Web Component Development with Zope 3

Philipp von Weitershausen

Web Component Development with Zope 3

With a Foreword by Phillip J. Eby

Second Revised and Enlarged Edition

With 48 Figures and 10 Tables

Springer

Philipp von Weitershausen
Bernhardstr. 66
01187 Dresden
Germany
philipp@weitershausen.de

Library of Congress Control Number: 2006935572

ISBN-10 3-540-33807-1 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-33807-9 Springer Berlin Heidelberg New York
ISBN-10 3-540-22359-2 1. Edition Springer Berlin Heidelberg New York

Typesetting: PTP, Berlin
Production: LE-TEX Jelonek, Schmidt & Vöckler GbR, Leipzig
Cover: KünkelLopka, Heidelberg

Printed on acid-free paper 33/3100/YL - 5 4 3 2 1 0

Für Mami und Papi.

A common gateway linked code to the net,
Its limitations plain for some to see.
While on a plane an engineer set
This common gateway object-orientedly.

Created through a clown's principia,
All built upon acquisitive ideals:
Dark reservoirs of content will see a
Great framework that an acqueduct reveals.

Alas, ideals of acquisition fall
Beneath the weight of complex modes of use.
Who listens to the new religion's call?
How many will the purer creed seduce?

Foundations that we took such pains to mold
Support a new world better than the old.

Steve Alexander, inspired by Aroldo Souza-Leite's
Sonnets from Pythia

Subclassing made Zope and TR
much harder to work with by far.
    So before you inherit,
    be sure to declare it
Adapter, not PyObject*

Glyph Lefkowitz in the docstring of
`twisted.python.components.Adapter`

Foreword

Where Zope leads, Python follows.

So it has been for a decade, and the trend doesn't show any signs of stopping. Whatever the latest buzzword—be it RESTful web programming, standardized interfaces, pluggable components, or practical restricted-execution environments, Zope has quietly led the way, delivering the goods years ahead of anyone else. Not just as technology concepts, but shipped and working in *paying* clients' offices.

And yet, strangely, Zope's role in the ongoing development of Python is little-known and little-appreciated among Python developers. It is frequently the case that some new and much-touted development in the Python community—especially in the web application and object security arenas—is something that Zope has already been doing for many years.

I'm somewhat baffled by this peculiar blind spot in the Python community. Even when I tell people that Zope's already done something that they're working on, the response is usually a blank look, or no response at all. It's almost as if the innovations of Zope don't really exist until somebody else reinvents them. In fact, the pattern has led me coin this little saying:

Those who do not study Zope, are condemned to reinvent it.

It doesn't matter if you don't plan to actually use Zope. Frankly, I haven't used Zope in years. But the lessons I learned from Zope, I use constantly. Studying Zope—Zope 3 in particular—will make you a better programmer, without question.

Of course, "better programmer" begs the question: better how? Better at achieving what? Zope was created so that Zope Corporation (originally Digital Creations) could do contracting business more efficiently. It allows them to keep an ever-growing toolkit of reusable solutions for their clients, reducing the costs of development and maintenance of these applications. Its purpose is to let you "write once, use many": a multiplier of *economic* effectiveness.

If this is the path your career is taking, you can only benefit from studying how this has been achieved in Zope, whether you actually use Zope for this purpose or not.

If you are developing any new or cutting-edge technology for Python, you can only benefit by asking, "Does Zope already have this, or something like it? And if not, how would Zope use this?" These are the questions I asked when developing Python Eggs, setuptools, and even the WSGI (Python Web Server Gateway Interface) specification. The success of these projects is a *direct* reflection of me asking WWZD: What Would Zope Do (with this idea)?

That's because what's good for Zope, is usually good for Python. Not in the language sense—Python's "Benevolent Dictator" and the "Zope Pope" often disagree quite strenuously on how the language should change. What I mean is, tools that make Zope a better platform, make Python a better platform. If you study Zope diligently, you may begin to understand why.

