

Desarrollo de Componentes Web con zope3

Nando Quintana Hernández
fquintana@codesyntax.com



Algunos derechos reservados...

<http://creativecommons.org/licenses/by-sa/2.5/es/>

Contenidos (I)

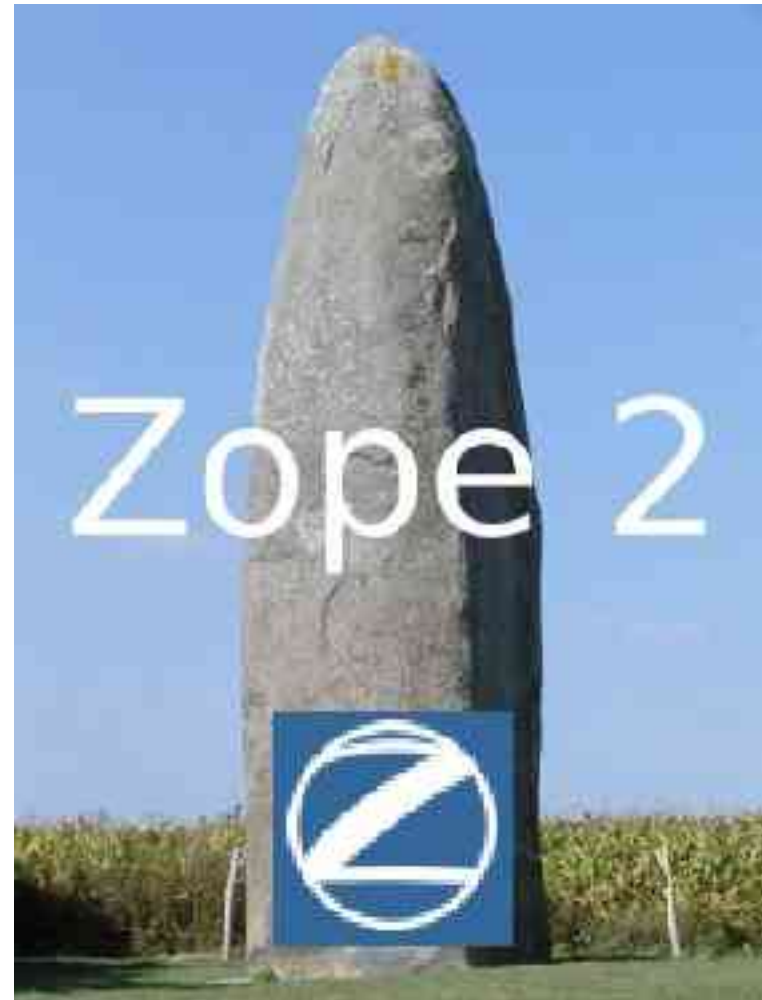
- Yet Another Zope (I-III)
- Interfaces y Adaptadores (I-XIII)
- Esquemas y Widgets (I-VI)
- Contenedores (I-IX)
- Creación de Módulos Zope (I-XI)
- Factorías (I-III)

Contenidos (II)

- Vistas (I-VII)
- Otros Adaptadores (I-III)
- i18n y l10n (I-VII)
- Layers y Skins (I-VIII)
- Metadatos (I-V)
- En el tintero... (I-?)

Yet Another Zope (I)

- Zope2 monolítico
 - ▶ 1 responsabilidad
 - ▶ 1 clase base
- Clases “mixtura”
 - ▶ herencia múltiple
- Problemillas
 - herencia de métodos sin implementar
 - “ugly hacks” para cambiar una coma



Yet Another Zope (II)

- Zope3
 - ▶ 1 responsabilidad
 - ▶ 1 componente
- Delegación
 - ▶ distintos objetos
- Intercambiables y reutilizables
 - alta cohesión
 - bajo acoplamiento



Yet Another Zope (III)

- Curva de aprendizaje más suave [1]
- Modelo de programación más familiar
 - ▶ Sobre todo para programadores de Python
 - ▶ Utilizar código Python en Zope con pocos cambios
- Software enfoque KISS (Keep It Simple Stupid)
 - ▶ Más reusabilidad
 - ▶ Mejor documentación y pruebas
 - ▶ Fácil I18n y L10n

Interfaces y Adaptadores (I)

- La funcionalidad que un componente promete, viene redactado en un contrato público llamado *Interfaz*.
- En él se describe qué métodos y atributos se compromete a tener.
 - ▶ Qué pero no Cómo.
- En Python no existe este mecanismo,
 - ▶ El paquete Zope: `zope.interface`

Interfaces y Adaptadores (II)

- ejemplo00.py

[2]

```
#!/usr/bin/python
```

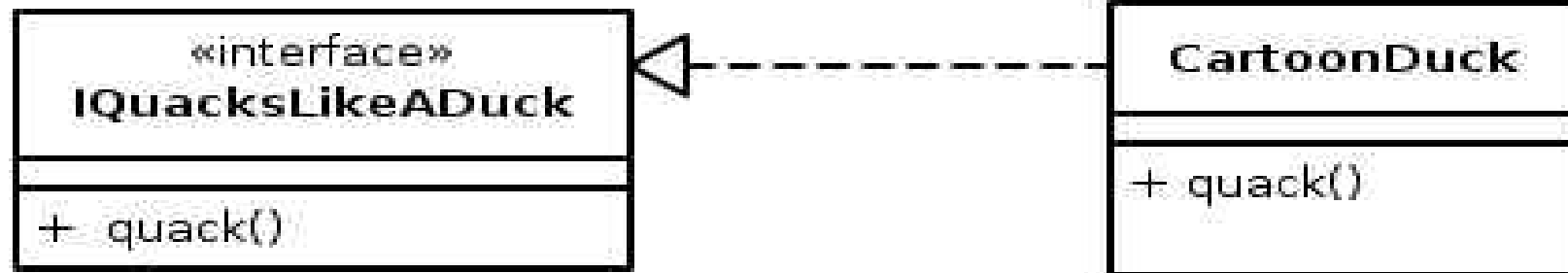
```
## ejemplo00.py (parte 1)
```

```
from sys import path
```

```
path.append('/usr/local/Zope-3.2.0b3/lib/python/')
```

```
...
```

Interfaces y Adaptadores (III)



