Developing Web Applications with Zope X3

FORMULA X

Zope, a web application server written in Python, is an extremely popular open source content management system platform. The newly developed version X3.0 was released just recently. We'll show you what's new in Zope X3.0.

BY PHILIPP VON WEITERSHAUSEN

Since the release of the Plone [1] content management system (CMS), it has been hard to imagine a world of free web application servers without Zope [2]. Python, the language Zope is primarily written in, brings object-oriented and modular source code to the world of Zope, along with the kind of agility and flexibility more typically associated with web scripting languages.

From the CMF to New Developments

Zope is a popular platform for CMS. Most free Zope CMS systems (such as Plone [1], Silva [3] and CPS [4]) use a Zope product known as the Content Management Framework (CMF). The CMF framework for Zope 2 not only had the tools necessary for developing a content management system, it also provided a component-based architecture that supported flexible modification of individual components. The experience gained developing with the CMF was then supposed to be incorporated into the Zope development process.

Instead of reworking the labyrinthine Zope source code, the Zope Corporation opted for a clean sheet three years ago. Because of its firm commitment to the principle of free software, Zope has always had the support of a strong community. Experienced Zope programmers, and newcomers, organized major and minor developer sessions (so-called "sprints") all over the world to keep the project on the move. Methods such as extreme programming (XP) and unit testing brought quality assurance to the project. Also, the fact that Zope version 2 is a stable and successful product gave developers enough time to optimize key features in multiple iteration steps. And nobody complained if code had to be ditched to make way for better implementations.

Listing 1: Interfaces (interfaces.py)

```
from zope.interface import Interface
from zope.schema import Text, TextLine
class IBuddy(Interface):
  """Information about a buddy"""

  first = TextLine(title=u"First name")
  last = TextLine(title=u"Last name")
  address = Text(title=u"Address")
  postal_code = TextLine(title=u"Postal
code")
```

Features

The core of Zope X3 is its component architecture. Dif-

ferent components are assigned responsibility for specific tasks. For instance, components manage tasks such as data storage, data processing, and presentation.

Many readers will be familiar with the ZODB (Zope Object Database) from Zope 2. The ZODB allows objects to be persisted fairly transparently in the database, and

Listing 2: Content Components (buddy.py)

```
from persistent import Persistent
from zope.interface import implements
from buddydemo.interfaces import IBuddy

class Buddy(Persistent):
    implements(IBuddy)

    def __init__(self, first='', last='',
                 address='', postal_code=''):
        self.first = first
        self.last = last
        self.address = address
        self.postal_code = postal_code
```

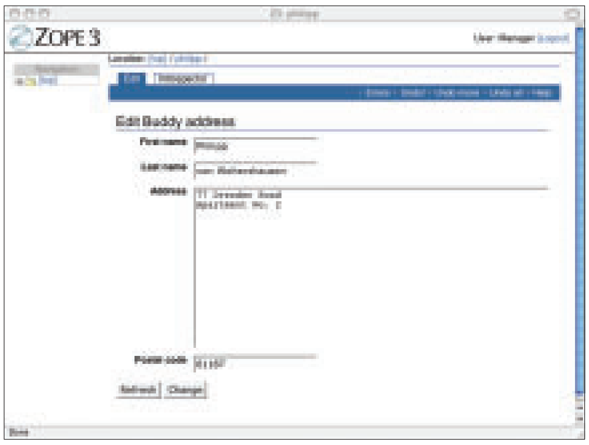

Figure 1: This editing form was generated automatically by reference to the data schema for the content component.

at the same time provides enterprise-level features such as transactions, revisions, and pluggable storage back-ends. ZEO (Zope Enterprise Objects) even support clusters of multiple Zope instances, thus providing ease of scalability.

A flexible security system allows the user to assign permissions to protect components, properties, and methods. Users wanting to access protected components need to be authorized. Modular components authenticate and authorize users, giving operators the ability to adapt the security to the situation without reworking the underlying application.

Zope not only has the localization tools developers need to provide international support for applications, Zope is itself fully international. A full range of features, such as SMTP or sendmail-based email services, an event notification system, and support for XML RPC, round off the Zope palette.

Paradigm Change

Zope X3 removes some of the issues that affected Zope 2. For example, the earlier version expected class instances to provide the attributes and methods they needed to interact with Zope. This led to developers overloading objects with a variety of methods, limiting portability and making subsequent changes to the functions difficult.

Zope X3, on the other hand, keeps individual components as simple as possible and adds components if it needs to add functionality. Zope's architecture defines the following component types:

- *Content* components do not typically have methods, but just properties which they use to publish stored data. Data types and values are typically specified using a data schema.
- *Utilities* are context-independent components that perform a specific task, such as database connections, indexing, or email delivery.
- *Adapters* are probably the most powerful components. Adapters allow developers to add functionality to existing components without having to modify the components themselves. This technique is extremely useful if a framework requires a specific API. Developers can leave the original component as is and implement an adapter that interfaces between the component and the required API.
- *Views* visualize the other components for the user. An example of this is a web browser, which uses views that render HTML. A view is actually a special kind of adapter that gives the other objects a feature they do not possess natively (presentation).

Abstract Contract

To allow components to remain independent of the implementation, the components are not referenced by class. Instead, interfaces are used to describe the functionality a component provides. Interfaces are a kind of formal contract that guarantees the provision of a specific function in the form of an API. Because the Python language does not use interfaces, Zope had to implement them from scratch.

Listing 1 shows an interface from a sample application that manages private

Xs and Us

When development work on Zope 3 started, it quickly became clear that the project would need to drop API compatibility to Zope 2. The "X" version prefix was originally intended to indicate the "experimental" nature of the project. Of course, the stable Zope X3.0 version is anything but experimental, but the "X" remains in the name to warn users that Zope X3 is not compatible with the previous version.

