Universidad Nacional Autónoma
De México

Facultad de Ciencias

Diseño y desarrollo de portales de servicios en *Plone*©

Trabajo profesional

Que para obtener el título de:
Licenciado en Ciencias de la Computación

P R E S E N T A :

Nombre del Alumno
Arturo Abimael Quijada Mendoza

Tutor
Dr. José de Jesús Galaviz Casas

2010

Hoja de Datos de Jurado
1. Datos del Alumno Quijada Mendoza Arturo Abimael 5529018180 Universidad Nacional Autónoma de México Facultad de Ciencias 09923545-5
2. Datos del tutor Dr. José de Jesús Galaviz Casas
3. Datos del sinodal 1 Dr. Sergio Rajsbaum Gorodezky
4. Datos del sinodal 2 Dra. Elisa Viso Gurovich
5. Datos del sinodal 3 Mat. Mónica Leñero Padierna
6. Datos del sinodal 4 L. en C.C. Mauricio Aldazosa Mariaca
7. Datos del trabajo escrito Diseño y desarrollo de portales de servicios en Plone 33 p 2010

Agradecimientos

En primer lugar me gustaría agradecer a mis papás, fue gracias a sus esfuerzos, sacrificios y apoyo que he podido llegar hasta aquí.

A mis hermanos que tantas noches soportaron el sonido de las teclas.

Al Dr. José Galaviz por la ayuda que me brindó para terminar este trabajo.

A Lu y Joel que siempre me han ayudado y apoyado, aunque a veces Lu se enojaba un poquito :-D.

También a todos mis amigos contenidos en los conjuntos *Vagos* y *Hashes*, sin ellos no hubiera sido tan divertido.

Y por último a Rubén, que con su humor negro creo que trató de ayudarme, eso espero o será golpeado.

Índice general

Capítulo 1

Introducción

1.1 Descripción

El trabajo descrito en este documento fue elaborado mientras trabajé en la empresa de desarrollo de sistemas *Root Technologies S.A. de C.V.* En febrero de 2007 los servicios de ésta fueron requeridos por otra, a la que llamaremos en lo sucesivo *el cliente*, que necesitaba un portal de servicios donde los usuarios pudieran acceder a la información de la empresa. Se requería además que existieran diferentes tipos de usuarios con distintos privilegios de acceso, de tal forma que no todos los usuarios pudieran acceder a toda la información.

Se necesitaba un portal que fuera capaz de administrar su contenido entre diferentes usuarios, ubicados en una jerarquía que les provee de diferentes funciones (roles) y por tanto de diferentes privilegios de acceso. Por supuesto el sitio debía contar con la seguridad necesaria para garantizar que sólo los usuarios autorizados, en función de su rol, pudieran acceder a la información en él contenida. Además el sitio debía ofrecer características que permitieran a ciertos usuarios autorizar o vetar la publicación de cierta información.

El cliente en cuestión es una empresa dedicada a la comercialización en gran escala de productos farmacéuticos. El sistema final debía contener información particular acerca de cada uno de los productos ofrecidos por la empresa, similar a la contenida en un Vademecum farmacéutico, así como ligas a la legislación que, posiblemente, rija la comercialización del producto. La información inicial contenida en el sitio debía ser cargada de forma automática a partir de archivos de texto, documentos de procesador de palabras y archivos PDF. Posteriormente esta información debía poder

actualizarse a través de las facilidades que, para ello, provee Plone. La información debía, además, ser organizada de acuerdo a diferentes tipos de contenido diseñados para tal propósito. El sitio no sería público y su diseño visual sería adecuado a las necesidades del cliente.

1.2 Requisitos

Describiendo con mayor detalle, los requisitos del sistema fueron los siguientes:

Requisitos funcionales:

- Crear o utilizar un Sistema de Manejo de Contenido (SMC).
- Crear un tema (presentación(G. 9)) para el portal.
  Esto incluye:
  - Logotipo.
  - Imágenes.
  - Botones.
  - Iconos.
  - Hojas de estilo.
  - Tipos de letra.
  - Estructura del contenido.
- Página principal.
  Se requiere que la página principal del portal incluya un diseño gráfico atractivo, sin afectar la funcionalidad del mismo. El diseño debe incluir un menú intuitivo, que contenga por completo el mapa de navegación establecido.
- Funcionamiento sobre Internet Explorer 6© (desde ahora IE 6©).
- Jerarquía de usuarios.
  Se tendrán definidos los siguientes tipos de usuarios:
  - Superusuario: Rol que controla la totalidad del portal.
  - Administrador: Rol con los permisos correspondientes para generar usuarios con rol de editores y visitantes, además de contar con los permisos de edición y consulta.

  - Editor: Rol que permite la edición de contenidos y autorización de la publicación de los mismos.
  - Visitante: Rol que permite exclusivamente la consulta de contenido.

- Servicio de autenticación mediante contraseña (*password*).

- Autorización para las publicaciones.
  Se requiere que el sistema permita al usuario administrador o editor revisar los contenidos antes de ser publicados y autorizar o negar su publicación en el portal.

- Composiciones o *layouts*(G. 6)[1] para cada plantilla o *template*(G. 11).
  Se requiere el análisis de las *fichas*(G. 3), ya que éstas contienen los campos que serán utilizados para la definición de composiciones, que permitirán la carga automática del contenido.

- Motor de búsqueda.
  Es necesario tener una búsqueda donde el usuario pueda encontrar todos los contenidos del sistema.

- Descarga de contenido.
  Se requiere que el portal permita la descarga de archivos en diversos formatos.

- Fácil acceso a *fichas*.

- Contador estático.
  Se requiere incluir un contador estático que muestre en pantalla la cantidad de días restantes para la entrada en vigor de un registro. El contador estático únicamente aplicará en plantillas donde exista una prórroga de registro.

- Buzón de sugerencias.
  Se requiere que el sistema cuente con un buzón de sugerencias que permita al usuario del portal enviar sus observaciones a los administradores.

- Reportes.
  Se requiere que el sistema cuente con una sección de reportes, en la cual se encuentre un listado de reportes que ofrece al cliente, con su

[1] Ver glosario.

respectiva descripción. La sección de reportes debe permitir al usuario del portal enviar una solicitud de compra vía correo electrónico a la persona destinada para la venta de reportes, la gestión de venta y distribución de reportes le corresponde al cliente.

- Carga automática de contenidos.
  Se requiere realizar la carga de contenido de forma automática por medio de los siguientes pasos:

  1. Captura de contenido que se publicará en el portal, apegándose a las composiciones definidas.
  2. Envío de composiciones, con la información capturada, al operador del portal a través de un medio seguro.
  3. Carga automática de las composiciones asistida por el operador.