Interfaces y Adaptadores (IV)

```
## ejemplo00.py (parte 2)

from zope.interface import Interface, implements, providedBy

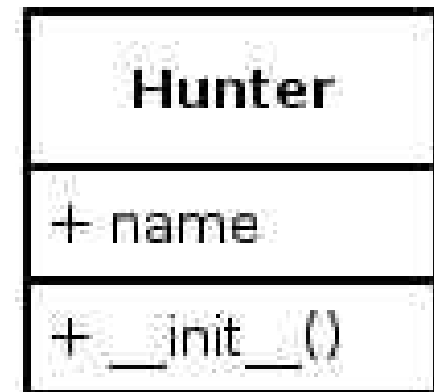
class IQuacksLikeADuck(Interface):
    def quack():
        """ Returns a quacking sound. """

class CartoonDuck(object):
    implements(IQuacksLikeADuck)

    def quack(self):
        return "The Cartoon Duck goes 'quack!'"

lucas = CartoonDuck()
print IQuacksLikeADuck.providedBy(lucas) # True
print lucas.quack() # The Cartoon Duck goes...
```

Interfaces y Adaptadores (V)



Interfaces y Adaptadores (VI)

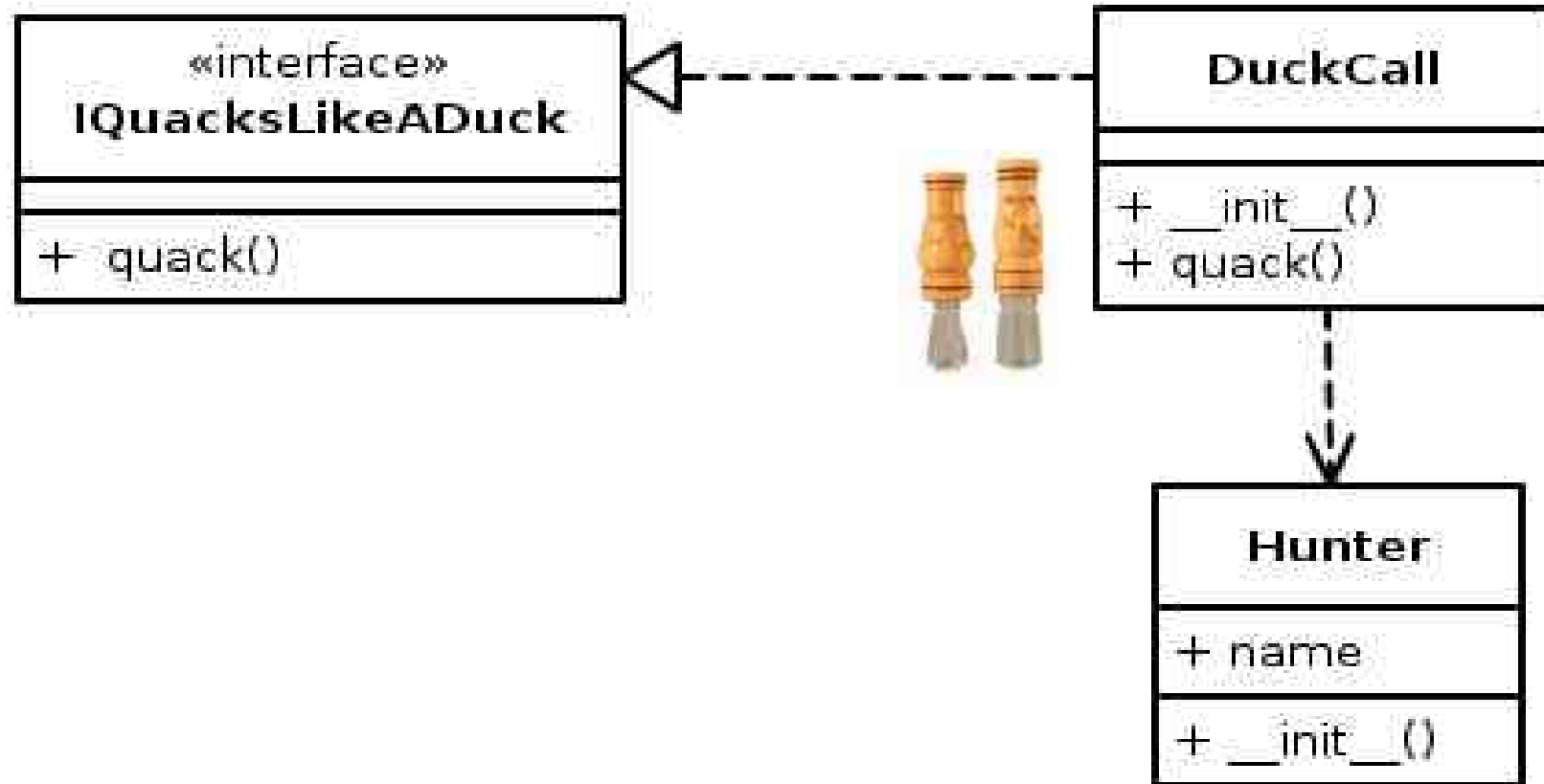
```
## ejemplo00.py (parte 3)

from zope.interface import providedBy

class Hunter(object):
    def __init__(self, name):
        self.name = name

elmer = Hunter('Elmer')
print IquacksLikeADuck.providedBy(elmer) # False
print elmer.quack()                       # Error
```

Interfaces y Adaptadores (VII)



Interfaces y Adaptadores (VIII)

```
## ejemplo00.py (parte 4)

from zope.interface import implements
from zope.component import adapts, provideAdapter

class DuckCall(object):
    implements(IQuacksLikeADuck)
    adapts(Hunter)

    def __init__(self, hunter):
        self.hunter = hunter

    def quack(self):
        return self.hunter.name + ' quacks with a duck call'

provideAdapter(DuckCall)
```

Interfaces y Adaptadores (IX)

```
## ejemplo00.py (parte 5)
```

```
from zope.component import getAdapter, queryAdapter
```

```
# elmer = getAdapter(elmer, interface=IquacksLikeADuck)
```

```
# elmer = queryAdapter(elmer, interface=IQuacksLikeADuck, default=None)
```

```
elmer = IquacksLikeADuck(elmer)
```

```
print IquacksLikeADuck.providedBy(elmer) # True
```

```
print elmer.quack() # Elmer quacks with a ...
```

Interfaces y Adaptadores (X)

- ejemplo01.py

[2,3]

```
#!/usr/bin/python
## ejemplo01.py (parte 1)

from sys import path
path.append('/usr/local/Zope-3.2.0b3/lib/python/')

...
```