And maybe, just maybe, you'll find yourself a little bit ahead of other programmers, especially when it comes to new ideas... but hopefully, not *so* far ahead, that they begin to act as though *you* don't exist, either!

Good luck!

Phillip J. Eby

Foreword to the First Edition

Zope 3 is here! Congratulations to Jim Fulton, the Zope Pope, and the global team that made it happen. Now the rest of us can start putting its power to use. Truly, with the arrival of the component architecture, Zope can now change its tagline from "Brilliant but clunky" to just "Brilliant".

The world of Zope and its offspring such as Plone have now matured to the point of mainstream legitimacy. In fact, one major European telecom recently told me, "If you bid on a government CMS project, and you don't use Zope, you have to explain why." Incredible!

The world of Zope is big. There are hundreds of add-on packages that integrate into the system in ways that other architectures simply don't anticipate. However, the full potential of Zope's broad basis for business has been held back by limitations that we didn't expect when Zope 2 was designed.

Enter the component architecture! With these lessons from Zope 2, and particularly from the trail blazed by the CMF, software packages can be plugged in and replaced on a more robust and industrial-strength basis. This will help raise the level of basic capabilities common to all Zope-based software, thus accelerating a trend towards enterprise-class services such as versioning.

Most exciting, though, is the business integration. Small companies are specializing in certain platform services and working together for full solutions. The component architecture is crucial for making this work to level of quality demanded by consulting deployments. Also, the often-overlooked upside of Zope 3's test-oriented culture gives us all a boost into better ways of working.

Just like Zope 3 will prove indispensible to the business of Zope, so too will this book. Zope 3 is a commercial-class platform for application services. This book is an in-depth guide to Zope 3, written by one of the most important developers of Zope 3 itself. The overview of core concepts in Chapter 2 is, alone, worth the price of the book. And because the writing style is thoroughly

instructive, while also entertaining, the book makes getting started with Zope 3 a pleasure.

This book appears only shortly after a first version of Zope X3 has been released. This is a monumental accomplishment, serving as a testimony to several factors: the discipline of the author, the dedication of the team of developers that managed the last phase of the project in a professional way, and the now-provable wisdom of Jim Fulton's design decisions. Compared to the early days of Zope 1 and 2, the upside of having this book to accompany Zope 3's launch cannot be overstated. It's a big deal.

I've been lucky to know Philipp for a few years, even working with him on some consulting projects. Not only is he great fun to be around, he is also a real leader in the world of Zope. He has the credibility, the reliability, and the right touch for working with the far-flung army of volunteers that are producing Zope. And atop all that, he has written for us the guide to developing killer apps for Zope 3.

I hope you enjoy this book as much as I have, and good luck in your start on Zope 3.

Paul Everitt, co-founder of Zope Corporation

Contents

Part III Expert

Part IV Appendices

Part I

Beginner

1

Introduction

1.1 About this book

Is this book for me?

This book is for web developers. If you are migrating from Zope 2 or have experience in competing technologies, such as J2EE or Vignette StoryServer, or Python-based frameworks like Django and TurboGears—this book is for you! It is probably not for you if you are looking for an introduction to web application development.

This book introduces the web developer step-by-step to the world of Zope 3 and its Component Architecture. It describes how to build web applications with Zope 3 step by step using detailed examples. It is not a "cookbook" with recipes for specific tasks or solutions to problems.

Examples in this book

To demonstrate Zope's wide range of features, we will use a consistent example application throughout the whole book. Whenever a new feature of Zope 3 is covered, this example application will be extended accordingly. It is supposed to drive the fictitious *World Cookery* website which allows hobby cooks from around the globe to share their recipes online. This particular application was chosen because it incorporates the most important characteristics of the majority of Zope-based web applications:

- A limited set of content object types. In the World Cookery example application, the primary type of content object is a *recipe*.
- Content is added in a management interface accessible through-the-web using a web browser.
- The layout of the application follows a common theme, for example a *corporate identity*.
- The application has to cope with multiple or even numerous users with different roles and responsibilities.

