# Zope: Plataforma Libre para la Gestión de Contenidos

Fernando Quintana Hernández
nando@trapagaran.org

28 y 29 de Julio de 2005
www.e-ghost.net

# Índice de Contenidos (I)



ZOPE Community

# Índice de Contenidos (II)

- La adquisición
- Creación de plantillas
- Zope Page Templates
- DTML
- Python Scripting en Zope
- Web Scripting en Zope
- Búsquedas en catálogos
- Aplicaciones Zope en Python

ZOPE Community

# Content Management System (I)

- Permite:
  - almacenar grandes cantidades de documentos electrónicos y administrarlos durante todo su ciclo de vida (creación-publicación)
  - trabajar en colaboración, contribuyendo con información mediante un interfaz (web) amigable.
  - crear portales web donde vertebrar la gestión de toda la información de una organización

# Content Management System (II)

- Permite:
  - definir un look and feel corporativo mediante una plantilla y reutilizarlo para cada página del portal
  - que autores y editores sin conocimientos técnicos, publiquen sus contenidos en la web de manera rápida y sencilla.
  - establecer procesos de publicación específicos (workflow) y regular mediante permisos, las acciones permitidas sobre los contenidos publicados
    - Workflow = ciclo de vida de un documento desde su creación hasta su publicación, pasando por los procesos de corrección, traducción, aprobación

# Content Management System (III)

- Ejemplos:
  - Mambo ( http://www.mamboserver.com/ )
  - php-nuke ( http://www.phpnuke.org/ )
  - Etomite ( http://www.etomite.org/ )
  - TikiWiki ( http://www.tikiwiki.org/)

  - **Zope ( http://www.zope.org )**

  - http://opensourcecmf.org

# Introducción a Zope (I)

- ¿Qué es Zope?
  - Servidor de aplicaciones basado en Python
    - http://www.zope.org/
  - Gestor de aplicaciones web
    - Escrito en Python
    - Con licencia libre
  - A Zope se le suele achacar una larga curva de aprendizaje

# Introducción a Zope (II)

- Z Objects Publication Environment =
  - programación orientada a objetos (python) +
  - persistencia de objetos
  - acceso remoto a los objetos (http, xml-rpc, etc.)
- Ofrece:
  - Interfaz de administración web (ZMI)
  - Base de datos OO (soporta otras BBDD relacionales)
  - Servidor web

# Introducción a Zope (III)

- Está diseñado para delegar, de forma segura, el control de un sitio web a diferentes grupos de usuarios
  - Se puede administrar utilizando un navegador web
  - El interfaz de administración se parece mucho a un administrador de ficheros (nautilus, windows explorer, etc.)
  - Permite el acceso a los recursos de forma concurrente
  - Facilita la recuperación frente a errores.

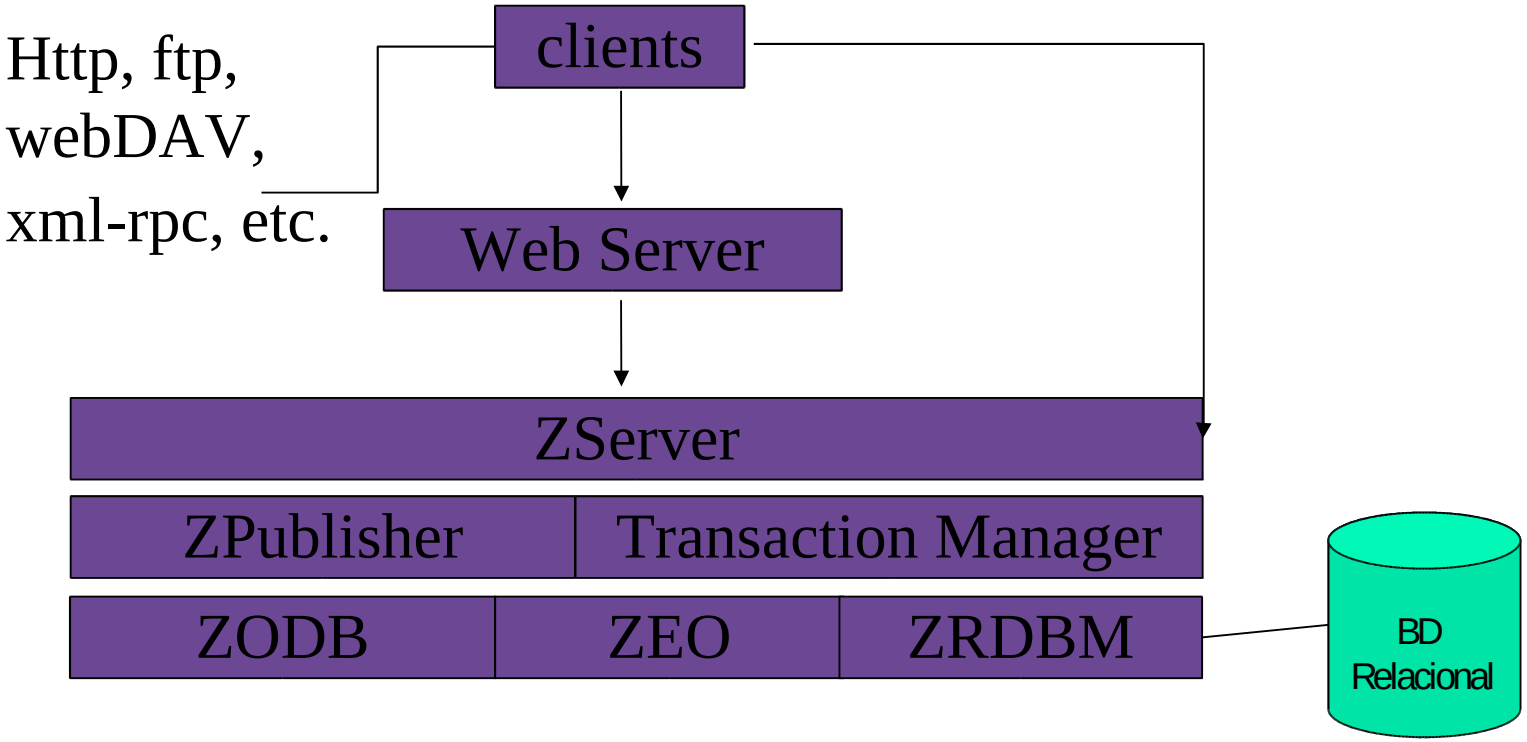
# Introducción a Zope (IV)