Interfaces y Adaptadores (XI)

```
## ejemplo01.py (parte 2)

from zope.interface import Interface

class IQuacksLikeADuck(Interface):
    def quack():
        """ Returns a quacking sound. """

class Duck(object):

    def quack(self):
        return "The best quack is a duck quack."
```

Interfaces y Adaptadores (XII)

```
## ejemplo01.py (parte 3)
```

```
from zope.interface import providedBy,  
from zope.interface import directlyProvides, classImplements
```

```
potas = Duck()
```

```
print IQuacksLikeADuck.providedBy(potas)           # False
```

```
directlyProvides(potas, IQuacksLikeADuck)
```

```
print IQuacksLikeADuck.providedBy(potas)           # True
```

```
hunted = Duck()
```

```
print IQuacksLikeADuck.providedBy(hunted)           # False
```

```
classImplements(Duck, IQuacksLikeADuck)
```

```
print IQuacksLikeADuck.providedBy(hunted)           # True
```

Interfaces y Adaptadores (XIII)

```
## ejemplo01.py (parte 4)

from zope.interface.verify import verifyClass

print verifyClass(IQuacksLikeADuck, Duck)           # True

class NoDuck(object):
    pass

print verifyClass(IQuacksLikeADuck, NoDuck)         # Error
```

Esquemas y Widgets (I)

- Un esquema (schema) es un interfaz cuyos atributos son objetos especiales llamados campos (fields).
- Los campos están implementados en forma de métodos
 - ▶ y están definidos en el paquete `zope.schema`.
- Otorgan ciertas facilidades como:
 - ▶ metadatos, generación automática de formularios de entrada robusta, etc.

Esquemas y Widgets (II)

- ejemplo02.py

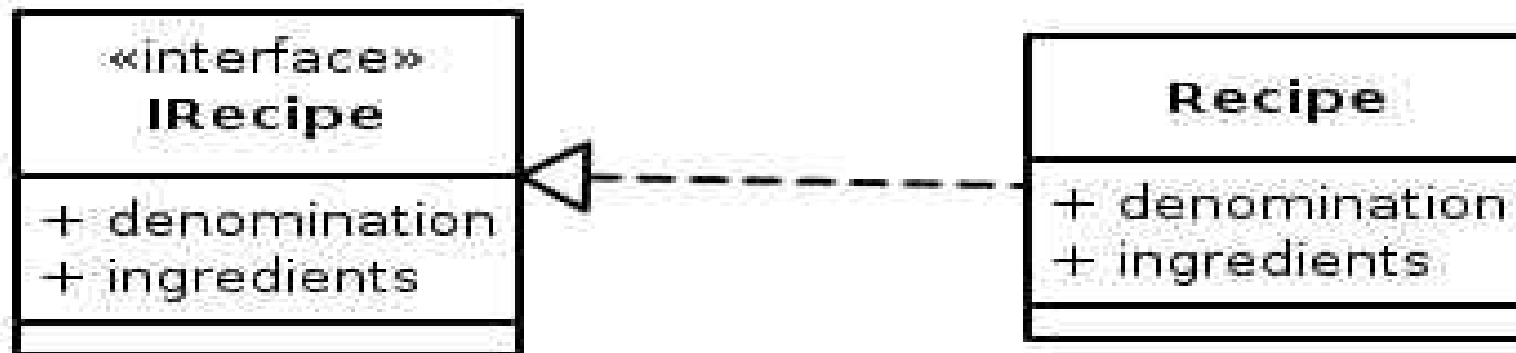
[3,4]

```
#!/usr/bin/python
## ejemplo02.py (parte 1)

from sys import path
path.append('/usr/local/Zope-3.2.0b3/lib/python/')

...
```

Esquemas y Widgets (III)



Esquemas y Widgets (IV)

```
## ejemplo02.py (parte 2)
```

```
from zope.interface import Interface
from zope.schema import TextLine, List
```

```
class IRecipe(Interface):
```

```
    """ Store information about a recipe """
```

```
    denomination = TextLine( title = u"Denomination",
                             description = u"Name of the dish",
                             required = True )
```

```
    ingredients = List( title = u"Ingredients",
                        description = u"List of ingredients.",
                        required = True,
                        value_type = TextLine(title = u"Ingredient") )
```

Esquemas y Widgets (V)

```
## ejemplo02.py (parte 3)
```

```
from zope.interface import implements
from zope.schema.fieldproperty import FieldProperty
```

```
class Recipe(object):
    implements(IRecipe)

    denomination = FieldProperty( IRecipe['denomination'] )

    ingredients = FieldProperty( IRecipe['ingredients'] )
```

Esquemas y Widgets (VI)

```
## ejemplo02.py (parte 4)
```

```
from zope.publisher.browser import TestRequest
from zope.app.form.browser import TextWidget, ListSequenceWidget

field = IRecipe['denomination']
request = TestRequest( form = {'field.denomination': u'Porrusalda'} )
widget = TextWidget(field, request)

print widget().replace(' ', '\n ')           # <input...

field = IRecipe['ingredients']
request = TestRequest()
widget = ListSequenceWidget(context=field, field=None, request=request)

print widget().replace(' ', '\n ')           # <table...
```

Contenedores (I)

- Un componente contenedor se comporta como un diccionario de Python
 - ▶ agrupa otros objetos y permite recuperarlos por su nombre
- IContainer: API de un contenedor
- Se puede añadir un objeto a un contenedor con `__setitem__` aunque se pueden explicitar restricciones

Contenedores (II)