- The target audience is international, thus internationalization is required.
- Existing features must easily be extensible and new functionality easily be addable.

Apart from source code examples, there are hands-on sessions with the Python interpreter. In these sessions we quickly try out things or test components from the example listings.

This book also features...

- *Summaries* at the end of each section allow you to review a chapter and also serve as good reminders in case you need to go back and restudy a certain section.
- *Flashback* boxes explain how a Zope 3 feature compares to its Zope 2 or CMF equivalent.
- Explanations entitled *Using Zope 2* indicate that a Zope 3 feature is available in Zope 2 (see *Zope 2 and backwards-compatibility* below).
- Rocky Burt, a J2EE developer as well as Zope and Plone expert, compares some of Zope's features with those of the J2EE world in *Rocky says...* sections.

1.2 What is Zope?

The rest of this chapter and the next chapter introduce Zope. They give an overview of how Zope works and what Zope can do. If you are eager to get started, you can skip them and go on to install Zope in Chapter 3 or dive directly into the code with Chapter 4.

Zope is

- *a collection of free software*
- *jointly developed by Zope Corporation and a large community of software developers*
- *that you can use in whole or in part*
- *to manage complexity in gluing software components together,*
- *securely publish objects on the web and other systems,*
- *and make it easy to do Quality Assurance.*

Steve Alexander at EuroPython 2004

a collection of free software: Zope is not just a web application server product, it is also a large collection of mostly web-related Python software. It is also freely distributable under the *Zope Public License* [29] and maybe modified and improved by anyone. This increasingly successful concept is called *Open Source* [10].

jointly developed by Zope Corporation and a large community of software developers: When Zope Corporation opened Zope 2's source code to the public, the development process was also opened to encourage contributions from other developers. Zope 3 is a pure community effort from the beginning, with the majority of the contributions coming from a community of software developers around the globe. The benefits for the single developer, the whole community as well as the quality of the software have proven to be enormous.

that you can use in whole or in part: Zope the web application server product is modular and its parts can be used independently from Zope. That is why we need to differentiate between *Zope-the-product* which is the web application server and *Zope-the-project* which produces a collection of reusable software components written in Python.

to manage complexity in gluing software components together: There is a demand for web-based applications to be more and more complex. This can be a time consuming challenge to many applications. Zope solves this problem elegantly by separating activities into many different components. The way these components are then "glued together" determines the behaviour of the overall application.

Secondly, complex applications require a lot of planning and resources. Zope 3's Component Architecture allows the responsibilities for components to be divided across a team. Zope is an excellent platform for collaborative development.

Thirdly, refactoring is easy because only the components needing refactoring are worked on. One example is the customization of presentation and layout components, work can be carried out on HTML pages and CSS style sheets, while the underlying software components are not touched.

securely publish objects on the web and other systems: Zope's main focus is to allow people to manage objects that are published on the web as part of a larger application, not just a static website.

Zope values security very highly. Compliance to international IT security norms is a major part of the Zope 3 philosophy. Zope 3 is undergoing a development-concurrent examination process run by an official IT security agency in Germany. As a result, the Common Criteria Certificate (ISO-15854), a security certificate accepted in Europe, North America and many other countries, will be issued to Zope. An invaluable by-product of this will also be that Zope's security model in its full extent is being documented in an ISO-compliant manner.

and make it easy to do Quality Assurance: The developers of Zope 3 have enforced a high level of quality during development. Zope is tested by several thousand automated tests; any modifications need to assure that

the tests still pass, any new feature needs to be covered by new tests. This ensures the quality of Zope and makes it a great platform to do quality assurance with.

Zope runs on all major Unix platforms, including Mac OS X, as well as Microsoft Windows operating systems. It comes with its own webserver but can interoperate with an existing webserver such as Apache.