- En 1996 Jim Fulton de Digital Creations ideó el núcleo de Zope.
- Comienza con tres componentes:
  - `Bobo`, un ORB ligero para la web
  - `Document Template`, un lenguaje de scripting
  - `BoboPos`, una base de datos orientada a objetos
- Se desarrolla una aplicación comercial llamada `Principia.`
  - En 1998 se libera su código bajo el nombre de `Zope`

# Introducción a Zope (V)

- ¿Dónde encontrar documentación?
  - Zope Book:
    - disponible online o en PDF: http://www.zope.org/Documentation/Books/ZopeBook/2_6Edition/
  - Zope Developers Guide:
    - disponible online o en PDF: http://www.zope.org/Documentation/Books/ZDG/
  - Multitud de sitios en Internet:
    - http://google.com
  - Listas de correo
    - [zope-es], [zope]

# Arquitectura Zope (I)




# Arquitectura Zope (II)

- **ZServer**
  - Actúa de intermediario entre el cliente y el ZPublisher
  - Conoce los protocolos HTTP, FTP, XML-RPC, etc. soportados por Zope.
  - Traduce las peticiones de los clientes en solicitudes de métodos u objetos
  - Traduce las respuestas del ZPublisher para devolverlas a los clientes

# Arquitectura Zope (III)

- **ZPublisher**
  - Es el componente ORB de Zope
  - Recibe peticiones del ZPublisher
  - Se encarga de localizar objetos y métodos e invocarlos sin son invocables (`callable`)
  - Devuelve la respuesta al ZServer

# Arquitectura Zope (IV)

- **Transaction Manager**
  - Se asegura de que todas las operaciones de una transacción hayan tenido éxito
  - O revierte la base de datos al estado anterior al inicio de la transacción.

# Arquitectura Zope (V)

- **ZODB**
  - Se encaga de la persistencia de los objetos Zope
  - Almacena información transaccional (descripción de los objetos + cambios realizados)
  - Permite 'deshacer' acciones.

# Arquitectura Zope (VI)

- **ZEO**
  - Permite compartir una ZODB entre varias instancias de Zope corriendo en varios servidores
  - Ofrece la posibilidad de escalar un servidor en varias máquinas para soportar más tráfico.

# Arquitectura Zope (VII)

- **ZRDBM**
  - Ofrece un interfaz abstracto para operar con bases de datos relacionales de forma transparente.
  - Siempre que estas bases de datos soporten transacciones.

# Instalación de Zope (I)

- Instalación básica:
  - `http://www.zope.org/Products/Zope/2.7.4/Zope-2.7.4-0-win32.exe`
- Se crean los siguientes directorios:
  - `C:\Archivos de Programa\Zope-2.7.4-0\bin\`
  - `C:\Archivos de Programa\Zope-2.7.4-0\doc\`
  - `C:\Archivos de Programa\Zope-2.7.4-0\import\`
  - `C:\Archivos de Programa\Zope-2.7.4-0\lib\`
  - `C:\Archivos de Programa\Zope-2.7.4-0\eskel\`
- Creamos el usuario administrador:
  - Usar como password de ‘admin’ po rejemplo ‘tknika’

# Instalación de Zope (II)

- Una misma instalación de Zope puede ejecutar múltiples instancias
  - Cada instancia puede escuchar en un puerto diferente
    - Útil para obtener diferentes ramas de nuestro producto
    - Ofrecer servicios de hosting

# Instalación de Zope (III)

- Después, creamos una instancia Zope

```
python.exe /bin/mkzopeinstance.py
```

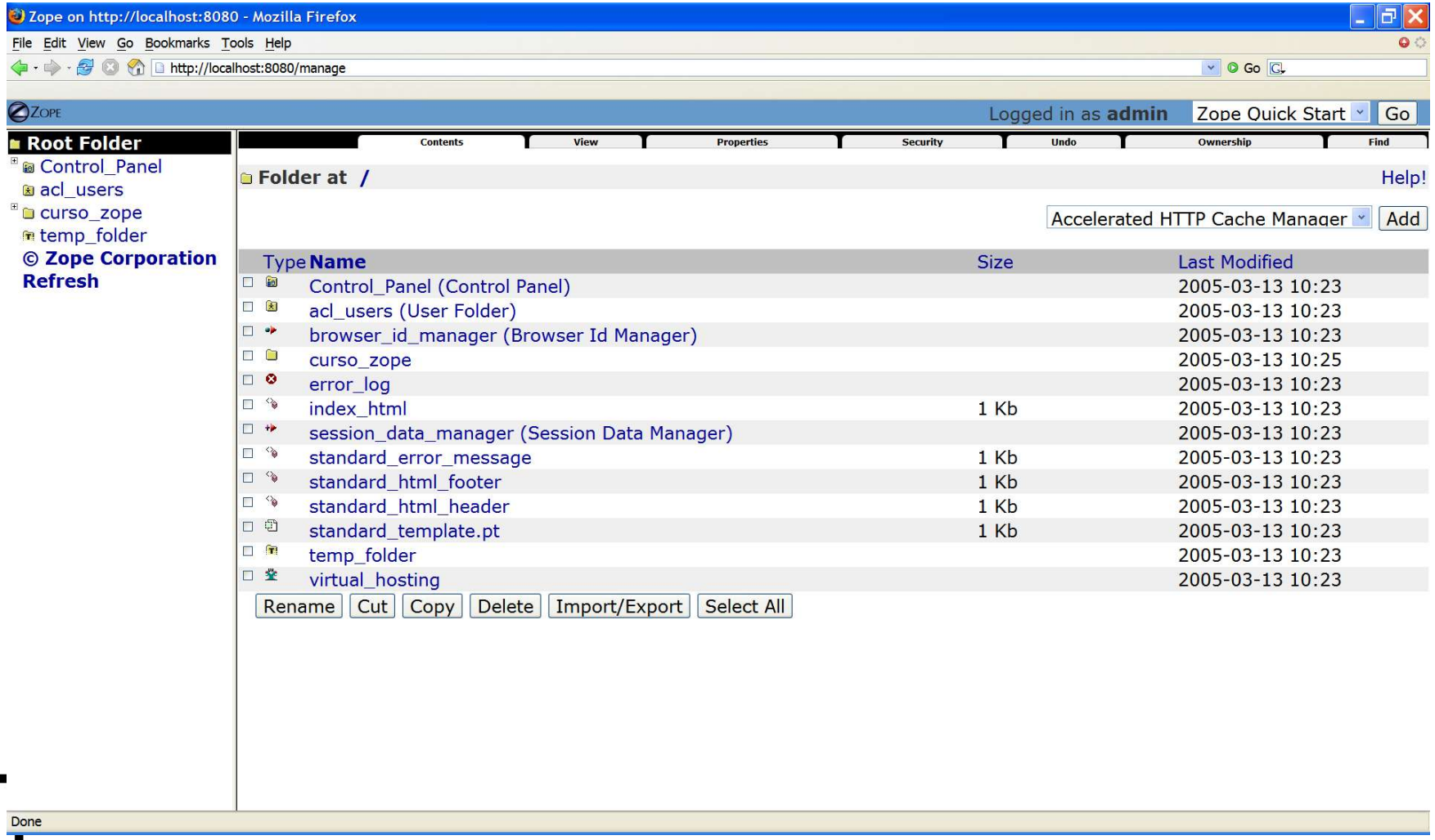
- Se crean los siguientes directorios:

```
C:\Zope\bin\
C:\Zope\etc\
C:\Zope\var\
C:\Zope\log\
C:\Zope\Products\
C:\Zope\Extensions\
C:\Zope\import\
```