- ejemplo03.py

[5]

```
#!/usr/bin/python
```

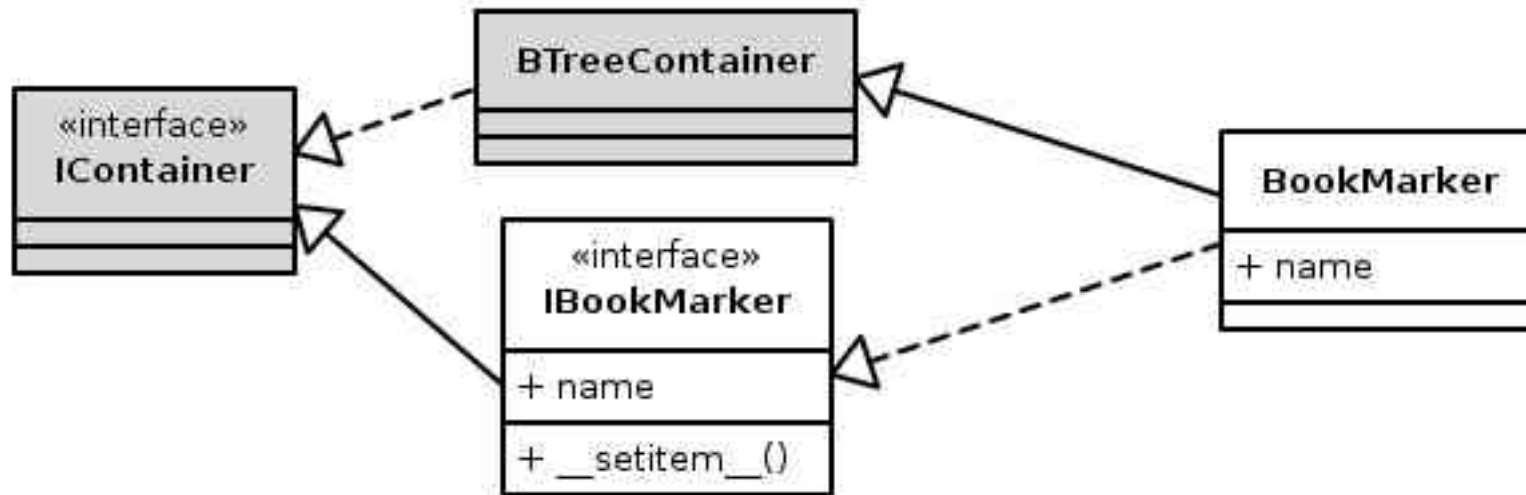
```
## ejemplo03.py (parte 1)
```

```
from sys import path
```

```
path.append('/usr/local/Zope-3.2.0b3/lib/python/')
```

```
...
```

Contenedores (III)



Contenedores (IV)

```
## ejemplo03.py (parte 2)
```

```
from zope.schema import TextLine
from zope.app.container.constraints import ItemTypePrecondition
from zope.app.container.interfaces import IContainer
```

```
class IBookMarker(IContainer):
```

```
    """This is the container for all book marks."""
```

```
    name = TextLine( title=u"Name of BookMarker",
                    description=u"A name for BookMarker",
                    default=u"",
                    required=True )
```

```
    def __setitem__(name, obj):
        pass
```

```
    __setitem__.precondition = ItemTypePrecondition(IMark)
```

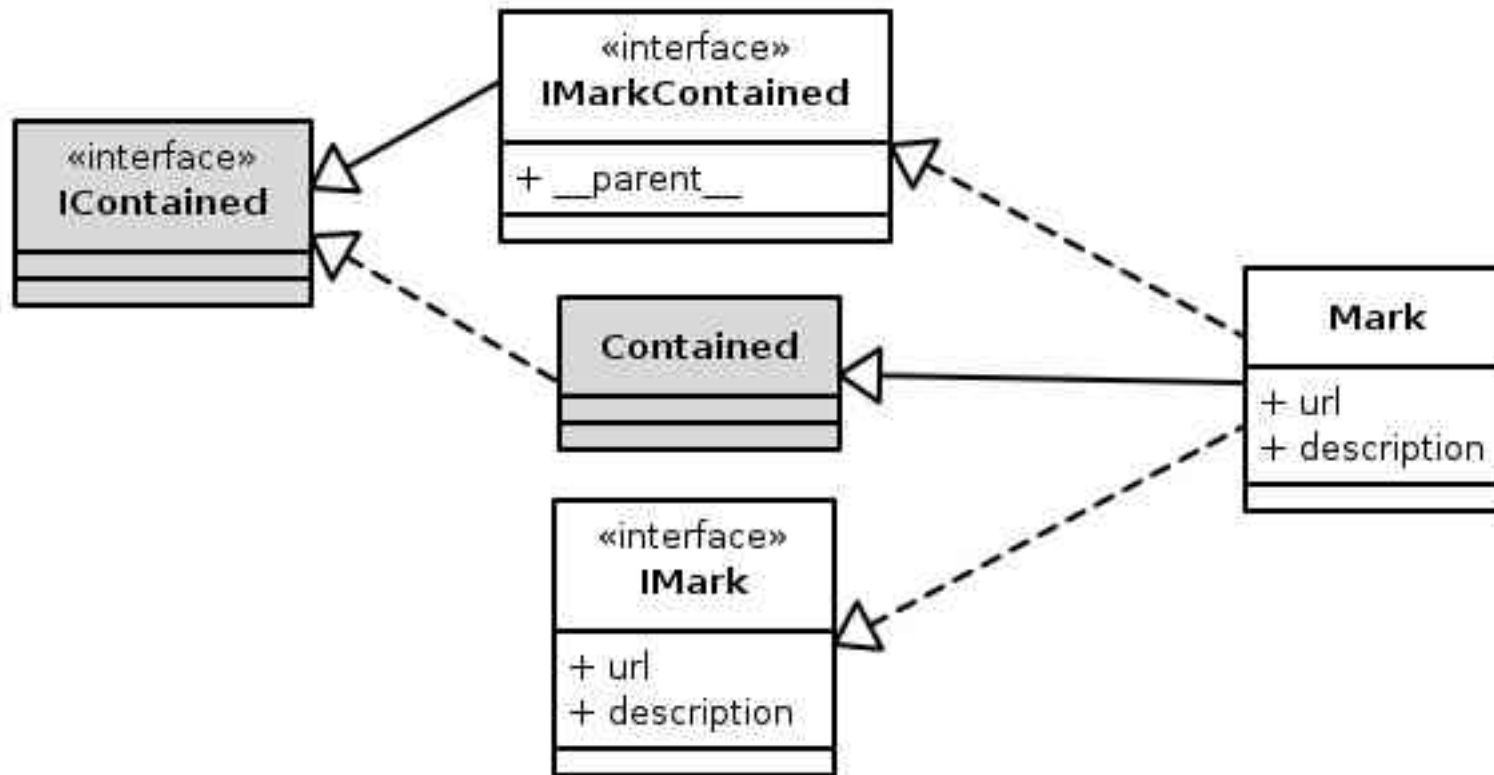
Contenedores (V)

```
## ejemplo03.py (parte 3)
```

```
from zope.app.container.btree import BTreeContainer  
from zope.interface import implements
```

```
class BookMarker(BTreeContainer):  
    """Implementation of IBookMarker using B-Tree Container"""  
    implements(IBookMarker)  
  
    name = u""
```