  Lo anterior implicará la creación de un módulo que será el encargado de interpretar las composiciones para realizar la carga de los contenidos.

- Edición de la información publicada.
  Se requiere que al usuario administrador o editor se le permita modificar el contenido del portal.

Requerimientos no funcionales:

- Respaldos.
  Se realizarán respaldos semanales alojados en servidores de *Root Technologies* con historial de hasta 2 meses.

- Soporte.
  El soporte que *Root Technologies* proporcionará incluye:

  – Servidor administrado por *Root Technologies*, optimizado para prestar servicios al portal con actualizaciones periódicas.
  – Mantenimiento básico.
  – Actualizaciones semanales.
  – Mesa de ayuda en días hábiles con un horario de 9 a 19 horas.
  – Administrador del sitio cubriendo un tiempo base de 16 horas acumulables semanales.

- Confidencialidad.
  Se requiere que la información proporcionada por el cliente se mantenga en confidencialidad y se utilice exclusivamente para el desarrollo del sistema.

1.3 Trabajos relacionados

Dentro de la UNAM(Universidad Nacional Autónoma de México) *Plone*© ha tomado gran auge, por lo que se han desarrollado diversos sistemas, por ejemplo *Sistema sobre Plone para la captura y recolección de información curricular del Instituto de Matemáticas*[21], *Implementación de un sistema de votación electrónica como un producto sobre la plataforma Plone*[22], *Desarrollo de un Sistema de Administración de Procesos en Plone*[23] y *Migración y nuevas características del sistema de votación electrónica del Instituto de Matemáticas de la UNAM*[24]. Con esto se demuestra la importancia que ha tenido *Plone*© en la conservación de información dentro de la UNAM.

Capítulo 2

Solución

2.1 Propuesta

Una vez que se analizaron los requisitos se decidió usar un SMC (Sistema de Manejo de Contenidos) de código abierto llamado *Plone*© versión 3.1, ya que cumple con varios de los requisitos:

- Funcionamiento sobre $IE6$©.
- Jerarquía de usuarios.
- Servicio de autenticación mediante contraseña.
- Autorización para las publicaciones.
- Motor de búsqueda.
- Descarga de contenido.
- Fácil acceso a fichas.
- Buzón de sugerencias.
- Edición de la información publicada.

En las siguientes secciones se tratará la solución a cada uno de los requisitos faltantes.

2.2 Tema

Para generar el tema de un portal en *Plone*© existe una herramienta llamada *paster* que sirve para crear el esqueleto o estructura (figura 2.1) que debe tener el nuevo tema, una vez que se tiene, es necesario conocer la estructura de las páginas y de esta forma saber cómo agregar nuevos elementos o manipular los existentes.

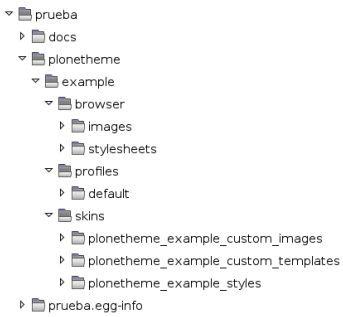

Figura 2.1: Estructura creada por el *paster*.

2.2.1 Estructura

Ésta es la vista de una página de *Plone*© con la configuración por omisión:

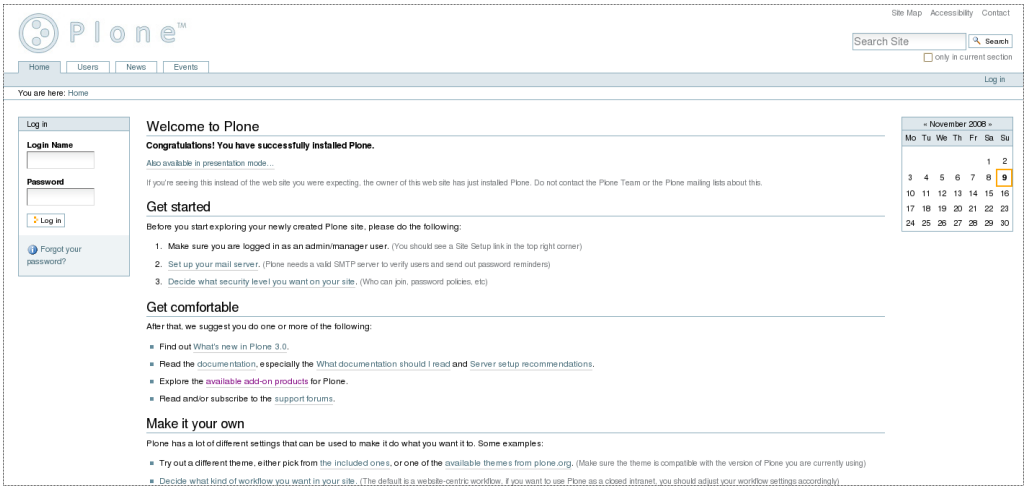

Figura 2.2: Página de *Plone*©.

Las secciones en las que se divide son:

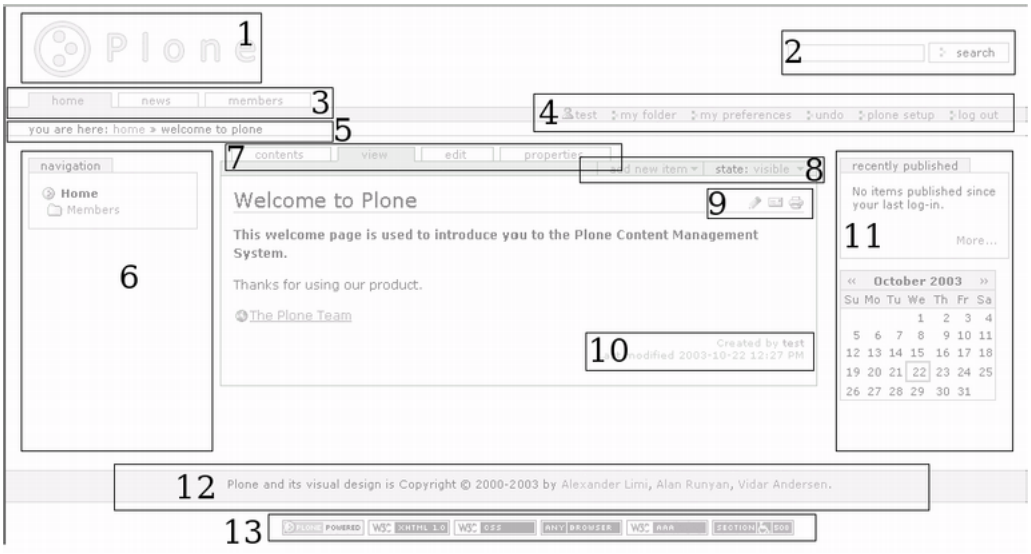

Figura 2.3: Estructura de una página de *Plone*©.