# ZMI (I)

- Zope Management Interface
  - http://localhost:8080/manage
  - http://localhost:8080/manage_main
- Interfaz de administración de Zope
  - Similar a un explorador de ficheros

# ZMI (II)


Zope on http://localhost:8080 - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost:8080/manage
Go
ZOPE
Logged in as admin
Zope Quick Start
Go
Root Folder
Control_Panel
acl_users
curso_zope
temp_folder
© Zope Corporation
Refresh
Contents
View
Properties
Security
Undo
Ownership
Find
Folder at /
Help!
Accelerated HTTP Cache Manager
Add
Type Name
Size
Last Modified
Control_Panel (Control Panel) 2005-03-13 10:23
acl_users (User Folder) 2005-03-13 10:23
browser_id_manager (Browser Id Manager) 2005-03-13 10:23
curso_zope 2005-03-13 10:25
error_log 2005-03-13 10:23
index_html 1 Kb 2005-03-13 10:23
session_data_manager (Session Data Manager) 2005-03-13 10:23
standard_error_message 1 Kb 2005-03-13 10:23
standard_html_footer 1 Kb 2005-03-13 10:23
standard_html_header 1 Kb 2005-03-13 10:23
standard_template.pt 1 Kb 2005-03-13 10:23
temp_folder 2005-03-13 10:23
virtual_hosting 2005-03-13 10:23
Rename
Cut
Copy
Delete
Import/Export
Select All
Done

# ZMI (III)

- Listado de clases (crear objetos)
- Listado de objetos (borrar, copiar, cortar y pegar)
- Propiedades de los objetos
- Importar y exportar objetos
- Deshacer acciones

# ZMI (IV)

- Tipos de Objetos Zope:
  - Objetos Contenedores
    - User Folder, Folder, etc.
  - Objetos Plantillas
    - Page Template, DTML Document, etc.
  - Otros objetos
    - File, Image, etc.

# Gestión de Usuarios y Permisos (I)

- Carpetas de usuarios (`acl_users`)
  - Contiene objetos de tipo usuario (`user`)
    - Cada uno de estos usuarios podrá autenticarse al acceder a la carpeta contenedora de la carpeta de usuarios y a todos sus contenidos
    - pero nunca más arriba en la jerarquía de objetos
  - Los usuarios que hayan de compartir los mismos permisos sobre los mismos objetos tendrán un mismo rol.

# Gestión de Usuarios y Permisos (II)

- Los roles que vienen predefinidos en Zope son:
  - Manager, que representa el perfil de un administrador
  - Anonymous, que es el rol que tiene el usuario anónimo
  - Owner, asignado automáticamente a un usuario en el contexto de un objeto creado por él mismo
  - Authenticated, asignado automáticamente a un usuario cuando se autentica con éxito.
- Se pueden definir nuevos roles locales en cada objeto

# Gestión de Usuarios y Permisos (III)

- Todo lo que se puede hacer con un objeto viene declarado en la lista de permisos de cada objeto.
  - Por ejemplo, el permiso "view" de los objetos "Page Template" representa la acción de visualizar una plantilla
- Lo que un usuario puede o no puede hacer con un recurso depende del rol que tenga dicho usuario.
  - En cada objeto se configura qué permisos tiene cada rol
  - Los objetos adquieren estas configuraciones de su contenedor

# Primer Contacto con Zope (I)

- Para acceder al Zope Management Interface vamos a la url:
  - http://localhost:8080/manage
  - Nos autenticaremos con 'admin', 'tknika'
  - ZMI es un centro de control de Zope
- Presenta una serie de objetos de diferentes tipos, cada uno con sus funcionalidades correspondientes

# Primer Contacto con Zope (II)

- Control Panel nos permite:
  - Parar o reiniciar una instancia
    - `Control_Panel/database/main`:
    - Compactar la base de datos de objetos
  - Para ver la lista de productos instalados:
    - `Control_Panel/Products`

# Primer Contacto con Zope (III)

- En zope todo es un objeto
- Cada objeto es accesible desde una URL, así como sus métodos y propiedades
  - Concepto de Publicación
  - Concepto de Adquisición
- Ejemplo:
  - http://localhost:8080/Control_Panel/Database/main/manage
  - Método `manage` de un objeto `main` situado en `Control_Panel` / `Database`