Content Management

As an object-oriented web application server with features such as object persistency, Zope provides an excellent platform for building content management systems (CMS). Additional open source party libraries jointly developed by content management vendors allow a custom CMS to be built on Zope. Zope itself is *not* a content management system. However its whole machinery is geared towards managing content. This is why the main focus of this book is the construction of applications that manage content.

"Take Five:" Zope 2 and backwards-compatibility

Zope 3 started out as a complete rewrite of the Zope code base and is not backwards compatible with its predecessor, Zope 2. While many of the concepts had been borrowed from Zope 2 and the Content Management Framework (CMF), Zope 2 and Zope 3 initially shared only very little actual code. That started to change with the formation of the Five project[1] which aims to bring Zope 3 technology into existing Zope 2 applications. Five was integrated into the Zope 2 source code with version 2.8. This has started an evolution process that has led and will continue to lead to an increasingly larger shared code base. This is not an instant process, of course. Vendors of existing Zope 2-based technologies have to be given adequate time to adjust their software.

This book's focus is on Zope 3. There is no explicit chapter covering Zope 2 and Five, although the *Flashbacks* throughout the book give helpful comparisons for Zope 2 developers. Because more and more Zope 3 technology can be used in Zope 2, many solutions presented in this book also apply to Zope 2 and Five. Whenever this is the case, a *Using Zope 2* box will mention it. Apart from that, the concepts and lessons taught in this book can also be applied (even if a bit differently) in the Zope 2 world. It is good to develop Zope 2-based software with Zope 3's way of doing things in mind to be fit for the future.

[1] Five project website <http://codespeak.net/z3/five>

1.3 Zope's features

It is Zope's goal to make the web application developer's life easier. Therefore Zope provides as much of the "application" as possible. The developer only has to link the components together and customize and extend Zope's features as needed. The following is an overview of Zope's major features:

Component Architecture. Zope 3 is a collection of software components. Components are objects with a clear understanding of their functionality and responsibility. Describing this in a formal way is the role of interfaces, a concept introduced by the Component Architecture. With the Component Architecture it is then possible to group components together to form a greater piece of software—the application. All of this is not very web-specific, the Component Architecture (`zope.component`) and the interface implementation (`zope.interface`) may be used outside of Zope. Chapter 2 gives a thorough introduction to the Component Architecture and Chapter 4 introduces Zope interfaces.

Object database. One of the strengths of Zope is the Zope Object Database (ZODB) which stores data as objects. Since Zope is all about object publishing, it makes a sense to store the data as objects, too. Developers can persist Python objects nearly transparently into a transactional database that can have load-balancing capabilities. The ZODB is not bound to Zope and can also be used in other Python applications. Chapter 6 covers the ZODB in detail.

HTML/XML templating. For decent web development, it is important to have an easy and productive way of producing HTML and other XML markup. Zope uses Zope Page Templates (ZPT), an XML templating system. This system retains the template as a well-formed XML document and, apart from a few namespaced tags and elements, also as a valid document as far as the original XML schema is concerned. The implementation (mainly in `zope.pagetemplate`) can also be used in other Python projects. Chapter 7 gives a full introduction to ZPT.

Form generation and validation. Many web applications need to process data that is entered through browser forms. Since the construction of a form in HTML and the validation of the user's input data is not only tedious but quite repetitive, Zope's form library (`zope.formlib`) can do both for you according to your content's data schema (`zope.schema`). This is covered in Chapter 8.

Internationalization. Zope software is used all over the world and needs to support different languages and locale-specific date and number format-

ting. Zope's internationalization (i18n) and localization (l10n) machinery in `zope.i18n` greatly aids the developer to create multilingual and locale-dependent applications. These aids are not tied to Zope. Internationalization is the covered in chapter, Chapter 9.

Security Zope allows for very fine-grained security checks. The publishing and storage mechanism is object-centric and so is the security system. Both the security policy and the authentication system are pluggable and allow the modelling of a flexible security system. Chapter 21 and Chapter 22 tell you how to do that.

Cataloguing. Storing content is one thing, finding and retrieving content is another. Zope has a cataloguing and indexing machinery that can make data stored in Zope easily searchable. This is demonstrated in Chapter 19.