1. El logotipo del portal:
   Se encarga de mostrar el logotipo, si es que existe, en cada página.

2. Caja de búsqueda:
   Se encarga de hacer una búsqueda sobre todo el portal (la *live search*, figura 2.4) muestra los resultados de la búsqueda sin tener que cambiar de página; para ello se debe tener permitido el uso de *JavaScript* en el navegador que se esté usando.

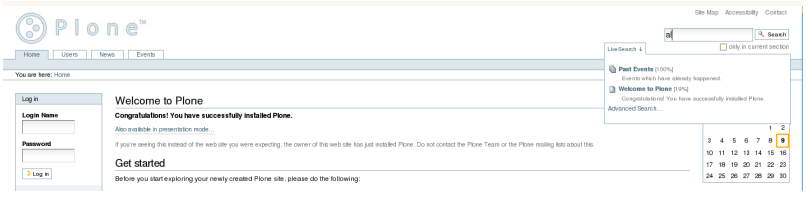

Figura 2.4: Búsqueda dinámica (*Live Search.*)

3. Las pestañas del portal:
   Muestra las secciones del portal.

4. La barra personal:
   Muestra los enlaces a la cuenta del usuario, deshacer y *logout*.

5. Ruta actual:
   Muestra la ruta actual dentro del portal donde se encuentra la página.

6. Espacio de la izquierda:
   Muestra los *portlets*(G. 7) que tenga asignados.

7. Pestañas del contenido:
   Muestra las acciones que el usuario puede tener sobre esta página; las acciones dependen del rol que tenga el usuario; y pueden ser: ver, editar, propiedades y compartir.

8. Menú desplegable:
   Muestra los tipos de documentos que puedes agregar y el estado del documento (publicado, borrador, etc.).

9. Opciones del documento:
   Muestra las acciones permitidas para el documento; éstas pueden ser enviar la página por correo, imprimir, etc.

10. Línea de créditos (*By-line*):
    Muestra el o los autores, contribuyentes y la fecha de la última actualización del contenido.

11. Espacio de la derecha:
    Muestra los *portlets* que tenga asignados.

12. Pie de página:
    Muestra información al final de cada página.

13. Colofón:
    Muestra más información después del pie de página.

Cada una de las secciones está conformada por una o varias estructuras llamadas *viewlets*[7, 8, 10], que son una parte de cada página que puede ser o no mostrada dependiendo de la configuración del portal y/o de los permisos que tenga el rol del usuario. Para poder configurar el estado de un *viewlet* (visible o invisible) es suficiente con entrar a

http://localhost:8080/Plone/@@manage-viewlets (figura 2.5), donde se podrán observar los *viewlets* que contiene el portal y las opciones de ocultar, mostrar, mover arriba y mover abajo.

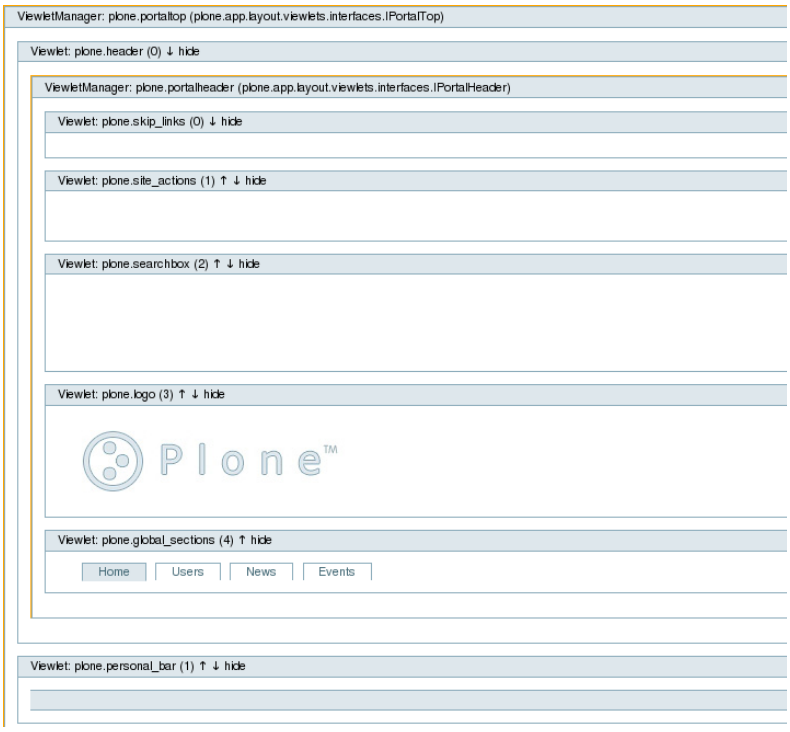

Figura 2.5: *Viewlets.*

Gracias a los *viewlets* se pueden agregar nuevas secciones a las páginas, como menús o una sección de enlaces a sitios de interés.

2.2.2 Desarrollo

Con la información anterior y con ayuda de un complemento de *Firefox*© llamado *firebug*© se pueden modificar algunas reglas para las hojas de estilo y agregar otras para los nuevos elementos, así como cambiar el logotipo de *Plone*© por el del cliente. Fue necesario crear un *viewlet* para el encabezado de las páginas que incluyera ligas para la página de contacto, el buzón de sugerencias y la de *logout*, y otro *viewlet* para incluir nuestro pie de página.

Para la creación de estos *viewlets* tuve que complementar la estructura del *paster*, creando nuevas carpetas para integrar los *viewlets* y cambiando los archivos de configuración para insertarlos en el lugar apropiado, ocultando los *viewlets* que no necesitaba.

Para lograr lo anterior fue necesario aprender:

- Cómo crear y modificar la estructura para un nuevo tema de *Plone*© (*paster*)[3, 11, 12, 13, 15].
- Hojas de estilo en cascada (CSS por las siglas en inglés *Cascading Style Sheets*)[1] con compatibilidad con $IE6$©[1, 2].
- La configuración para ocultar y ordenar los *viewlets*[15].