# Primer Contacto con Zope (IV)

- Vayamos a "`Root Folder`", pestaña `Contents`
- Hacer clic en el desplegable que nos permite seleccionar los objetos que podemos crear
  - Depende de los productos instalados
- Seleccionamos `Folder` con `ID 'depart_calidad'` y `Title 'Departamento de Calidad'`
- Marcamos los dos check boxes
  - Contendrá página por defecto y repositorio de usuarios
- Hacemos lo mismo con `'depart_comercial'`
- Ahora podemos acceder a los objetos creados como:
  - http://localhost:8080/depart_calidad/
  - http://localhost:8080/depart_comercial/

# Primer Contacto con Zope (V)

- Restringiendo accesos:
  - Los usuarios de un departamento solamente pueden acceder a su departamento en la intranet
- Accedemos al objeto "`acl_users`" de cada una de estas carpetas y creamos los usuarios: '`usuario_calidad`' / '`usuario_calidad`' y '`usuario_comercial`' / '`usuario_comercial`'
- No vamos a permitir acceso más que de lectura a los usuarios indicados, a los anónimos no les permitimos el acceso
  - Vamos a "`Root Folder`" / "`Security`"
    - Desmarcamos permisos de Anonymous
    - Asignamos '`Access contents information`' y '`View`' al grupo '`Authenticated`'

# Primer Contacto con Zope (VI)

- Accedemos a la url:
  - http://localhost:8080/depart_calidad
- y nos autenticamos con:
  - `'usuario_calidad'/'usuario_calidad'`
- La gestión de usuarios, grupos, permisos y control de acceso ya vienen proporcionados por Zope

# Modelo de Desarrollo (I)

- Dos formas de utilizar estos recursos:
  - Modo restringido
    - Aplicado a la programación realizada a través del ZMI
      - Evitar que nadie pueda dañar el sistema
      - El código tiene restricciones:
        - Menos funcionalidades, limita los módulos de Python que podemos usar
        - Perdida de rendimiento por comprobaciones
  - Modo no restringido
    - No hay ninguna traba podemos usar el poder de Zope y Python, e incluso programas externos
    - Es más peligroso
    - Se utiliza para desarrollar nuevos productos Zope

# Modelo de Desarrollo (II)

- Zope Page Templates (ZPT)
  - Herramienta para la generación de páginas web y contenido XML
  - Diseñado para separar capa de presentación del resto del código y facilitar colaboración entre diseñadores y programadores
  - El código se incluye como atributos del propio HTML

# Modelo de Desarrollo (III)

- Ejemplo: bucle que recorre una carpeta y genera un link a cada uno de los objetos contenidos

```
<tr tal:repeat="objeto here/objectValues">
 <td>
  <a tal:attributes="href objeto/absolute_url"
     tal:content="objeto/title_or_id"/></a>
 </td>
</tr>
```

# Modelo de Desarrollo (IV)

- Document Template Markup Language (DTML)
- Mismo propósito que ZPT, pero con sintaxis más básica
  - Se recomienda el uso de ZPT desde Zope.org

# Modelo de Desarrollo (V)

- Ejemplo: bucle que recorre una carpeta y genera un link a cada uno de los objetos contenidos

```
<dtml-in objectValues>
 <tr>
  <td>
   <a href="<dtml-var absolute_url>">
     <dtml-var title_or_id>
   </a>
  </td>
 </tr>
</dtml-in>
```

# Modelo de Desarrollo (VI)

- Script (Python)
  - Versión reducida y segura de Python
  - Limita la importación de módulos asi como funciones susceptibles de ser peligrosas como eval
  - Contiene la lógica de negocio

# Modelo de Desarrollo (VII)

- Ejemplo de Script Python:

```
objetos = container.objectValues()
para_ordenar=[(objeto.title_or_id(), objeto) for
   objeto in objetos]
para_ordenar.sort()
resultado=[]
for ordenado in para_ordenar:
   resultado.append(ordenado[1])
return resultado
```

ZOPE Community

# Modelo de Desarrollo (VIII)

- Bases de datos relacionales
  - La interacción con bases de datos relacionales se realiza por medio de contenedores o adaptadores de base de datos
  - Actúan de puente entre la base de datos y Zope.
  - Existen conectores para la gran mayoría de bases de datos relacionales

# Modelo de Desarrollo (IX)

- Contenedores o adaptadores de BBDD
  - No se distribuyen conjuntamente con Zope
  - Proporcionan independencia de la base de datos
    - El código se sitúa en objetos Z SQL Method, independientes del conector SQL
  - Para buscar información sobre conectores externos ir a:
    - http://zope.org/Products/external_access

# Modelo de Desarrollo (IX)

- Zope Object Data Base (ZODB)
  - Base de datos en la que se basa Zope
  - Existe cierta controversia si utilizar BBDD relacionales o ZODB
  - El uso de ZODB permite funcionalidades más avanzadas
    - Útil para restringir accesos a contenidos

# Modelo de Desarrollo (IX)

- ZCatalogs
  - Son catálogos de objetos
  - Permiten indexar los objetos por sus propiedades o métodos y realizar búsquedas rápidas sobre ellos

# Creación de Plantillas (I)

- Zope Page Templates
  - Se trata de un XML válido, con elementos y atributos de varios espacios de nombres (XHTML, TAL y METAL)
  - Estos atributos modifican el XML final en el momento de su invocación
- DTML
  - Es un fichero de texto (HTML, CSS, etc.) con sentencias de código intercalado

# Creación de Plantillas (II)

- Zope Page Templates
  - Ejemplo, mostrar las propiedades `title` y `body` del objeto que contiene la plantilla:

```
<html>
 <head>
  <title tal:content="container/title"></title>
 </head>
 <body>
  <div tal:content="container/body"></div>
 </body>
</html>
```

# Creación de Plantillas (III)

- Zope Page Templates
  - El ejemplo anterior podría renderizarse en algo así:

```
<html>
 <head>
  <title>Sección uno</title>
 </head>
 <body>
  <div>En la sección uno les mostramos...</div>
 </body>
</html>
```

# Creación de Plantillas (IV)

- DTML Document
  - Ejemplo, rellenar una css con propiedades de una carpeta:

```
#carpeta
{
 text-color: <dtml-var colorDelTexto>;
 border-color: <dtml-var colorDelBorde>
}
```

# Creación de Plantillas (V)

- DTML Documents
  - El ejemplo anterior podría renderizarse en algo así:

```
#carpeta
{
 text-color: #FFFFFF;
 border-color: #C3C3C3
}
```

# Zope Page Template (I)

- Las siguientes sentencias pertenecen al lenguaje TAL:
  - attributes,
  - content,
  - define,
  - condition,
  - omit-tag,
  - on-error,
  - repeat,
  - replace.