Contenedores (VI)



Contenedores (VII)

```
## ejemplo03.py (parte 4)
```

```
from zope.interface import Interface
from zope.schema import TextLine
```

```
class IMark(Interface):
```

```
    """This is the book mark object."""
```

```
    url = TextLine( title=u"URL/Link",
                    description=u"URL of the website",
                    default=u"http://www.zope.org",
                    required=True )
```

```
    description = Text( title=u"Description",
                        description=u"Description of the website",
                        default=u"",
                        required=False )
```

Contenedores (VIII)

```
## ejemplo03.py (parte 5)
```

```
from zope.app.container.contained import Contained
from zope.app.container.interfaces import IContained

class IMarkContained(IContained)
    """A book mark can only contain in a BookMarker"""

    __parent__ = Field ( constraint =
                        ContainerTypesConstraint(IBookMarker)
                        )

class Mark(Contained):
    """Implementation of IMark"""
    implements(IMark, IMarkContained)

    url = u"http://www.zope.org"
    description = u""
```

Contenedores (IX)

```
## ejemplo03.py (parte 6)
```

```
from zope.app.container.contained import setitem
```

```
estanteria = BookMarker()
```

```
mi_libro = Mark()
```

```
setitem(estanteria, estanteria.__setitem__, u"Mi Libro", mi_libro)
```

```
estanteria[u"Mi Libro"].url = "Este es mi libro"
```

```
print estanteria[u"Mi Libro"].url # Este es mi ...
```

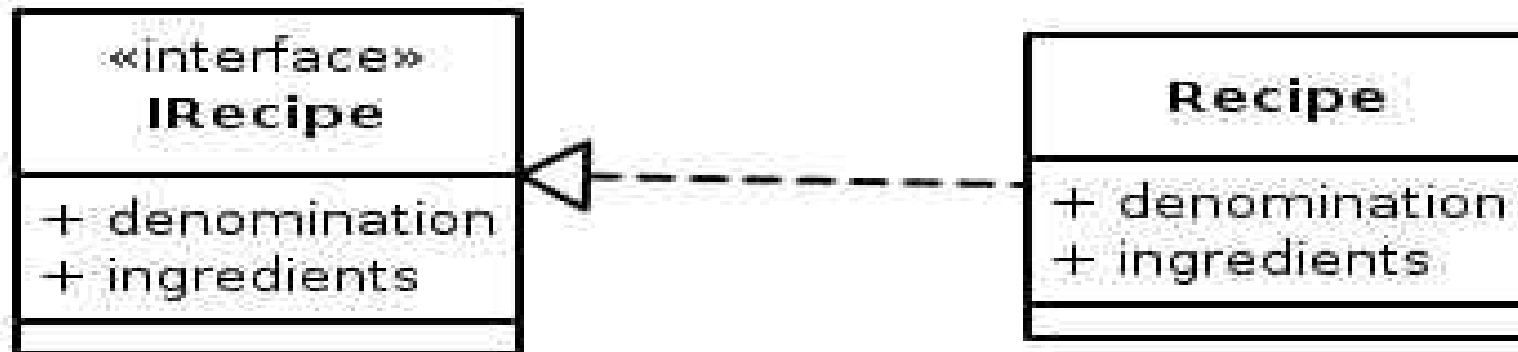
Creación de Módulos Zope (I)

- Para que una instancia conozca nuestros interfaces:
 - ▶ crear un módulo Python en:
 - `lib/python/modulo/`
 - ▶ con nuestros interfaces y clases:
 - `/lib/python/modulo/__init__.py`
 - `/lib/python/modulo/interfaces.py`
 - `/lib/python/modulo/modulo.py`
 - ...

Creación de Módulos Zope (II)

- Además:
 - ▶ crear un fichero de configuración en:
 - `/lib/python/modulo/configure.zcml`
 - ▶ y otro en:
 - `etc/package-includes/modulo-config.zcml`

Creación de Módulos Zope (III)



Creación de Módulos Zope (IV)

- `/lib/python/ejemplo04/__init__.py`

```
## este fichero convierte automáticamente  
## el directorio en un módulo  
  
## puede ser un fichero vacío
```

Creación de Módulos Zope (V)

- `.../ejemplo04/interfaces.py`

```
from zope.interface import Interface
from zope.schema import TextLine, List
```

```
class IRecipe(Interface):
```

```
    """ Store information about a recipe """
```

```
    denomination = TextLine( title = u"Denomination",
                             description = u"Name of the dish",
                             required = True )
```

```
    ingredients = List( title = u"Ingredients",
                        description = u"List of ingredients.",
                        required = True,
                        value_type = TextLine(title = u"Ingredient") )
```

Creación de Módulos Zope (VI)

- .../ejemplo04/recipe.py

```
from persistent import Persistent
from zope.interface import implements
from ejemplo04.interfaces import IRecipe
```

```
class Recipe(Persistent):
    implements(IRecipe)
```

```
    denomination = u''
    ingredients = []
```

Creación de Módulos Zope (VII)

- .../ejemplo04/configure.zcml (I)

```
<configure xmlns="http://namespaces.zope.org/zope"  
           xmlns:i18n="http://namespaces.zope.org/i18n"  
           i18n_domain="ejemplo04"
```

```
>
```

```
<interface  
    interface=".interfaces.IRecipe"  
    type="zope.app.content.interfaces.IContentType"
```

```
/>
```

```
<!--
```

Con esto añadimos un nuevo "Content Type" a nuestra instancia. Nuestro interfaz es, también, un tipo de contenido. Ahora, en nuestra instancia, podremos añadir nuevos objetos de este tipo

Creación de Módulos Zope (VIII)

- .../ejemplo04/configure.zcml (II)

```
<content class=".recipe.Recipe" >
```

```
<require
```

```
    permission=".zope.View"
```

```
    interface=".interfaces.IRecipe"
```

```
/>
```

```
<!--
```

Le decimos al "security proxy" que cree un wrapper de seguridad alrededor de cada objeto "Recipe" que exija el permiso ".zope.View" cada vez que quieran acceder a alguno de los atributos definidos en el schema IRecipe

```
-->
```

Creación de Módulos Zope (IX)