2.3 Tipos

2.3.1 Estructura

Plone© nos provee de una interfaz gráfica muy poderosa para el manejo de objetos (que dentro del contexto de *Plone*© son llamados tipos de contenido). Por omisión tiene los siguientes tipos: carpeta, imagen, noticia, archivo, liga, evento y página, que en la mayoría de los casos es suficiente; para el nuestro lamentablemente no lo fue, por lo que tuve que recurrir a la creación de nuevos tipos de contenido.

[1] http://www.w3schools.com/css/

Para la creación de nuevos tipos también se puede utilizar el *paster*. En mi caso era necesario crear varios tipos de contenido, incluyendo la validación de campos obligatorios y validación del contenido de campos[2]; en cuanto a la vista, implementar el estilo elegido para cada tema, *scripts* para algunos cálculos y el uso de *widgets*(G. 12) para la edición y/o captura de cada campo.

La estructura obtenida por el *paster* para lograr lo anterior es:

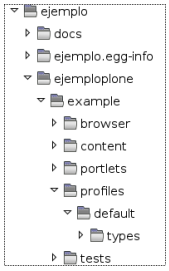

Figura 2.6: Estructura creada por el *paster*.

2.3.2 Creación

Para empezar a crear los tipos de contenido es muy útil usar el patrón de interfaces, ya que de esta forma podemos hacer el código extensible y claro. El lenguaje de programación a utilizar es *Python*(G. 8). Lo primero que se tiene que hacer es definir los tipos y crear sus respectivas interfaces, que son creadas en un mismo archivo llamado *interfaces.py*.

Una vez hecho lo anterior es necesario crear las clases que implementarán las interfaces y por lo tanto se encargarán de representar a cada uno de los nuevos tipos de contenido; estas clases deben estar contenidas en el sub-paquete *content*. La parte más importante de cada clase es la definición de un esquema (*schema*) acorde al constructor *atapi.Schema()* que recibe una secuencia de campos o *fields* como argumentos. Es importante usar el

[2]Por ejemplo: el correo que se pida debe tener un formato correcto, como login@sitio.ext

esquema base y pegarle el nuestro, ya que de esta forma conservamos las características importantes como el *id* o identificador del tipo de contenido. Existen varios campos, entre los más comunes se encuentran los siguientes:

Campo	Descripción
StringField	Una línea simple de texto
TextField	Un campo de varias líneas de texto
LinesField	Una lista de cadenas
IntegerField	Un entero
FixedPointField	Un número decimal
BooleanField	Un check box
FileField	Una caja para subir un archivo
ImageField	Una caja para subir una imagen
DateTimeField	Un calendario

Cada uno de los campos acepta propiedades y algunas configuraciones. Las propiedades que más se usan son las siguientes:

Propiedad	Descripción
required	El valor puede ser *true* o *false* para hacer que un campo sea requerido
searchable	Al asignarle el valor de *true*, el campo es indexado en el catálogo para ser usado en las búsquedas
default	Asigna un valor por omisión al campo
validators	Una lista de validadores para el campo
widget	El *widget*(hablaré de ellos más tarde) que será utilizado para mostrar el campo

Los *widgets* son mecanismos para capturar los valores de los campos; los más importantes se encuentran definidos en *Products.Archetypes.Widget*. Como los campos, los *widgets* también tienen propiedades como:

Propiedad	Descripción
label	Una cadena que es usada como etiqueta del *widget*
description	Una cadena que es usada como el texto de ayuda del *widget*
size	Usado para definir la longitud de las cajas de texto
rows	Usado para definir la altura de las cajas de texto

Al tener lo anterior ya se pueden empezar a formar los *schemas* que contendrán los campos de cada tipo de contenido; por ejemplo, si tenemos en el esquema de uno de los tipos de contenido algo como en la figura 2.7, nos dará como resultado la pantalla de captura 2.8

```
miSchema = folder.ATFolderSchema.copy() + atapi.Schema((
    atapi.TextField(
        required=True,
        searchable=True,
        rows = 10,
        widget=atapi.RichWidget(label="Contenido")
        ),
))
```

Figura 2.7: *Schema.*

Aquí podemos observar que es un campo requerido por el punto rojo a un lado de la etiqueta *Contenido*, y el uso del *widget RichWidget* que nos proporciona herramientas para la edición de texto.

El cliente nos porporcionó algunos de los contenidos que se iban a cargar, basándome en ellos y viendo las opciones de campos que *Plone*© proporciona, diseñé los tipos de contenido, cada uno de ellos contienen diversos tipos de campos, cada uno de acuerdo a las secciones o propiedades de los contenidos que el cliente nos dió. A continuación un relación de los campos que cada tipo de contenido diseñado compartía, los demás debido a que son propiedad intelectual del cliente no pueden ser mostrados.

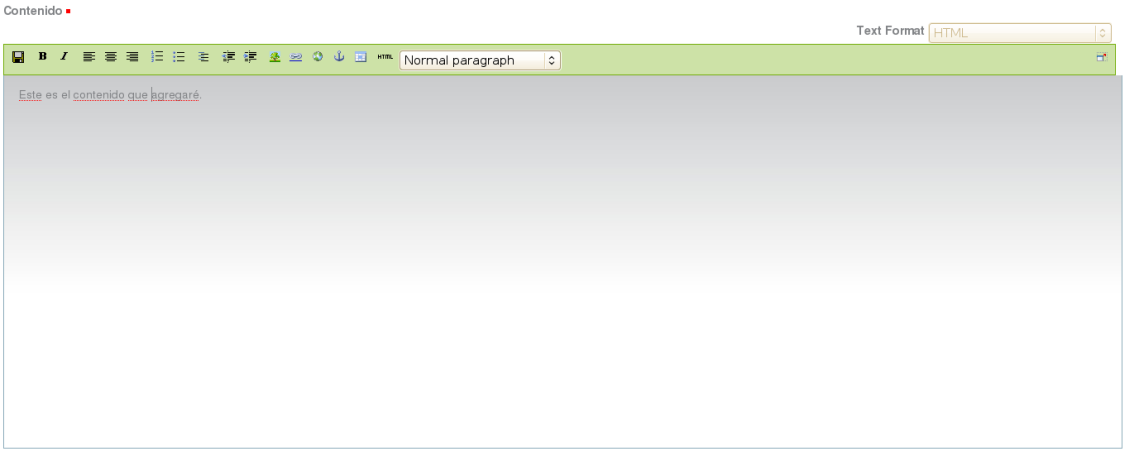

Figura 2.8: Pantalla de captura.