# Zope Page Template (II)

- `tal:attributes`
  - Cambia dinámicamente los atributos del elemento

```
<img src="imagen.png"
     alt="texto"
     tal:attributes="alt container/title" />

<img src="imagen.png"
     alt="título de la carpeta" />
```

# Zope Page Template (III)

- `tal:content`
  - Reemplaza el contenido de una etiqueta

```
<div tal:content="container/title"></div>
```

  - Se convertiría en algo así:

```
<div>El título de la carpeta</div>
```

# Zope Page Template (IV)

- `tal:define`
  - Define una variable local o global

```
<body tal:define="miTitulo
  container/title">
 <div tal:content="miTitulo"></div>
</body>
```

  - Se convertiría en algo así:

```
<body>
 <div>El título de la carpeta</div>
</body>
```

# Zope Page Template (V)

- `tal:condition`
  - Evalúa la expresión y quita la etiqueta si es falsa

```
<div tal:condition="python:container.title == ''">
  La carpeta no tiene ningún título.
</div>
<div tal:condition="python:container.title != ''"
     tal:content="container/title">
</div>
```

  - Se convertiría en alguno de estos:

```
<div>La carpeta no tiene ningún título</div>
<div>El título de la carpeta</div>
```

# Zope Page Template (VI)

- `tal:omit-tag`
  - Quita las etiquetas y deja el contenido si la expresión es verdadera

```
<b tal:omit-tag="container/noNegrita == 'True'">
 Texto resaltado o sin resaltar.
</b>
```

  - Si `container/noNegrita` es verdadero quita `<b></b>`

```
Texto resaltado o sin resaltar.
```

  - Si es falso

```
<b>Texto resaltado o sin resaltar.</b>
```

# Zope Page Template (VII)

- `tal:on-error`
  - Cuando una sentencia TAL ha producido un fallo, on-error se evalúa como una content.

```
<div tal:on-error="Error por la propiedad body"
     tal:content="container/body">
</div>
```

  - El resultado sería algo así:

```
<div>La carpeta contiene una serie de ...</div>
```

  - Pero si se produce un error:

```
<div>Error por la propiedad body</div>
```

# Zope Page Template (VIII)

- `tal:repeat`
  - Repite la etiqueta una vez por cada item de una secuencia

```
<div tal:repeat="idObjeto container/objectIds"
     tal:content="idObjeto">
</div>
```

  - El resultado sería algo así:

```
<div>objeto1</div>
<div>objeto2</div>
<div>objeto3</div>
<div>objeto4</div>
<div>objeto5</div>
```

# Zope Page Template (IX)

- `tal:replace`
  - Reemplaza la etiqueta y su contenido por el valor de una expresión

```
<h1 tal:replace="container/title">
  Aquí va el título de la carpeta!
</h1>
```

  - El resultado sería algo así:

```
<h1>El título de la carpeta.</h1>
```

# Ejemplos

- ejemplos_1

```
ejemplo01_TALES_exists.pt
ejemplo02_TALES_not.pt
ejemplo03_TALES_path.pt
ejemplo04_TALES_string.pt
ejemplo05_TALES_nocall.pt
ejemplo_METAL_1.pt
ejemplo_METAL_2.pt
ejemplo_TAL_CONTEXTS.pt
ejemplo_TAL_ordenes_sentencias.pt
ejemplo_TAL_repeat.pt
```

# DTML (I)

- Las siguientes Tags permiten hacer scripting DTML:
  - `if`, `in`,
  - `with`, `let`,
  - `call`,
  - `comment`,
  - `tree`,
  - `return`,
  - `unless`,
  - `raise`,
  - `try`, `except`.

# DTML (II)

- dtml-if
  - Implementa una condicional

```
<dtml-if nombre>
 El tipo se llama <dtml-var nombre>.
<dtml-else>
 No está definida la variable 'nombre'.
</dtml-if>
```

  - Esto tomará alguno de estos resultados:

```
El tipo se llama Arcadio.
No está definida la variable 'nombre'.
```

# DTML (III)

- `dtml-in`
  - Itera sobre una secuencia

```
<ul>
<dtml-in expr="_.range(3)">
  <li><dtml-var sequence-item></li>
</dtml-in>
</ul>
```

  - Se convertiría en algo así:

```
<ul>
  <li>0</li>
  <li>1</li>
  <li>2</li>
</ul>
```

# DTML (IV)

- dtml-with
  - Modifica el namespace, especificando donde buscar las variables

```
<dtml-with expr="REQUEST.form">
  El nombre es <dtml-var nombre>.
</dtml-with>
```

  - Imprime el parámetro nombre enviado desde un formulario:

```
El nombre es Nando.
```

# DTML (V)

- `dtml-let`
  - Añade un nuevo namespace, para definir nuevas variables

```
<dtml-let nombre='Arcadio' apell='Buendía'>
  Su nombre es <dtml-var nombre> <dtml-var apell>.
</dtml-let>
```

  - El resultado será el siguiente:

```
Su nombre es Arcadio Buendía.
```

# DTML (VI)

- `dtml-call`
  - Invoca un método pero no inserta el resultado.

```
<dtml-if expr="REQUEST.has_key('nombre')">
  <dtml-call expr="manage_changeProperties(nombre=REQUEST
   ['nombre'])">
  Propiedad cambiada.
<dtml-else>
  Ninguna propiedad cambiada.
</dtml-if>
```

  - El resultado será alguno de estos:

```
Propiedad cambiada
Ninguna propiedad cambiada
```