- .../ejemplo04/configure.zcml (III)

```
<require  
    permission=".zope.ManageContent"  
    set_schema=".interfaces.IRecipe"  
>
```

```
<!--
```

De la misma forma, este wrapper de seguridad exigirá el permiso ".zope.ManageContent" cada vez que quieran modificar alguno de los atributos definidos en el schema IRecipe

```
-->
```

```
</content>
```

```
</configure>
```

Creación de Módulos Zope (X)

- etc/package-includes/
ejemplo04-configure.zcml

```
<include package="ejemplo05"/>
```

```
<!--
```

```
    Gracias a esta directiva, la instancia de zope,  
    parseará al arrancar el fichero configure.zcml  
    de nuestro módulo
```

```
-->
```

Creación de Módulos Zope (XI)

- ejemplo05.py

```
#!/usr/bin/python
from sys import path
path.append('/usr/local/Zope-3.2.0/lib/python/')
path.append('/var/zope3/third/lib/python/')

from zope.app.debug import Debugger
debugger = Debugger(db="var/Data.fs", config_file="etc/site.zcml")

from ejemplo04.recipe import Recipe

porrusalda = Recipe()
porrusalda.denomination = u"Porrusalda Casera"
porrusalda.ingredients = [u"puerros", u"patatas",u"sal",u"aceite"]

print porrusalda, porrusalda.denomination, porrusalda.ingredients
```

Factorías (I)

- `.../ejemplo06/recipe.py`

```
from zope.component.interfaces import IFactory
from zope.interface import implementedBy
...
class RecipeFactory:
    implements(IFactory)

    def __call__(self, denomination=u'', ingredients=[]):
        recipe = Recipe()
        recipe.denomination = denomination
        recipe.ingredients = ingredients
        return recipe

    def getInterfaces(self):
        return implementedBy(Recipe)
```

```
RecipeFactory = RecipeFactory()
```

Factorías (II)

- .../ejemplo06/configure.zcml

```
<configure xmlns="http://namespaces.zope.org/zope">  
  
...  
<factory  
    component=".recipe.RecipeFactory"  
    id="ejemplo06.Receta"  
    title="Crear una nueva receta"  
    description="Crea una nueva receta a partir de parámetros"  
/>  
  
...  
  
</configure>
```

Factorías (III)

- ejemplo07.py

```
#!/usr/bin/python
from sys import path
path.append('/usr/local/Zope-3.2.0/lib/python/')
path.append('/var/zope3/third/lib/python/')
from zope.app.debug import Debugger
debugger = Debugger(db="var/Data.fs", config_file="etc/site.zcml")

from zope.app.zapi import createObject

porrusalda = createObject("ejemplo06.Receta", context=None,
                           denomination=u"Porrusalda Casera",
                           ingredients = [u"puerros", u"patatas",
                                           u"sal",u"aceite"])

print porrusalda, porrusalda.denomination, porrusalda.ingredients
```

Vistas (I)

- Son componentes responsables de la interacción con el usuario
- Una vista se registra
 - ▶ para ~~una clase~~ un interfaz
 - ▶ para una petición
 - `zope.publisher.interfaces.http`
 - `zope.publisher.interfaces.ftp`

Vistas (II)

- .../ejemplo08/configure.zcml

```
<configure xmlns="http://namespaces.zope.org/zope"  
           xmlns:browser="http://namespaces.zope.org/browser" >
```

```
...
```

```
<browser:addMenuItem  
    title="Receta"  
    class="ejemplo08.recipe.Recipe"  
    permission="zope.ManageContent"  
    view="AddRecipe.html"
```

```
/>
```

```
<!--
```

Con esta directiva, nuestra 'Receta' aparecerá en la lista de tipos que el ZMI muestra en el menu 'crear nuevo objeto'

```
-->
```

```
</configure>
```

Vistas (III)

- .../ejemplo08/configure.zcml

```

<configure xmlns="http://namespaces.zope.org/zope"
           xmlns:browser="http://namespaces.zope.org/browser" >
...
  <browser:addform
    schema="ejemplo08.interfaces.IRecipe"
    content_factory="ejemplo08.recipe.RecipeFactory"
    label="Add Recipe"
    name="AddRecipe.html"
    permission="zope.ManageContent"
  />
  <!--
    Así registramos la otra vista, que es un formulario
    creado automáticamente a partir del esquema 'IRecipe'
  -->
</configure>

```

Vistas (IV)

- .../ejemplo08/configure.zcml

```
<configure xmlns="http://namespaces.zope.org/zope"  
           xmlns:browser="http://namespaces.zope.org/browser" >
```

```
...
```

```
<browser:editform  
    schema="ejemplo08.interfaces.IRecipe"  
    label="Add Recipe"  
    name="edit.html"  
    menu="zmi_views" title="Edit"  
    permission="zope.ManageContent"
```

```
/>
```

```
<!--
```

De nuevo, registramos la vista 'edit.html', que es un formulario creado automáticamente a partir del esquema 'IRecipe'

```
-->
```

```
</configure>
```

Vistas (V)

- .../ejemplo08/configure.zcml

```
<configure xmlns="http://namespaces.zope.org/zope"  
           xmlns:browser="http://namespaces.zope.org/browser" >
```

```
...
```

```
<browser:page  
    for="ejemplo08.interfaces.IRecipe"  
    name="index.html"  
    template="recipeview.pt"  
    permission="zope.View"
```

```
/>
```

```
<!--
```

Ahora, registramos la vista 'index.html'. Se trata de una plantilla 'recipeview.pt' que programaremos mediante TAL y METAL

```
-->
```

```
</configure>
```

Vistas (VIII)

- .../ejemplo08/configure.zcml

```
<configure xmlns="http://namespaces.zope.org/zope"  
           xmlns:browser="http://namespaces.zope.org/browser" >
```

```
...
```

```
<browser:resource name="global.css" file="recipe.css" />
```

```
<browser:resource name="footer.png" file="recipe.png" />
```

```
<browser:icon  
    name="zmi_icon"  
    for="ejemplo08.interfaces.IRecipe"  
    file="recipe_icon.png"  
/>
```

```
</configure>
```

Vistas (VII)

- .../ejemplo08/recipeview.pt

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:tal="http://xml.zope.org/namespaces/tal" >
  <head>
    <title tal:content="context/denomination/title"></title>
    <link rel="stylesheet" type="text/css" href="++resource++global.css" />
  </head>
  <body>
    <h1>Un plato de <span tal:replace="context/denomination/title"></span>
      alimenta mucho.
    </h1>
    <h2>Ingredientes:</h2>
    <ul><li tal:repeat="ingredient context/ingredients"
          tal:content="ingredient"></li></ul>
    