Propiedad	Tipo	Descripción
Tema relacionado	*TextField*	El tema relacionado con este producto
Estadísticas	*FileField*	Un archivo con estadísticas calculadas por el cliente.
Fecha Inicio	*DateTimeField*	Fecha de inicio para su validez
Fecha Fin	*DateTimeField*	Fecha caducidad de la validez
Impacto	*TextField*	El impacto que tiene sobre el paciente
Estado	*TextField*	El estado en que se encuentra la patente del medicamento

2.3.3 DTML

Una vez que fueron creados los campos para cada tipo de contenido es necesario crear la vista de cada uno, para ello se usan archivos con extensión *.pt* (*page template*). En ellos se define la forma y los campos que se mostrarán en la vista de cada tipo de contenido, para lo que se usa un lenguaje de marcado llamado DTML (*Document Template Markup Language*) que usa *Zope*©(G. 16) para mostrar las páginas de forma dinámica.

Para cada tipo de contenido que creé también hice sus respectivos archivos *.pt* donde definí el orden y el lugar donde se mostrarían los campos, así como cuáles sólo se mostrarían en la página de captura; en algunos casos hice llamadas a *scripts* que se encargarían de hacer algunos cálculos antes de mostrar la información de la página.

Algunas de las vistas tenían que mostrar información de acuerdo a los datos contenidos en los tipos (de contenido). Esto se pudo hacer gracias a que también se puede utilizar el TAL (*Template Attribute Language*) que agrega nuevas etiquetas que dan más opciones de manipulación. Con estas herramientas se pueden usar condicionales, algunas líneas de código en *python* y tener acceso directo a las propiedades de los tipos de contenido que están representando.

Por ejemplo, dentro de las propiedades mencionadas se encuentra *Fecha Fin*, ésta fue utilizada para calcular los días restantes hasta ella, para esto utilicé el siguiente segmento de código:

```
<td id="borde" tal:define="resta here/calculaDias">
       <div class="timer" tal:condition="resta">
          Quedan <span tal:replace="resta"></span>
       </div>
</td>
```

Donde *calculaDias* es un script de *Python* que calcula la fecha usando lo siguiente:

```
hoy = DateTime()
ultima = DateTime(str(context.fecha\_fin))
diferencia = ultima - hoy
```

Con lo anterior si *resta* tiene un valor (*tal:condition*) entonces el *span* es sustituido (*tal:replace*) por él, de esta forma se muestran los días que faltan.

Para lograr lo anterior fue necesario aprender:

- A modificar y configurar la estructura de los tipos[6, 9, 15].
- El lenguaje *python*[16, 17].
- DTML(G. 2) y TAL[15](G. 10).
- Estructura del *Schema* que usa *Plone*©[15].

2.4 Carga automática

Otra parte muy importante del desarrollo fue la carga automática de contenidos iniciales; esto surgió debido a que el cliente ya tenía gran parte de la información de los contenidos, por lo que era necesario obtenerla de achivos de texto y de procesador de palabras y crear los nuevos tipos de contenido de forma automática.

Para lograr esta parte se tuvieron que crear reglas para que el cliente nos entregara la información en un formato específico (de ahora en adelante llamado composición), que tomaríamos como entrada para un analizador gramatical comúnmente llamado *parser*.

Para cada tipo de contenido se creó una composición constituida por una tabla en *HTML (HyperText Markup Language)*(G. 4), en la que cada columna era una propiedad y cada renglón un contenido.

2.4.1 Analizador gramatical (parser)

El analizador fue creado en *Python* utilizando paquetes[17] para leer *HTML* que facilitaban la obtención de cada renglón de la tabla; con ello sólo se tenía que dividir cada contenido y agregarlo a un diccionario[3] que es análogo al *map* de JAVA; así, por cada contenido teníamos un diccionario y, al final, por cada tipo, un diccionario de diccionarios, del cual podíamos obtener cada uno de los contenidos para ser guardado.

2.4.2 Comunicación

Para poder guardar los contenidos teníamos que comunicarnos con *Zope*© que es el encargado de guardarlos en la base de datos. Para lograr esta comunicación tuvimos que crear un *método externo*[19, 20] que en *Zope*© es la forma de tener módulos de *python*.

Los métodos externos deben ser módulos seguros, por lo que sólo pueden ser creados si se tiene acceso al sistema de archivos del servidor, ya que es necesario agregar el analizador directamente en la carpeta *Plone/zinstance/parts/zope2/Extensions/*;

[3] Consiste en un conjunto de pares clave-valor, donde las claves son inmutables, mientras que los valores pueden ser de cualquier tipo

además de esto también hay que crear la carpeta *Plone-3.1/zinstance/parts/zope2/lib/python/Products/GlobalModules* y en ella agregar un nuevo *script* con el nombre *__init__.py*, en el cual tenemos que agregar los módulos que serán permitidos para el analizador; de otra forma no podríamos usarlo.

Una vez que fue creado el método externo, se le puede pasar un parámetro a partir del enlace que se usa para tener acceso a él, por ejemplo, *http://localhost:8080/Plone/metodoexterno?parametro=esteeselparametro*, donde *metodoexterno* es el nombre de nuestro método externo, *parametro* es el nombre del parámetro y *esteeselparametro* es el valor del parámetro, que en mi caso es la ruta de la composición.

Una vez que se ejecuta el método externo, el analizador obtiene el diccionario de diccionarios y uno a uno se encarga de guardar los contenidos llamando a la fábrica de *Zope*©. Dado que la composición incluía la ruta de cada contenido, el analizador es capaz de saber dónde insertar cada uno de ellos.

Para lograr lo anterior fue necesario aprender:

- A usar paquetes de *Python*[17].
- Configuración de seguridad para módulos de *python* de *Zope*©.
- Comunicación con la fabrica de *Zope*©[18].

Capítulo 3

Flujo de trabajo (*Workflow*)

Un flujo de trabajo (*workflow*)[4, 5] es un conjunto de estados por los que se tiene que pasar para poder completar una actividad. Para el tipo de portal que desarrollé utilizamos el flujo de trabajo de *intranet*(G. 5), que tiene predefinidos los estados que se deben cumplir para publicar contenido. A continuación mostraré un ejemplo de publicación utilizando este flujo de trabajo.

Para lograr la publicación de un contenido es necesario tener tres tipos de usuarios:

1. Un usuario con el rol de editor, quien será el que crea y edita los contenidos.

2. Un usuario con el rol de revisor, será el encargado de validar que los contenidos creados por el editor sean correctos; si lo son, serán publicados internamente; si no, serán rechazados y enviados de regreso para que el editor haga los cambios pertinentes.

