?php
defined('_JEXEC') or die ('Acceso restringido');
require_once(JPATH_COMPONENT.DS.'controller.php');
$classname = 'AcademicoController';
$controller= new $classname();
$controller->execute(JRequest::getVar('task'));
$controller->redirect();
?>
```

En la Figura 5 se muestra la estructura general del "front end" del componente aplicando el modelo mencionado.

FIGURA 5. ESQUEMA DE LA ESTRUCTURA DE ARCHIVOS UTILIZADAS PARA EL COMPONENTE EN EL FRONTEND
Fuente: Autor

Fuente: http://www.templatemonster.com/joomla-templates/21599.html

FIGURA 6. INTERFAZ DEL FRONT END PARA LA VISUALIZACIÓN DEL COMPONENTE
Fuente: Autor

- ***Integración del componente con el portal***

En adición al gran número de tecnologías y herramientas de software orientadas al desarrollo de sitios Web 2.0, existe un elemento central que actúa como integrador de las anteriores, y que constituye el punto de acceso para los usuarios, el portal de información.

Visualizar el componente tal como se encuentra hasta este punto, solo requiere incluir el nombre del componente como valor del parámetro *option* de la url del sitio en el que está instalado, tal como se ilustra en la Figura 6.

La integración definitiva del componente al portal resulta un procedimiento muy sencillo, si el componente ha sido desarrollado siguiendo el modelo propuesto. En el módulo de administración del sitio se dispone de una utilidad que instala y registra el nuevo componente a partir de un archivo empaquetado, tal como se muestra en la Figura 7.

Una vez registrado lo único que resta es vincularlo a la opción de menú que servirá como puerta de entrada al componente.

D. Resultados

Una vez instalado el componente se tiene acceso a la funcionalidad definida, que para el caso de ejemplo es la de mostrar la información sobre estudiantes, almacenada en la base de datos. En este caso por ser un ejemplo muy sencillo, se debe considerar la previa alimentación de la tabla, ya que el componente lo único que hace es ejecutar una consulta sobre la misma. Sin embargo el modelo presentado sirve de partida para definir nuevas vistas y modelos que permitan realizar las diferentes operaciones con los datos como son inserción, actualización y eliminación. No sobra mencionar que las aplicaciones reales manejan un gran número de tablas, y que no hay ningún problema en vincularlas a un componente como el desarrollado en el ejemplo, siempre y cuando se sigan las convenciones mencionadas en especial en lo relativo al nombrado de las clases y archivos.

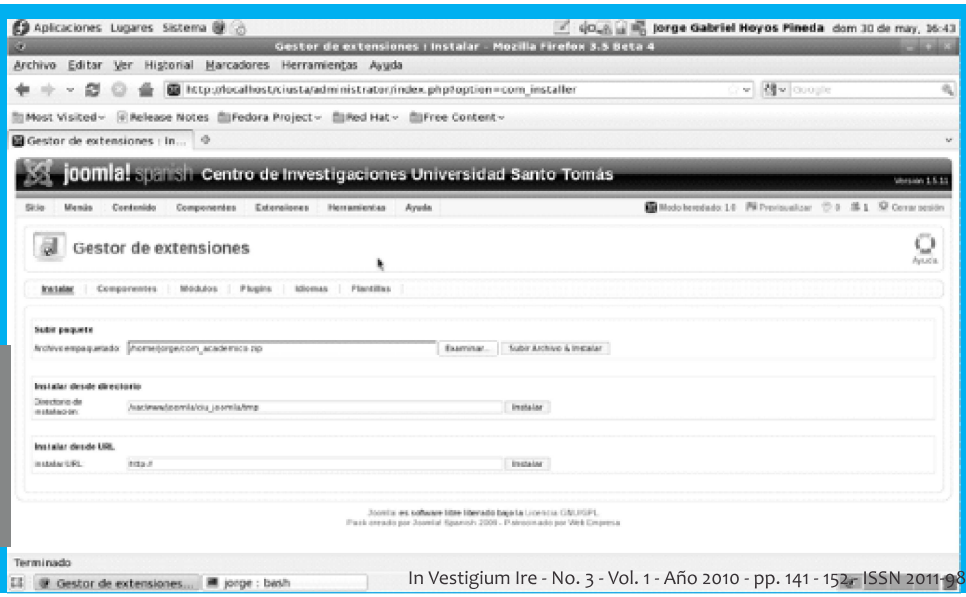


FIGURA 7. INTERFAZ DEL BACK END PARA LA INSTALACIÓN DEL COMPONENTE
Fuente: Autor

In Vestigium Ire - No. 3 - Vol. 1 - Año 2010 - pp. 141 - 152 - ISSN 2011-9836

Fuente Fotográfica: *http://www.joomlashack.com/blog/joomla-templates/292-joomlashack-releases-its-latest-native-joomla-15-template*

Conclusiones

Existe una gran variedad de tecnologías y herramientas, la mayoría de uso libre, que pueden ser utilizadas en la construcción de componentes, y que exigen del desarrollador ahondar en el conocimiento de las mismas y de toda su creatividad para combinarlas y sacar el mejor provecho de las mismas al hacerlas parte de sus aplicaciones. Por ejemplo Joomla, la herramienta utilizada para el desarrollo de que trata este documento, proporciona un modelo de desarrollo de componentes, las clases base necesarias para su implementación y un conjunto de reglas de nombrado que garantizan en gran medida la adecuada integración del componente con el portal Web.

Teniendo en cuenta la naturaleza de los componentes, es posible adicionar nuevas funcionalidades al componente desarrollado, o crear nuevos componentes que complementen al primero, atendiendo temas relacionados.

Fuente
http://www.yafaonline.com/portal/joomla/

Referencias

Bascon, E. (2009). El patrón de diseño Modelo-Vista-Controlador (MVC) y su implementación en Java Swing.

Deacon, J. (2009). Model-View-Controller (MVC) Architecture.

Retrieved November 6, 2009, from http://www.jdl.co.uk/briefings/mvc.html .Framework - Joomla!

Documentation. (2009). http://docs.joomla.org/Framework.

Graf, H. (2008). Building Websites with Joomla! 1.5. Birmingham: Packt.

Hoyos Pineda, J. G. (2009). Herramientas de desarrollo para la Web 2.0. Investigium Ire. Kennard, J. (2007).

Mastering Joomla! 1.5 Extensión and Framework Development.

Birmingham: Packt Publishing Ltda.

Larman, C. (2003). UML y Patrones. Madrid: Prentice Hall.

LeBlanc, J. (2007). Learning Joomla! 1.5 Extension Development.

Birmingham: Packt Publishing Ltd.Merlino, H. (2009).

Patrón de Diseño de Vistas Adaptables.

Robertson, J. S. (2003). What is a CMS? Retrieved May 12, 2009, from http://www.steptwo.com.au/papers/kmc_what/index.html.