# DTML (VII)

- `dtml-comment`
  - Ignora el trozo de código que encierra

```
<dtml-comment>
 El comentario no se verá
 <dtml-var nombre>
</dtml-comment>
Pero esto sí.
```

  - El resultado será el siguiente:
    - `Pero esto sí.`

# DTML (VIII)

- `dtml-return`
  - Fuerza al script DTML a devolver un valor

```
<dtml-var nombre>
Texto de sobra.
<dtml-return 'El retorno es este.'>
Este otro texto ni se procesará.
```

  - El resultado será el siguiente:

```
El retorno es este.
```

# DTML (X)

- `dtml-unless`
  - Ejecuta un bloque de código a menos que la expresión sea cierta.

```
Puede que exista su nombre.
<dtml-unless expr='not nombre'>
  Su nombre existe.
</dtml-unless>
```

  - El resultado será alguno de los siguientes:

```
Puede que exista su nombre.
```

```
Puede que exista su nombre.
Su nombre existe.
```

# DTML (XI)

- `dtml-raise`
  - Devuelve un error tipificado.

```
<dtml-raise type="404">
 Recurso no encontrado
</dtml-raise>
```

# DTML (XII)

- dtml-try, dtml-except
  - Captura posibles excepciones

```
<dtml-try>
 Resultado: <dtml-var expr="_.float(a/b)"> unids.
<dtml-except ZeroDivisionError>
 Resultado: N/A
</dtml-try>
```

  - El resultado podría ser alguno de los siguientes:

```
Resultado: 5 unids.
Resultado: N / A
```

# Ejemplos

- ejemplos_2

```
ejemplo_1.pt
ejemplo_2.pt
ejemplo_3.pt
ejemplo_4.pt
ejemplo_5.pt
ejemplo_6.pt
ejemplo_7.pt
ejemplo_8.pt
ejemplo_9.pt
...
ejemplo_15.pt
```

```
ejemplo_1.dtml
ejemplo_2.dtml
ejemplo_3.dtml
ejemplo_4.dtml
ejemplo_5.dtml
ejemplo_6.dtml
ejemplo_7.dtml
ejemplo_8.dtml
ejemplo_9.dtml
...
ejemplo_15.dtml
```

# La adquisición (I)

- Mecanismo que permite a un objeto A obtener atributos de su entorno
- Un objeto B pertenece al entorno de A si:
  - aparece en el contexto (path) de invocación de A
    - http://localhost:8080/C/D/E/B/A
  - o bien A 'contiene' a B
    - / A /
    - / A / B
    - / A / C /

# La adquisición (II)

- En la práctica, es como si el objeto A tubiese todos los atributos de B.
- Si tenemos estas carpetas y objetos:
  - / B /
  - / B / A
  - / B / C /
  - / B / C / D
- ¿Podemos hacer las siguientes invocaciones?
  - http://localhost:8080/B/A
  - http://localhost:8080/B/C/A
  - http://localhost:8080/B/C/D/C/B/D/A

# La adquisición (III)

- Ejercicio:
  - crear las siguientes carpetas:
    - `/animales`
    - `/animales/aves/`
    - `/animales/aves/gallina/`
    - `/animales/perro/`

# La adquisición (IV)

- dentro de la carpeta '`animales`' crear un objeto '`Page Template`' llamado '`hago_como_animal`':

```
<html>
  <head>
    <title tal:content="template/title">
    </title>
  </head>
  <body>
    Hago como animal
  </body>
</html>
```

# La adquisición (V)

- dentro de la carpeta '`ave`' crear un objeto '`Page Template`' llamado '`hago_como_ave`':

```
<html>
  <head>
    <title tal:content="template/title">
    </title>
  </head>
  <body>
    Hago como ave
  </body>
</html>
```

# La adquisición (VI)

- dentro de la carpeta '`gallina`' crear un objeto '`Page Template`' llamado '`hago_como_gallina`':

```
<html>
  <head>
    <title tal:content="template/title">
    </title>
  </head>
  <body>
    Hago como gallina
  </body>
</html>
```

# La adquisición (VII)

- dentro de la carpeta '`perro`' crear un objeto '`Page Template`' llamado '`hago_como_perro`':

```
<html>
  <head>
    <title tal:content="template/title">
    </title>
  </head>
  <body>
    Hago como perro
  </body>
</html>
```

# La adquisición (VIII)

- probar las siguientes URLs:

- `127.0.0.1:8080/animales/hago_como_animal`
- `127.0.0.1:8080/animales/perro/hago_como_perro`
- `127.0.0.1:8080/animales/aves/hago_como_ave`

- `127.0.0.1:8080/animales/perro/hago_como_animal`
- `127.0.0.1:8080/animales/aves/hago_como_animal`

- `127.0.0.1:8080/animales/perro/aves/hago_como_perro`
- `127.0.0.1:8080/animales/aves/perro/hago_como_aves`

- `127.0.0.1:8080/animales/perro/hago_como_gallina`
- `127.0.0.1:8080/animales/aves/hago_como_gallina`

- `127.0.0.1:8080/animales/aves/perro/aves/perro/aves/perro/aves/perro/hago_como_ave`
- `127.0.0.1:8080/animales/aves/perro/gallina/aves/perro/gallina/aves/perro/aves/perro/hago_como_gallina`

# Python scripting en Zope (I)

- Invocación de un script
- Variables de enlace (binding)
- Paso de parámetros
- Restricciones de seguridad
- Scripts Externos

# Python scripting en Zope (II)

- Para invocar un objeto Zope `Script (Python)`:
  - Basta poner su URL en el navegador
  - Puede aparecer en el campo `action` de un formulario web
  - Podemos hacer que otro objeto lo adquiera y tratar el script como un método propio más
    - de esta forma, el objeto que lo adquiere pasa a ser su contexto

# Python scripting en Zope (III)

- El objeto `Script (Python)` recibe una representación de su entorno
- mediante las variables de enlace
  - `context` es el objeto que ha invocado el script
  - `container` es el objeto que contiene el script
  - `script` es el objeto script en sí mismo
  - `container.REQUEST` es un wrapper que contiene variables relacionadas con la invocación del script
  - `container.REQUEST.RESPONSE` con variables relacionadas con la respuesta al cliente que hizo la petición