3. Un usuario con el rol de lector, solo podrá leer el contenido publicado.

Lo primero es crear una nueva página; para ello existe el menú *Agregar elemento* (figura 3.1), lo que sólo puede hacer el usuario con el rol de editor.

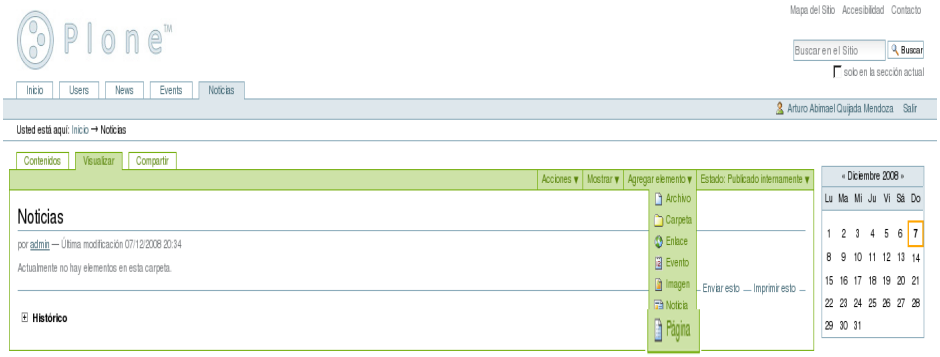

Figura 3.1: *Creación de una página.*

Ahora se muestra una página (figura 3.2) donde podemos agregar nuestro contenido; lo primero es poner el título que será utilizado para generar el *id* (con el que será identificada la página) y el enlace con el que se podrá tener acceso a este contenido; aquí se puede resaltar el editor de texto que soporta edición en *HTML*, formateo wiki(G. 13) o *wikiformatting*(G. 14).

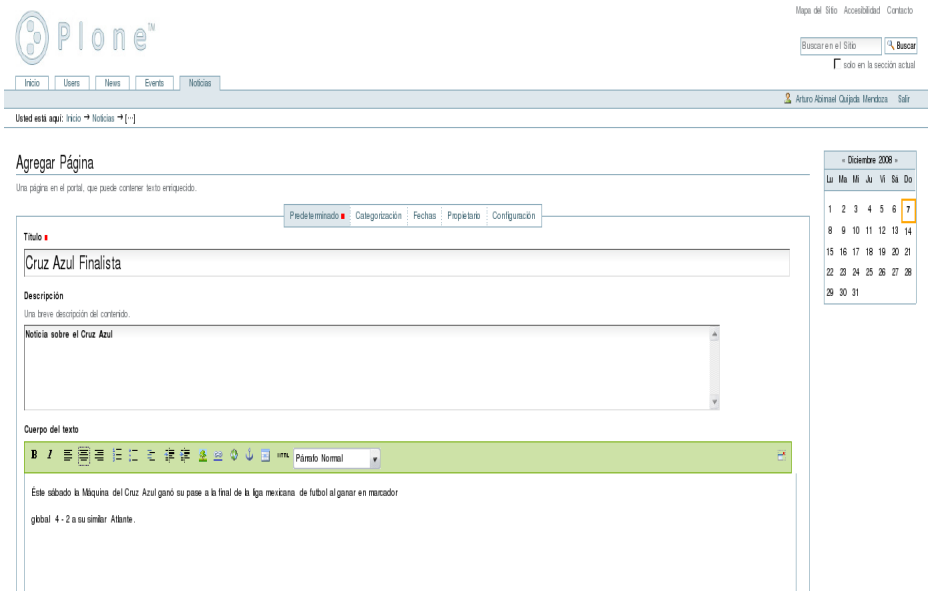

Figura 3.2: *Edición de una página.*

Una vez que hemos guardado la página con nuestro contenido, ésta adquiere el estado de borrador interno; ahora tenemos que cambiarle el estado a *Enviar para publicación* (figura 3.3).

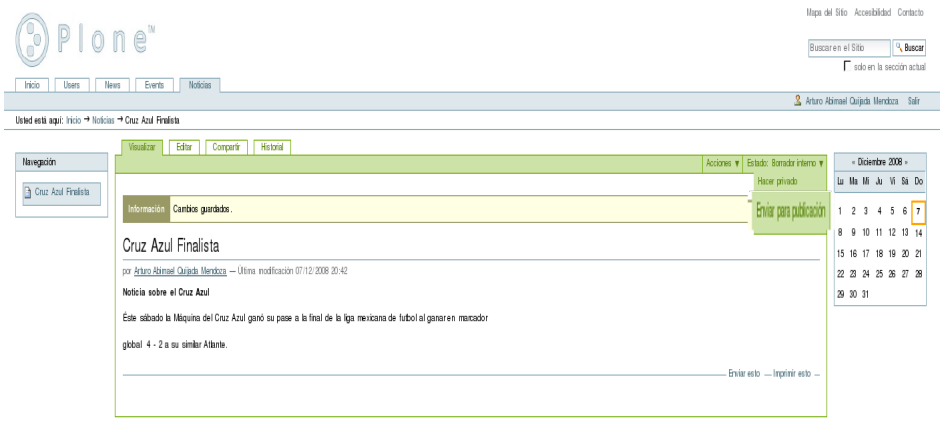

Figura 3.3: *Enviar a publicación una página.*

Con esto el nuevo estado de la página es *Revisión pendiente* (figura 3.4); hasta este punto el trabajo del editor termina.

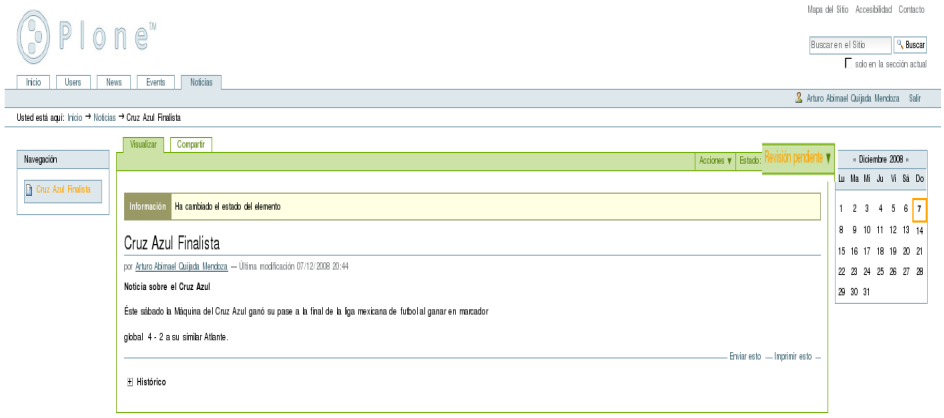

Figura 3.4: *Revisión pendiente página.*

Ahora es momento de que el revisor entre en acción. Él ve el estado de nuestra página y revisa el contenido; si es correcto cambia el estado a *Publicar internamente* (figura 3.5); si no, lo rechaza y el editor tendrá que arreglarlo para intentar publicarlo nuevamente. Una vez que ha sido publicado, los usuarios con rol de lectores podrán leer el nuevo contenido.