Testing. Zope 3 puts a strong emphasis on testing and software maturity. Its own source code contains over 5000 automated tests that ensure its stability even when big changes to the code base are introduced. These include low-level unit tests as well as integration tests and functional, HTTP-based tests. Zope has also taken taken part in the propagation and improvement of innovative ways of testing such as doctests. Doctests provide developer documentation in forms of automated tests. Most of the testing machinery is available through the very generic `zope.testing` package which also employs a very capable test runner. All of Chapter 12 is devoted to automated tests in Zope.

Playing well with others. Zope has inspired Python technologies such as new-style classes, the WSGI specification, doctests and the `datetime` module. It also reuses code from other Python projects such as Twisted, pytz, docutils and wwwsearch. Zope has in return contributed code and concepts back to these projects. Because of its modularity, many of the above mentioned features can be and are used outside of Zope.

Superseded and emerging features

Zope is under constant development by a world-wide community of developers. This means that features in use now may be superseded in later versions. Why are they documented here? Because Zope 3.3, the Zope 3 version that this book is based on, can be used *now*. Unlike the development code, Zope 3.3 is being used in production successfully. Constant improvement is a good thing. It shows that Zope development is not standing still. Having stable versions to work with is important. The Zope 3 development cycle guarantees both.

Here are some emerging features that have not yet gained the maturity needed to be fully covered in this book:

Workflow Content management systems often need a workflow where people or the system have to do certain actions in a certain order. With the `zope.wfmc` package Zope already features a workflow engine that follows the standard of the Worfklow Management Coalition (WfMC). Apart from that engine, however, there is not a lot of machinery yet for making custom applications easily workflow-aware.

Relational database access. Zope 3 has rudimentary support for relational databases through Python's database connectors. In Zope these are enhanced with a bit of glue code that makes them play well within Zope. There they are called *database adapters*. This only provides a connection to a relational database, though. It cannot be considered an adequate approach for Zope which is about publishing objects, not rows from a relational database. Projects like SQLObject[2] and sqlos[3] are aiming to solve this.

XML. Zope 3 uses XML in quite a few places (Page Templates, ZCML, etc.), however it does not yet have a framework dedicated to XML processing. There have been initial prototypes that were developed in external projects and there are plans to ship an XML library like lxml[4] with Zope, thus allowing Zope to depend on XML features on a broader basis. These plans are not yet definite enough to make them part of this book.

If you think that a vital feature is missing from Zope, do not hesitate to participate in development. Everyone can contribute and the community welcomes even the smallest contribution.

1.4 The history of Zope

Although the Zope community was small when I joined, it already had its myths. "Jim wrote the first version of Zope on a plane" the Zope birthing myth said. Fact is that Jim Fulton was not happy with web development in Python when he returned from the International Python Conference in 1998. He had just given a tutorial on CGI programming at the conference, certainly not the nicest way of doing web development. Back at Digital Creations, he started writing Bobo, an object database and object request broker capable of publishing Python objects over the web. Later, the commercial Principia,

[2] SQLObject project website <`http://sqlobject.org/`>
[3] sqlos project website <`http://codespeak.net/z3/sqlos`>
[4] lxml project website <`http://codespeak.net/lxml`>

Zope's predecessor, joined its little brother Bobo. Their heritage in Zope 2 even lives on today in method and attribute names of the Zope 2 API such as `bobobase_modification_time` and `isPrincipiaFolderish`.

Open Source success story

In 1998, Hadar Pedhazur from the venture capital firm Verticality Investment Group invested into Digital Creations and convinced them to make their successful web applications open source. Paul Everitt, CEO of Digital Creations at the time, announced the opening of the source code at the Python Conference 1998. That same year, Bobo and Principia became Zope, the *Z Object Publishing Environment*, and was released under the Zope Public License. Zope 2.0 followed the next year and has been the basis for all stable Zope 2 releases.