  </body>
</html>
```

Otros Adaptadores (I)

- `.../ejemplo09/size.py`

```
from zope.interface import implements
from zope.app.size.interfaces import ISized
```

```
class RecipeSize(object):
    implements(ISized)
```

```
def __init__(self, context):
    self.context = context
```

```
def sizeForSorting(self):
    """Compute a size for sorting"""
    chars = 0
    chars += len(self.context.denomination)
    chars += sum(map(len, self.context.ingredients))
    return ('byte', chars)
```

...

Otros Adaptadores (II)

- `.../ejemplo09/size.py`

```
from zope.interface import implements
from zope.app.size.interfaces import ISized

class RecipeSize(object):
    implements(ISized)

    ...

    def sizeForDisplay(self):
        """Generate a displayable size report"""
        unit, chars = self.sizeForSorting()
        msg = _('${chars} characters')
        msg.mapping['chars'] = chars
        return msg
```

Otros Adaptadores (III)

- ../ejemplo09/configure.zcml

```
<configure xmlns="http://namespaces.zope.org/zope"
           xmlns:browser="http://namespaces.zope.org/browser" >
...

  <adapter
    for=".interfaces.IRecipe"
    provides="zope.app.size.interfaces.ISized"
    factory=".size.RecipeSize"
  />

</configure>
```

i18n y l10n (I)

- Mensajes e ID de mensajes
 - ▶ vista
 - "view", "view-component"
 - ▶ ver :
 - "view", "view-action"
 - ▶ permiso para ver
 - "view", "view-permission"

i18n y l10n (II)

- Dominios de i18n
 - ▶ componentes en los que vamos a centralizar la tarea de traducción de mensajes.
- Cada tipo de fichero tiene sus propias reglas sintácticas para la i18n:
 - ▶ ficheros zcml, plantillas, scripts, etc.

i18n y l10n (III)

- Extraemos los mensajes del módulo
 - ▶ `bin/i18nexttract -> .pot`
- Los traducimos manualmente
 - ▶ `gtranslator, poEdit, etc. -> .po`
- Los movemos al directorio correspondiente y en el formato adecuado
 - ▶ `msgfmt -> .mo`

i18n y l10n (IV)

- `.../ejemplo10/interfaces.py`

```
from zope.interface import Interface
from zope.schema import TextLine, List
from zope.i18nmessageid import MessageIDFactory
_ = MessageIDFactory('ejemplo10')

class IRecipe(Interface):
    """ Store information about a recipe """
    denomination = TextLine( title = _(u"Denomination"),
                             description = _(u"Name of the dish"),
                             required = True )

    ingredients = List( title = _(u"Ingredients"),
                        description = _(u"List of ingredients."),
                        required = True,
                        value_type = TextLine(title = _(u"Ingredient")) )
```

i18n y l10n (V)

- .../ejemplo10/recipeview.pt

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:tal="http://xml.zope.org/namespaces/tal"
      xmlns:i18n="http://xml.zope.org/namespaces/i18n">
...
  <body i18n:domain="ejemplo10">

    <h1 i18n:translate="" tal:omit-tag="">
      Un plato de
      <span i18n:name="nombre_de_receta"
            tal:replace="context/denomination/title"></span>
      alimenta mucho.
    </h1>
    <h2 i18n:translate="">Ingredientes:</h2>
...
  </html>
```

i18n y l10n (VI)

- .../ejemplo10/configure.zcml

```
<configure xmlns="http://namespaces.zope.org/zope"
           xmlns:i18n="http://namespaces.zope.org/i18n"
           i18n_domain="ejemplo10"
>
...
  <browser:addMenuItem
    title="[label-receta] Receta"
    class="ejemplo10.recipe.Recipe"
    permission="zope.ManageContent"
    view="AddRecipe.html"
  />
...
  <i18n:registerTranslations directory="locales" />
</configure>
```

i18n y l10n (VII)

```
# ./bin/i18nexttract -p lib/python/ejemplo10 -d ejemplo10 -o locales

# mkdir lib/python/ejemplo10/locales/es/
# mkdir lib/python/ejemplo10/locales/es/LC_MESSAGES/
# mkdir lib/python/ejemplo10/locales/eu/
# mkdir lib/python/ejemplo10/locales/eu/LC_MESSAGES/
# cp lib/python/ejemplo10/locales/ejemplo10.pot
    lib/python/ejemplo10/locales/es/LC_MESSAGES/ejemplo10.po
# cp lib/python/ejemplo10/locales/ejemplo10.pot
    lib/python/ejemplo10/locales/eu/LC_MESSAGES/ejemplo10.po

# gtranslator lib/python/ejemplo10/locales/es/LC_MESSAGES/ejemplo10.po
# gtranslator lib/python/ejemplo10/locales/eu/LC_MESSAGES/ejemplo10.po
# msgfmt -o lib/python/ejemplo10/locales/es/LC_MESSAGES/ejemplo10.mo
    lib/python/ejemplo10/locales/es/LC_MESSAGES/ejemplo10.po
# msgfmt -o lib/python/ejemplo10/locales/es/LC_MESSAGES/ejemplo10.mo
    lib/python/ejemplo10/locales/es/LC_MESSAGES/ejemplo10.po
```

Layers y Skins (I)

- Skin: “theme” o “look” de un sitio web
- Es una lista de layers
 - ▶ Que son elementos intercambiables y reutilizables que pueden compartir varias skins.
- Proceso de presentación de una vista:
 - ▶ Determinar skin (request | default)
 - ▶ Determinar lista de layers del skin
 - ▶ Buscar dicha vista en este layer, o en la siguiente, o en la siguiente... o lanzar error.