# Python scripting en Zope (IV)

- Para pasar parámetros a través de una URL
  - `http://localhost:8080/script?parm1=val1&parm2=val2`
- Desde un formulario, se puede especificar el tipo de estos

```
<form action="script">
 edad <input type="text" name="edad:int">
 <input type="checkbox" name="colores:list” value="rojo">
 <input type="checkbox" name="colores:list” value="verde">
 <input type="checkbox" name="colores:list” value="azul">
</form>
```

# Python scripting en Zope (V)

- Por seguridad:
  - Se restringe el número de iteraciones en los bucles
  - No se pueden importar todos los módulos
  - Cada vez que se accede un objeto, se realiza un chequeo de permisos
- Para evitar estas restricciones:
  - `External Methods`
  - el objeto conoce la ruta física (en el servidor) donde se encuentra el script (no es un objeto, es un fichero.py)

# Web Scripting en Zope (I)

- Zope encapsula por completo el protocolo HTTP, facilitando los siguientes objetos para interactuar con él:

  `REQUEST` (llamada HTTP)

  `RESPONSE` (respuesta HTTP)

  `SESSION` (sesión del usuario)

- A partir de `REQUEST` se pueden obtener los otros dos:

```
REQUEST.RESPONSE
REQUEST.SESSION
```

# Web Scripting en Zope (II)

- `REQUEST` está disponible en todo ZPT a través de "`request`"
  - Para acceder a los valores de las variables ya sean `POST`, `GET` o `Cookies`, es tan sencillo como escribir
  - `REQUEST['nombre_variable']`
  - Como `REQUEST` es un diccionario podemos usar: `has_key` o `keys`

# Web Scripting (III)

- Para guardar un valor en la session utilizar

```
REQUEST.SESSION['nombre_variable']=valor
```

- y para borrarla

```
del (REQUEST.SESSION['nombre_variable'])
```

- El objeto `RESPONSE` se utiliza para redirigir llamadas HTTP hacia otra página o método:

```
REQUEST.RESPONSE.redirect('url/de/destino')
```

# Web Scripting en Zope (IV)

- `RESPONSE` también se utiliza para grabar Cookies:

```
REQUEST.RESPONSE.setCookie('nombre_cookie', 'valor')
```

- El método u objeto por defecto en Zope se llama `index_html`. Si se produce una llamada a un objeto de la forma
  - http://localhost:8080/objeto
  - Zope buscará un objeto o método llamado index_html y lo ejecutará

# Web Scripting en Zope (V)

- Page Template que muestra las variables:

```
<html>
  <body>
    <span tal:repeat="variable
   python:request.keys()">
       <p tal:replace="variable"/> = <p
   tal:replace="python:request[variable]"/> <br>
    </span>
  </body>
</html>
```

# Ejemplos

- ejemplos_3

```
ejemplo_formulario_1.pt
ejemplo_formulario_2.pt
```

# Búsquedas en catálogos (I)

- el objeto `ZCatalog`:
  - sólo es capaz de indexar los objetos contenidos por su contenedor
  - hay que definir una lista de índices y otra de metadatos
  - Indexar objetos significa, por cada índice y objeto
    - ver si el objeto tiene un atributo o método con el mismo nombre que el índice:
      - invocarlo (si es un método) o recuperlo si es un atributo
      - completar la entrada en cada tabla (índices y metadatos)

# Búsquedas en catálogos (II)

- el objeto `Z Search Inteface`:
  - se crean dos objetos
    - `Page Template` o `DTML Method`
    - un formulario para introducir los parámetros de búsqueda
    - un documento que lista los resultados
- desde script python:

```
return context.Catalog({'title':'tituloaBuscar'})
```

# Búsquedas en catálogos (III)

- Crea una carpeta llamada '`libros`'.
- Crea dentro de ella, 3 '`Page Template`'s:
  - 1. '`lord_of_de_rings`', con el título '`El señor de los anillos`', edítala y escribe el siguiente código :

```
<html>
  <head><title tal:content="template/title">título</title>
  </head>
  <body>
    <p>Cuando el señor Bilbo Bolsón de Bolsón Cerrado anunció que muy
    pronto celebraría su cumpleaños centesimodecimoprimero con una
    fiesta de especial magnificencia, hubo muchos comentarios y
    excitación en Hobbiton.</p>
  </body>
</html>
```

# Búsquedas en catálogos (IV)

- Crea también la siguente '`Page Template`':
  - 2.'`el_nombre_de_la_rosa`', con el título '`El nombre de la rosa`', edítala y escribe las siguientes líneas de código :

```
<html>
  <head>
    <title tal:content="template/title">título</title>
  </head>
  <body>
    <p>En el principio era el Verbo y el Verbo era en Dios, y el Verbo
    era Dios. Esto era en el principio, en Dios, y el monje fiel
    debería repetir cada día con salmodiante humildad ese
    acontecimiento inmutable cuya verdad es la única que puede
    afirmarse con certeza incontrovertible.</p>
  </body>
</html>
```

# Búsquedas en catálogos (V)

- y la última '`Page Template`':
  - 3. '`los_pilares_de_la_tierra`', con el título '`Los pilares de la tierra`', edítala y escribe las siguientes líneas de código :

```
<html>
  <head>
    <title tal:content="template/title">título</title>
  </head>
  <body>
    <p>Los chiquillos llegaron temprano para el ahorcamiento. Todavía
    estaba oscuro cuando los tres o cuatro primeros se escurrieron con
    cautela de las covachas, sigilosos como gatos, con sus botas de
    fieltro.</p>
  </body>
</html>
```

# Búsquedas en catálogos (VI)

- Crea un objeto `ZCatalog` en la carpeta '`libros`' con el id '`Catalog`'
- Entra en '`Catalog/manage`', en el tab '`Indexes`' y elimina todos los índices excepto '`PrincipiaSearchSource`' y '`title`'
- Entra en '`Catalog/manage`' una vez más, en el tab '`Metadata`' y crea un nuevo metadato llamado '`absolute_url`'