Digital Creations—later renamed to Zope Corporation—now sells its services as a software consultant implementing solutions using Zope open source software. Many other companies around the world now also base their services on Zope and have developed successful solutions such as Plone, Silva, and CPS. Due to the wide-spread network of solution providers, professional Zope support and consultancy is available around the world. Zope itself is now maintained and further developed by developers from these solution providers. In March of 2006, the Zope Foundation was founded to govern the Zope source code and to provide a platform for the Zope community to drive the Zope project.

Zope 2's success from a technical point of view is based on two revolutionary concepts. Firstly, it publishes data on the web via objects that are able to presents themselves to a web browser. Secondly, its web application development process enables people to configure pre-coded software components through a web browser or, as the Zope world calls it, *through-the-web* (TTW). These processes are carried out through a web interface, the *Zope Management Interface* (ZMI) and are stored in Zope's own object database, the ZODB. While this was a great way for developers to quickly get started with Zope, it also meant that the transition from a quickly sketched web application to a larger, filesystem-based software project that needed version control and the like was not easy to do. Hence the phrase Zope's "Z-shaped learning curve" was coined, which meant that Zope made things very easy in the beginning, then required a lot of work to understand the filesystem-based development model at which point it could be considered easy again.

How Zope 3 was born

The Zope 3 project started in 2001 when Zope Corporation experimented with the *Component Architecture*. The goal was to split up the responsibilities and functionality of current Zope 2 objects because they had grown to be incredibly large with respect to their functionality and code in their classes.

The new approach coupled several smaller objects together rather than having one big bloated one. It also included a move away from through-the-web development. One of the goals of this was to smooth Zope's learning curve. A number of these concepts were prototyped in the Content Management Framework (CMF), a library on top of Zope 2 that is the basis for many Zope-based content management systems, such as Plone and CPS.

Zope 2 had grown to be too complex to incorporate Component Architecture. It was decided to do a complete rewrite, Zope 3, which used Component Architecture as its foundation and at the same time borrowed and learned from the strengths and weaknesses of Zope 2. It was first called Zope 3X (X for *experimental*), then Zope X3, its first stable version was Zope X3 3.0. No longer being experimental, Zope 3 dropped the X for Zope 3.1 and all subsequent releases.

Having two very different versions of Zope was still a problem. Martijn Faassen, an active Zope 2 and Zope 3 developer, and others wanted to use Zope 3 technology in existing Zope 2 software. To solve this the Five project was started. Its goal is to make Zope 3 technology available in Zope 2. Five gained momentum from all over the Zope 2 community and was adopted as the official migration strategy. Zope 2.8 had Five integrated into its source code. Zope 2 development was spurred on and subsequent Zope 2 versions started using more and more Zope 3 technology. Probably the most exciting things we will see in Zope's future will come from this integration project since it allows the tremendous amount of excellent Zope 2 software to work with Zope 3 technology in the future, thus enriching both.

1.5 The Python Programming Language

> *Zope magic is Python magic.*
>
> Jim Fulton at the EuroZope Conference Berlin 2002

Zope profits from Python's supremacy. Even more, Zope would be impossible without Python. Python is a quick, easy-to-learn, Swiss Army knife-like scripting language—but so is Perl. Python also is an industrial strength, highly dynamic, object-oriented, interpreted programming language—but so is Java. What makes Python special then?

Python is easy to learn. It took me one afternoon to go through the Python tutorial [13]. Sure, I bought a few books on Python afterwards, but not for learning the language. I merely wanted to know what it was capable of.

Python is easy to read. Go and open an arbitrary python source code file in Zope 3. Chances are good you will understand most of what is going on without actually having seen Python before. Using indentation for marking blocks certainly is the most aesthetic aspect of the language, but also

one many people have to get used to first. Obviously, it is still possible to write cryptic code in Python, but less likely.

Python is easy to write. I rarely need the library reference when programming. When developing Java, I constantly find myself looking up standard library interfaces, abstract classes, etc. Python is not as wordy as Java, but it is not as cryptic as Perl, either.