Layers y skins (II)

- .../ejemplo10/configure.zcml

```
<configure xmlns="http://namespaces.zope.org/zope"
           xmlns:browser="http://namespaces.zope.org/browser" >
...

  <browser:layer name="ejemplo10" />

  <browser:skin name="ejemplo10" layers="ejemplo10 rotterdam default" />

...
</configure>
```

Layers y Skins (III)

- Todo Skin debe proporcionar una vista especial llamada “standard_macros”
 - ▶ Es decir, esta vista tiene que estar disponible para alguna de las layers de las que está compuesto un skin
- En esta vista encontraremos varias macros:
 - ▶ page, una página web normal
 - ▶ view, una página de administración
 - ▶ dialog, un pop-up

Layers y skins (IV)

- `.../ejemplo10/standardmacros.py`

```
from zope.app.rotterdam.standardmacros import StandardMacros as BaseMacros
```

```
class StandardMacros(BaseMacros):  
    macro_pages = ('ejemplo10_plantilla_macros',) + BaseMacros.macro_pages
```

```
## Creamos una nueva vista que sustituye a standard_macros
```

```
## Va ser una copia exacta de StandardMacros (del skin rotterdam)
```

```
## Más las macros que definamos en la plantilla ejemplo10_plantilla_macros
```

Layers y skins (V)

- .../ejemplo10/configure.zcml

```
<configure xmlns="http://namespaces.zope.org/zope" > ...
  <browser:page
    for="*"
    name="standard_macros"
    permission="zope.View"
    class=".standardmacros.StandardMacros"
    layer="ejemplo10"
    allowed_interface="zope.interface.common.mapping.IItemMapping" />
  <browser:page
    for="*"
    name="ejemplo10_plantilla_macros"
    permission="zope.View"
    layer="ejemplo10"
    template="recipes_macros.pt" />
  ...
</configure>
```

Layers y skins (VI)

- .../ejemplo10/recipe_macros.pt

```
<metal:macro define-macro="page">
  <metal:macro use-macro="context/@@skin_macros/page">
    <!-- ... /lib/python/zope/app/rotterdam/template_tablelayout.pt -->

    <metal:slot fill-slot="style_slot">
      <link rel="stylesheet" type="text/css"
            tal:attributes="href context/++resource++global.css" />
      <metal:slot define-slot="style_slot" />
    </metal:slot>

    <metal:slot fill-slot="body">
      <metal:slot define-slot="body" />
    </metal:slot>

  </metal:macro>
</metal:macro>
```

Layers y skins (VII)

- .../ejemplo10/recipeview.pt

```

<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:tal="http://xml.zope.org/namespaces/tal"
      xmlns:metal="http://xml.zope.org/namespaces/metal"
      metal:use-macro="context/@@standard_macros/page"
      i18n:domain="ejemplo10">
  <head>
    <title metal:fill-slot="title"
          tal:content="context/denomination/title"></title>
  </head>
  <body>
    <div metal:fill-slot="body">
      ...
    </div>
  </body>
</html>

```

Layers y skins (VIII)

- etc/overrides.zcml

```
<configure xmlns="http://namespaces.zope.org/zope"  
           xmlns:browser="http://namespaces.zope.org/browser">
```

```
  <browser:defaultSkin name="ejemplo10" />
```

```
</configure>
```

```
<!--
```

Mediante esta directiva, el skin que se recupera por defecto es el creado en el ejemplo 10

```
http://localhost:8080/++skin++Ejemplo10/porrusalda/@@index.html  
http://localhost:8080/porrusalda/@@index.html
```

```
-->
```

Metadatos (I)

- Existe un interfaz IAttributeAnnotable que permite describir una clase mediante metadatos (ej. DublinCore)

```
<configure xmlns="http://namespaces.zope.org/zope" >
...
  <content class=".recipe.Recipe" >
    ...
    <implements
      interface="zope.app.annotation.interfaces.IAttributeAnnotatable"
    />
    ...
  </content>
...
</configure>
```

Metadatos (II)

- ../ejemplo11/recipeview.pt

...

```

<div tal:define="creada context/zope:created;
             modificada context/zope:modified;
             formatter python:request.locale.dates.getFormatter('dateTime')"
>
  <div i18n:translate="fecha-receta">
    La receta ha sido creada
    <span i18n:name="created_date"
          tal:replace="python:formatter.format(creada)" />
    y modificada
    <span i18n:name="modified_date"
          tal:replace="python:formatter.format(modificada)" />
  </div>
</div>

```

...

Metadatos (III)

- `.../ejemplo12/recipe.py`

```
import time, xmlrpclib
from zope.app.publisher.xmlrpc import XMLRPCView
from zope.app.dublincore.interfaces import IZopeDublinCore

class RecipeView(XMLRPCView):

    def dublincore_info(self):
        dc = IZopeDublinCore(self.context)
        items = [(field, getattr(dc, field))
                 for field in getFields(IZopeDublinCore)]
        info = dict(items)
        for name in ('effective', 'created', 'expires', 'modified'):
            if info[name]: epochtime = time.mktime(info[name].timetuple())
                           info[name] = xmlrpclib.DateTime(epochtime)
            else:          info[name] = ''
        return info
```

Metadatos (IV)

- .../ejemplo12/configure.zcml

```
<configure xmlns="http://namespaces.zope.org/zope"  
           xmlns:xmlrpc="http://namespaces.zope.org/xmlrpc" >
```

...

```
<xmlrpc:view  
    for="ejemplo12.interfaces.IRecipe"  
    class=".recipe.RecipeView"  
    methods="dublincore_info"  
    permission="zope.View"  
/>
```

...

```
</configure>
```

Metadatos (V)

- ejemplo13.py

```
#!/usr/bin/python
```

```
import xmlrpclib
```

```
server_url = "http://localhost:8080/porrusalda"
```

```
server = xmlrpclib.Server(server_url)
```

```
resultado = server.dublincore_info()
```

```
for dato in resultado:
```

```
    print dato, resultado[dato]
```

En el tintero...

- DocTests
 - ▶ Documentación ejecutable
- Security
 - ▶ Principals, Permissions, etc.
- Events
 - ▶ handle, notify,
- Sources (vocabularies)
 - ▶ Restricción del dominio en schemas

Referencias (I)

- [1] `"http://www.zope.org/Wikis/DevSite/Projects/ComponentArchitecture/ProgrammerTutorial"`
- [2] `"http://griddlenoise.blogspot.com/"`
- [3] `"Web Component Development with Zope3"`. Philipp von Weitershausen. Springer. 2005
- [4] `"http://www.zope.org/Wikis/DevSite/Projects/ComponentArchitecture/Zope3Book/schema.html"`
- [5] `"http://zissue.berlios.de/z3/Zope3In30Minutes.html"`
- [6] `"http://gtranslator.sourceforge.net"`
- [7] `"http://www.gnu.org/software/gettext/"`