Listing 3: Configuration (configure.zcml)

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

<content class="buddydemo.buddy.Buddy">
   <require
       permission="zope.View"
       interface="buddydemo.interfaces.IBuddy" />
   <require
       permission="zope.ManageContent"
       set_schema="buddydemo.interfaces.IBuddy" />
</content>

<browser:addform
    schema="buddydemo.interfaces.IBuddy"
    label="Add a new buddy address"
    content_factory="buddydemo.buddy.Buddy"
    arguments="first last address zipcode"
    name="AddBuddy.html"
    permission="zope.ManageContent" />

<browser:editform
    schema="buddydemo.interfaces.IBuddy"
    label="Edit buddy address"
    name="edit.html"
    menu="zmi_views" title="Edit"
    permission="zope.ManageContent" />

<browser:addMenuItem
    class="buddydemo.buddy.Buddy"
    title="Buddy"
    permission="zope.ManageContent"
    view="AddBuddy.html" />

</configure>
```

addresses. The interface is a data schema that describes content components – the address data for a buddy. Storing the postal code allows you to search for city and state information later.

In Listing 1, the Python *class* statement is used to define an interface, as Python does not have interfaces natively. Additionally, Zope does not distinguish between an interface that uses methods to describe functionality and an interface that defines a data schema.

Simple Content Components

The task of writing a persistent class in Zope 2 was quite complex. At a minimum, you needed the metatype, security declarations, and instantiating methods to generate the new instances required by the web interface. In Zope 3, you'll be happy to hear, the requirements for writing a persistent class are much easier than in Zope 2. Persistent objects only need to handle the data passed to them; everything else is handled by other components.

Listing 2 shows a working implementation of the *IBuddy* interface shown in Listing 1. Note that the class inherits from *Persistent* to ensure that its instances are automatically stored in the ZODB. To ensure the class against other components, you also need to specify that the class implements the *IBuddy* interface.

XML-based Configuration

Zope 2 expected to import libraries from the *Products* directory. The initialization module for each package (*__init__.py*) used to contain the component registration. Other elements such as security declarations or browser view configurations ended up in the application code.

Zope X3 takes a different approach. Zope extensions are now just simple Python packages, and you can install them wherever you like – providing they are in the *PYTHONPATH*. Everything else that has to do with component configuration, such as the registration itself, security declarations, or browser views, is now wired to the configuration file. This approach gives developers an important advantage: the ability to disable components temporarily or permanently without modifying the code.

Listing 3 shows the typical configuration directives for a schema-based content component; the example only handles the security declarations for reading and writing data for buddy instances. The file then goes on to define two forms. One form generates buddy objects; the other edits buddy objects.

Zope has the ability to automatically generate a form from the data schema defined in the *IBuddy* interface. Thus – as Figure 1 shows – a schema, a simple persistent implementation, a few configuration directives, and no HTML at all give you a component that will run in a browser.

The last directive in Listing 3 adds an entry to the Zope web interface menu to allow users to create buddies. The fact that this directive exists at all exemplifies an important basic principle of the Zope 3 philosophy: “Explicit is better than implicit.” Although developing Zope 3 software may mean more typing, at least you will have an easier time reading the code in six months' time.

Future

Zope has had a massive following for quite a while now thanks to its features, its flexibility, and its reliance on Python; and it looks like this community will continue to grow. Use of the software for large-scale projects and in large enterprises has helped Zope mature. Zope X3 is a massive step forward for the project. This said, it may take some time for Zope X3 and the new paradigm to spread, so we can expect Zope 2 to be around for quite a while yet. Thanks to the Five project [6], which supports gradual and considered migration to Zope X3 despite its API incompatibilities with Zope 2. ■

Installation and Configuration

Installing Zope on Linux is simple. You will need the current Python Version 2.3.4 with Zlib support. Zope may be written in Python for the most part, but some modules were implemented in C for reasons of speed. You will need to build the application before you can install. The tar.gz archive has a *configure* script that will automatically generate a *Makefile* for the build and install process.

The Zope libraries are normally installed in */usr/local/Zope-3.0.x*. Of course, you can change this location with the *--prefix* parameter in the *configure* script. To launch an instance of the server, you first need to create a directory tree for the instance. This directory tree is not only used for storing the object database for the Zope instance (ZODB), it is also used for storing ancillary libraries specifically required by the instance. Of course, a Zope installation can use multiple parallel instances.

The *mkzopeinstance* script in the *bin* directory creates an instance directory. After specifying the path for the instance and the credentials for a temporary administrative account, you can launch the instance by typing *runzope* (in the *bin* subdirectory for the instance). The configuration for the server, the HTTP and FTP server ports, and various logging options are located in *etc/zope.conf*. The configuration file uses the same format as the Apache server. By default, the HTTP server instance will use port 8080 and the FTP server will use port 8021.

After launching Zope by typing *runzope*, you can press Ctrl-C to quit. As this is impractical for a server application, you can also run the *zopectl* script (also in the *bin* directory of the instance) to control the server; this is similar to *apachectl* for the Apache server.

INFO

[1] Plone: *http://plone.org*

[2] Zope community website: *http://zope.org*

[3] Silva: *http://infrae.com/products/silva*

[4] CPS: *http://www.nuxeo.org/cps*

[5] Zope X3 download: *http://zope.org/Products/ZopeX3*

[6] Five: *http://codespeak.net/z3/five*

THE AUTHOR

Philipp von Weitershausen studies physics in Dresden, Germany. He is also a freelance software developer and a consultant. Phillipp also serves as a member of the Zope 3 development team and is the author of the book *Web Component Development with Zope 3*.