Python is easy to develop with. Not having complicated package names and long class names in the standard library, Python can easily be written without an IDE that would ease the development process. In fact, the Zope contributor community is more or less equally split into those who use emacs and those who use vi. Moreover, Python does not require a compiler, obsoleting long compilation sessions.

Python comes with batteries included. The regular Python distribution comes with a rich library of useful modules, ranging from parsers for text formats to email messaging and processing. That means you can get started with Python very quickly without having to collect the basic modules yourself. In case you are looking for third party packages you can use the *Python Cheese Shop* website[5] to find the software package of your desire. There are numerous third party packages available as open source which can easily be used in Zope applications, too.[6]

"Python—it fits your brain" a popular Python t-shirt reads; it sums up my programming experience with Python in one sentence. If you are new to it, you are sure to be converted to Python by the end of this book. In case you are looking for a Python book to accompany your Zope 3 development, the author can strongly recommend the excellent *Dive Into Python* by Mark Pilgrim [4].

1.6 Changes since Zope X3 3.0

Since December 2005, Zope 2 and Zope 3 releases have been synchronized and put on a time-based schedule. This ensures that new features will appear in a stable release at a definite point in time while potentially unstable features will not be able to postpone a major release.

The first edition of this book covered Zope X3 3.0. Three major versions of Zope 3 have been released since then. This book uses Zope 3.3. For those of you who developed Zope applications with the first edition, here is a short overview of the most important changes in Zope 3.1, 3.2 and 3.3:

[5] Python Cheese Shop website <`http://cheeseshop.python.org/`>

[6] In Chapter 13, for example, we will use *ReportLab*, a third party library for PDF generation, to generate PDF documents from Zope content objects.

Restructuring

- Zope's publisher API was rebuilt to the *Python Web Server Gateway Interface* (WSGI) specification [12] and has been given a frontend that acts as a *WSGI application*. Similarly, Zope's own HTTP server (`zope.server`) has been made to act as a *WSGI gateway*. Zope also ships with the Twisted network framework. Its WSGI-capable web server, `twisted.web2`, is the default backend now.
- The Component Architecture has been greatly simplified. The concept of services has been dropped, only two types of components remain: utilities and adapters. While the adapter registry has taken over the responsibility of some services like the presentation service and browser menu service, most former services are now simply available utilities. The utility and adapter registries are now part of the *component registry* (formerly known as site manager), which is the successor to the service registry. Chapter 2 explains this in depth.
- With the abolition of all services including the presentation service, views are now named multi-adapters for the object they represent and request. Resources are named adapters for request. Browser layers and skins simply are interfaces extending `IBrowserRequest`, thus request marker interfaces.
- Menus, formerly registered through the browser menu service, are named utilities providing `IBrowserMenu`. Menu items are named multi-adapters for the context object and the request. Submenu items are also possible now.
- The authentication service has been replaced with the authentication utility. The pluggableauth service has also been replaced by a port of the *Pluggable Auth Service* (PAS) which is now called *Pluggable Auth Utility* (PAU) due to its being a utility. It is located at `zope.app.authentication`. Chapter 22 covers this new system.
- Several packages or modules were moved out of `zope.app` to make them easier to reuse, for example in Zope 2. These are:
  - `zope.app.size`
  - `zope.app.location`
  - `zope.app.annotation`
  - `zope.app.dublincore`
  - `zope.app.copypastemove`
  - `zope.app.mail` (to `zope.sendmail`)
  - `zope.app.rdb`
  - `zope.app.filerepresentation`
  - `zope.app.traversing`
  - `zope.app.event` (to `zope.component.event` and `zope.lifecycleevent`)

- The mutable implementation of i18n message IDs has been replaced with an immutable one, now simply called *i18n messages*. Chapter 9 covers this as well as the new semantics regarding immutability.
- The vocabulary feature is being replaced by *sources*, a simpler and less constraining framework.
- A number of redundant or superfluous ZCML directives have been removed:
  - `factory`
  - `vocabulary`
  - `content` (as an alias of `class`)
  - `modulealias`
  - `browser:addview`
  - `renderer:renderer`