# Búsquedas en catálogos (VII)

- De nuevo en '`Catalog/manage`', en el tab '`Find Objects`' selecciona '`Page Template`' en el campo '`Find Objects by Type`' y pulsa el botón '`Find and Catalog`'
- Ahora en la carpeta '`libros`' creamos un objeto de tipo '`Z Search Interface`'.
- Seleccionamos '`Catalog`' en el campo '`Select one or more searchable objects`'
- En el campo '`Report id`' introduciremos '`search_result`' y en '`Report Title`' '`Resultados de la búsqueda`'

# Búsquedas en catálogos (VIII)

- En el campo ‘`Search Input id`’ introduciremos ‘`search_form`’ y en ‘`Search Input Title`’ ‘`Formulario de búsqueda`’
- Elegimos ‘`Generate Page Templates`’ y pulsamos el botón ‘`Add`’
- Ponemos en el navegador la URL
  `localhost:8080/libros/search_form`
- Realizamos una búsqueda y observamos los resultados

# Búsquedas en catálogos (IX)

- Creamos un Python Script en la carpeta '`libros`' llamado '`que_libros`'
- Lo editamos con el siguiente código :

```
request = container.REQUEST
la_palabra = request.palabra
for libro in container.Catalog({'PrincipiaSearchSource' :
    la_palabra}):
  print libro.title + "\n"
return printed
```

- Ponemos en el navegador la URL
  - `localhosts:8080/libros/que_libros?palabra='Hobbiton`
- y observamos los resultados

# Aplicaciones Zope en Python (I)

- un producto:
  - directorios, estructura básica

```
/ Library /
/ Library / __init__.py
/ Library / book.py
/ Library / www /
/ Library / www /add_form.pt
/ Library / img /
/ Library / img / book.png
```

# Aplicaciones Zope en Python (II)

- `__init__.py`

```
import book

def initialize(context):
    """ add to the product list """
    context.registerClass(
        book.Book,
        constructors = (book.add_form,
                        book.add),
        icon='img/book.png')
```

# Aplicaciones Zope en Python (III)

- `add_form.pt`

```
<html>
<head><title>Add Form</title></head>
<body>
 <form action="add" method="post" >
  id    <input type="text" name="id"/><br/>
  title <input type="text" name="title"/><br/>
  <input type="submit" name="submit" value="Add"/>
 </form>
</body>
</html>
```

ZOPE Community

# Aplicaciones Zope en Python (IV)

- book.py (I)

```
# Zope modules
from Globals import Persistent
from Acquisition import Implicit
from OFS.SimpleItem import Item
from AccessControl import ClassSecurityInfo

# Modules from other products
from Products.PageTemplates.PageTemplateFile \
  import PageTemplateFile
```

# Aplicaciones Zope en Python (V)

- book.py (II)

```
# class constructor form
add_form = \
    PageTemplateFile('www/add_form.pt',globals())

def add(self, id, title, REQUEST=None):
    """ action of add_form """
    self._setObject(id, Book(id, title))
    return 'Ok.'
```

# Aplicaciones Zope en Python (VI)

- book.py (III)

```
class Book(Implicit,Persistent,Item):
      """ A Book """
      meta_type = 'Book'
      security = ClassSecurityInfo()

      def __init__(self, id, title):
     """ init"""
     self.id = id
     self.title = title
```

# Aplicaciones Zope en Python (VII)

- book.py (IV)

```
security.declareProtected('View','index_html')

def index_html(self,REQUEST=None,RESPONSE=None):
    """ view """
    return "El libro '"  +str(self.id)+    "', " \
          +"se titula '" +str(self.title)+ "'."
```

# Ejercicio 1

- Crear nuevo tipo de Objetos Zope basado en `Folder`
- Pistas:
  - Crearemos un nuevo producto llamado `MiCarpeta`
  - Definiremos una clase `MiCarpeta` que herede de `Folder`

```
from OFS.Folder import Folder
```

  - Utilizaremos el método `__init__` de la clase padre

```
MiCarpeta.__bases__[-1].__init__(self, id=id)
```

# Ejercicio 2 (I)

- Añadir al producto `MiCarpeta` una web de edición.
- Pistas:
  - Creamos un plantilla llamada `edit_form.pt` dentro de la carpeta:

```
<form action='edit'> ... </form>
```

  - En el método `edit` de la clase `MiCarpeta` :

```
self.manage_editProperties(REQUEST)
```

# Ejercicio 2 (II)

- Pistas:
  - Para saber donde estamos en el sistema de ficheros (en `miCarpeta.py`)

```
from Globals import package_home
file_path = package_home(globals())
```

  - Desde el método `add` leemos el fichero y creamos un objeto `ZPT`

```
f=open(file_path+'/www/edit_form.pt')
src=f.read()
f.close()
obj = getatrr(self,id)
obj.manage_addProduct['PageTemplates'].\ manage_addPageTemplate
   (id='edit_form',
                          title='', file=src)
```

# Referencias

- http://www.zopemag.com/Issue003/Section_Articles/article_ZPTintro.html
- http://www.zopemag.com/Issue002/Section_Articles/article_DbugZPTs_01.html

- http://www.zope.org/Wikis/DevSite/Projects/ZPT/TAL%20Specification%201.4
- http://www.zope.org/Wikis/DevSite/Projects/ZPT/TALES%20Specification%201.3
- http://www.zope.org/Wikis/DevSite/Projects/ZPT/METAL%20Specification%201.0

- http://www.zope.org/Members/peterbe/DTML2ZPT/
- http://www.zope.org/Members/Zen/howto/FormVariableTypes

- http://www.zope.org/Documentation/Books/ZopeBook/2_6Edition/ZPT.stx
- http://www.zope.org/Documentation/Books/ZopeBook/2_6Edition/AdvZPT.stx

- http://www.devshed.com/c/a/Zope/ZPT-Basics-part-1/
- http://www.devshed.com/c/a/Zope/ZPT-Basics-part-2/
- http://www.devshed.com/c/a/Zope/ZPT-Basics-part-3/
- http://www.devshed.com/c/a/Zope/ZPT-Basics-part-4/