Figura 3.5: *Publicación de una página.*

A continuación muestro el diagrama de estados del flujo de trabajo que usa Plone por omisión.

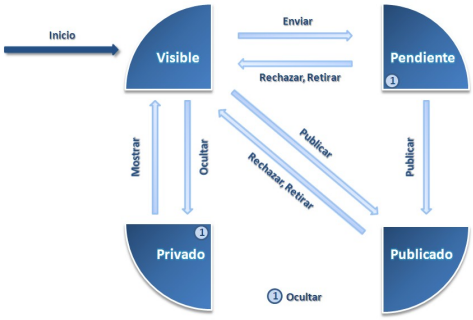

Figura 3.6: *Flujo de trabajo (workflow) para publicación.*

Capítulo 4

Conclusiones

El trabajo realizado sorprendió al cliente debido a que fue más de lo que esperaba y le ayudó a cambiar los procesos que utilizaba para ofrecer información a sus clientes. Con el sistema para él ahora es más sencillo hacer que sus clientes tengan acceso a la información que ofrece, debido a que el sistema les permite:

- Publicar información.
- Editarla en línea.
- Publicar archivos pdf.
- Incluir todos los contenidos dentro de una búsqueda.
- Carga masiva de los contenidos que ya tienen.

Con todo lo anterior el cliente quedó satisfecho y hemos tenido nuevos proyectos con él. Todo lo que hice pude lograrlo gracias a las bases sólidas que la carrera me proporcionó, en particular:

- Diseño.
- Programación.
- Solución de problemas.

Lo anterior me ayudó a diseñar la solución instrumentada, lo que requirió además aprender un nuevo lenguaje de programación en un corto tiempo. Esto fue posible gracias a la formación adquirida durante mis estudios de

licenciatura, en los que aprendí los fundamentos de diseño de lenguajes de programación y, en general, lo necesario para enfrentar nuevas tecnologías.

Esta experiencia resultó además enriquecedora desde el punto de vista profesional, dado que fue posible entrar en contacto con las tecnologías de vanguardia en la elaboración de sistemas manejadores de contenido. Me brindó a mí y al equipo de trabajo con el que estuve involucrado, la posibilidad de evaluar y analizar las cualidades y características de diversos sistemas manejadores de contenido, lo que nos permitió además abstraer las características relevantes de éstos y decidir cuáles eran las necesarias para realizar el portal que debíamos desarrollar.

Desde el punto de vista técnico se puede concluir que el desarrollo de un portal como el que se debía hacer, hubiera necesitado de muchos meses de trabajo, seguramente cuatro veces más de los que se utilizaron, si el desarrollo hubiera estado basado en las tecnologías de la generación anterior (como las basadas en *servlets* y sustentadas por entornos de trabajo como *tapestry*). Si bien en este tipo de desarrollos se tiene mayor control y es posible hacer un uso más eficiente de los recursos teniendo sólo lo indispensable, es cierto también que se requiere desarrollar mucha más infraestructura, que ya está integrada en los actuales SMC. En síntesis los SMC son una buena alternativa si no se requiere tener características muy peculiares y la velocidad de desarrollo y robustez del sistema tienen prioridad sobre la eficiencia del mismo.

Glosario

1. CMS (*Content Management System*) - Un Sistema de Manejo de Contenidos, es un programa que permite crear una estructura de soporte (*framework*), para la creación y administración de contenidos por parte de los creadores y administradores de contenido en sitios Web.

   Un ejemplo clásico es el de editores que cargan el contenido al sistema y otro de nivel superior que permite que estos contenidos sean visibles a todo el público.[25]

2. DTML(*Document Template Markup Language*) - Es una herramienta para generar plantillas que permitan el uso de *HTML* dinámico.

3. Ficha - Es una página de contenido dentro de *Plone*©.

4. HTML - siglas de *HyperText Markup Language* (Lenguaje de Marcas de Hipertexto), es el lenguaje de marcado predominante para la construcción de páginas Web. Es usado para describir la estructura y el contenido en forma de texto, así como para completar el texto con objetos tales como imágenes. HTML se escribe en forma de "etiquetas", rodeadas por corchetes angulares (<,>). HTML también puede describir, hasta cierto punto, la apariencia de un documento, y puede incluir scripts (por ejemplo Javascript), el cual puede afectar el comportamiento de navegadores Web y otros procesadores de HTML.[25]

5. Intranet - Una Intranet es un conjunto de contenidos compartidos por un grupo definido dentro de una organización.

6. Composición - Es una parte del diseño gráfico que permite el arreglo y diseño del estilo en los elementos de una página.

7. *Portlet* - Es una caja que muestra diferentes tipos de contenidos, como pueden ser noticias o eventos.

8. *Python* - Es un lenguaje de programación interpretado, creado por Guido van Rossum en el año 1990.

   Se compara habitualmente con *TCL, Perl, Scheme, Java y Ruby.* En la actualidad *Python* se desarrolla como un proyecto de código abierto, administrado por la *Python Software Foundation.* La última versión estable del lenguaje es la 3.1.1 .

   Python es un lenguaje de programación interpretado, lo que ahorra un tiempo considerable en el desarrollo del programa, pues no es necesario compilar ni enlazar. El intérprete se puede utilizar de modo interactivo, lo que facilita experimentar con características del lenguaje, escribir programas desechables o probar funciones durante el desarrollo del programa.[25]

9. Presentación - Es una serie de elementos gráficos que, al aplicarse sobre un determinado *software*, modifican su apariencia externa.

   Estos elementos son independientes de la propia aplicación, con lo que ésta puede tener entre sus opciones varias de estas presentaciones o ninguna, mostrando una mejor apariencia más o menos vistosa.

   Sin embargo, cada presentación se puede aplicar exclusivamente sobre un software determinado, no pudiendo enviarse a otros programas.[25]

10. TAL (*Template Attribute Language*) - Es un lenguaje utilizado para crear páginas en *HTML* y *XML.* Fue creado por Zope pero es utilizado en otras aplicaciones basadas en *Python.*

11. Template o plantilla - Es una estructura prefabricada para poder hacer más estructuras, usándola como base.

    Una plantilla agiliza el trabajo de reproducción de muchas copias idénticas o casi idénticas (que no tiene que ser tan elaborado, sofisticado o personal). A partir de la plantilla pueden, asimismo, diseñarse y fabricarse nuevas plantillas.

    Las plantillas como norma general pueden ser utilizadas por personas o por sistemas automatizados. Se utilizan plantillas en todos los terrenos de la industria y la tecnología. Una plantilla puede servir como muestra base de una diversidad sobre la que comparten elementos comunes (patrón) y que en sí es lo que constituye la plantilla.[25]

12. *Widget* - Es una pequeña aplicación o programa, usualmente presentado en archivos pequeños que son ejecutados por un motor de *widgets* o *Widget Engine*. Entre sus objetivos están los de dar fácil acceso a funciones frecuentemente usadas y proveer de información visual. Sin embargo los *widgets* pueden hacer todo lo que la imaginación desee e interactuar con servicios e información distribuida en Internet; pueden ser vistosos relojes en pantalla, notas, calculadoras, calendarios, agendas, juegos, ventanas con información del tiempo en su ciudad, etcétera.[25]

13. *Wiki* - Un *wiki*, o una *wiki*, es un sitio Web cuyas páginas pueden ser editadas por múltiples voluntarios a través del navegador Web. Los usuarios pueden crear, modificar o borrar un mismo texto que compartan. Los textos o "páginas wiki" tienen títulos únicos. Si se escribe el título de una "página-wiki" en algún lugar del wiki, esta palabra se convierte en un "enlace web" (o "link") a la página Web.[25]

14. *Wikiformatting* - Es la sintaxis que se usa para darle formato al texto de un *wiki*.

15. *Workflow* - Es una secuencia de acciones necesarias para completar un proceso.

16. *Zope*©.- Es una infraestructura de servicios para desarrollo de aplicaciones, en particular para manejo de contenidos.

Bibliografía

1. CSS Tutorial, http://www.w3schools.com/css/
   Guía básica para aprender CSS.

2. *htmlPlayground*, http://www.htmlplayground.com/
   Contiene una aplicación donde se pueden hacer cambios a código html de forma dinámica.

3. *Create new eggs and packages quickly with paster.*
   http://plone.org/documentation/kb/use-paster
   Documentación para aprender el uso del paster.

4. *Plone Workflows.*
   http://zamasing.com/zope_lounge/articles/plone_workflows__part_1
   Breve explicación sobre los flujos de trabajo(workflows) en *Plone*©.

5. *Collaboration and Workflow.*
   http://plone.org/documentation/manual/plone-3-user-manual/collaboration-and-workflow/referencemanual-all-pages
   Describe el proceso de publicación.

6. *Using ArchGenXML.*
   http://plone.org/documentation/manual/archgenxml2/referencemanual-all-pages
   Instalación y uso de *Archgen* y *ArgoUML* para la creación de tipos de contenido.

7. Plone.
   http://el-directorio.org/Plonehead-37a78e2d91dc8c4ca8765fc1a5b1d79f510bb9d9
   Descripción de algunas secciones y configuraciones de *Plone*©.

8. Anatomía de Plone.
   http://www.slideshare.net/r0ver/anatomia-de-plone
   Descripción de la estructura de *Plone*©.

9. *Content-types.*
http://plone.org/documentation/manual/archetypes-developer-manual
Creación de nuevos tipos de contenido usando *archetypes.*

10. *What controls what you see.*
http://plone.org/documentation/tutorial/where-is-what/tutorial-all-pages
Cambios en las *CSS* para cambiar la imagen de un sitio *Plone*©.

11. *Visual Design.*
http://plone.org/documentation/phc_topic_area?topic=Visual+Design
Cambio en la imagen de un sitio *Plone*© para que se vea bien en IE.

12. *Plone Book for Content Managers.*
http://www.contentmanagementsoftware.info/plone-book
Menciona como modificar el pie de página.

13. *Understanding permissions and security.*
http://plone.org/documentation/kb/understanding-permissions/tutorial-all-pages
Seguridad y permisos con workflows.

14. *Users, Authentication, and Permissions.*
http : //plone.org/documentation/phc_topic_area?topic = Users%2C + Authentication%2C + and + Permissions
Seguridad para usuarios y *LDAP* para permisos de archivos.

15. *Professional Plone Development*, septiembre 2007, Martin Aspeli.

16. *The Python Tutorial.*
http://docs.python.org/tutorial/
Tutorial básico para aprender *Python*

17. *HTML Tidy Library Project.*
http://tidy.sourceforge.net/
Biblioteca para parsear *HTML.*

18. *Adding Folders with Contents with FormGen.*
http://plone.org/products/ploneformgen/documentation/how-to/adding-folders-with-contents-with-formgen
Cómo generar contenido desde un script.

19. *Create and Use an External Method.*
http://plone.org/documentation/kb/create-and-use-an-external-method
Menciona cómo hacer un método externo.

20. *Using External Methods.*
http://www.zope.org/Documentation/How-To/ExternalMethods
Menciona cómo funcionan los métodos externos con un ejemplo.

21. Tesis de Maestría en Ciencias de la Computación de Marco Antonio López Rabadán. Título: "Sistema sobre Plone para la captura y recolección de información curricular del Instituto de Matemáticas," Posgrado Computación, UNAM, Diciembre 4, 2007. Mención Honoríca.

22. Tesis de Maestría en Ciencias de la Computación de Alexander Zapata Lenis. Título: Implementación de un sistema de votación electrónica como un producto sobre la plataforma Plone" Posgrado Computación, UNAM, Marzo 4, 2008. Mención Honoríca.

23. Tesis de Maestría en Ciencias de la Computación de Eduardo Espinosa Ávila. Título: "Desarrollo de un Sistema de Administración de Procesos en Plone" Posgrado Computación, UNAM, Febrero 27, 2009.

24. Tesis de Maestría en Ciencias de la Computación de Iván Christian Cervantes Coronado. Título: "Migración y nuevas características del sistema de votación electrónica del Instituto de Matemáticas de la UNAM," Posgrado Computación, UNAM, Octubre 1, 2009.

25. *Wikipedia.*
http://es.wikipedia.org
La enciclopedia libre.