New features

- A new library for form generation and validation, `zope.formlib`, has been added to Zope. Instead of defining forms through ZCML, they are now implemented in Python and thus give the developer much more flexibility over forms while automating the recurring parts of generating and validating forms. Chapter 8 covers the new form system.
- A new machinery for dynamically constructing user interfaces via *content providers* has been added with the `zope.contentprovider` package, along with a content provider implementation called *viewlets* located in `zope.viewlet`. Chapter 10 covers both in detail.
- A reinterpretation of Zope 2's *ZCatalog* for the Component Architecture is now available for Zope 3 at `zope.app.catalog`, along with some indices (`zope.index`) and a unique ID facility (`zope.app.intid`). This makes it possible to efficiently index and search objects in the ZODB. ZCatalog is covered in Chapter 19.
- Zope 3 now has support for browser sessions located in `zope.app.session`. Chapter 20 covers this.
- An engine for the Worfklow Management Coalition (WfMC) workflow standard has been added to Zope with the `zope.wfmc` package.

2

Zope and the Component Architecture

This chapter gives a bird's eye view of Zope as an object publishing environment. It explains how Zope works as well as the principles of its Component Architecture. Make sure you understand the basic concepts explained here before moving on.

2.1 How Zope works—an overview

Zope is a collection of software suitable for building web applications. Zope is typically used as a web application server in which case it consists of three major parts:

Server. The server is the component that lets users connect to Zope by allowing network requests from clients. Typically this means HTTP connections, such as from a web browser, XML-RPC client, or WebDAV client.

The *Python Web Server Gateway Interface* (WSGI) specification [12] calls this the *gateway*. WSGI-capable HTTP gateways that can currently be used with Zope are Zope's own `zope.server` and Twisted's `twisted.web2`.[1]

Publisher. Requests coming in through the server are handled by the publisher. WSGI calls this part the *application*, but in Zope, the application does not deal with such low-level things as parsing the HTTP input stream. This is the publisher's task instead, making it a distinct component between server and application.
The WSGI gateway invokes Zope's publisher via the `zope.app.wsgi.WSGIPublisherApplication` class. This class, among others, con-

[1] `twisted.web2` project website <`http://twistedmatrix.com/projects/web2`>

structs a *request* object from the input stream and hands it off to the actual Zope publishing machinery in `zope.publisher`. This is the machinery which makes Zope unique: object publishing. Zope's object publisher first finds the object addressed by the request and then invokes that object. The former is called *traversing*, the latter *publishing*. In object publishing, the request object and its corresponding *publication* object cooperate. The latter takes care of actions that are related to a particular request, for example starting and committing database transactions.

Application. Basically, the application is everything that the server and the publisher are not. For example, determining *how* the publisher traverses through objects to the one it finally invokes is part of the application, just like other parts such as security.
When we say that we develop Zope applications, we are specifically referring to the application part of Zope. Of course we do not have to write our own security system nor do we have to rethink every detail about object traversal. Zope already has this functionality. It can be used directly, customized, enhanced, or completely replaced for your own application.

Rocky says...

WSGI is about enabling different Python web frameworks to work along side each other. Java developers should see this akin to Java Servlet technology. JSP is an example of a framework built on top of servlets. Even filters as defined by the Servlet specification behave closely to WSGI's middlewares.

2.2 Introducing components

When using large frameworks like Zope, the goal is to avoid starting from scratch. That means Zope application development involves the skilled reuse, customization and extension of the existing functionality. The question is how do this as easily and as flexibly as possible.

Subclassing

Object-orientation provides two approaches to such customization. The first one is subclassing. A subclass derived from one or more superclasses inherits all of their functionality, unless explicitly overridden. That certainly is a very powerful concept, allowing you as the developer to mix as many functions together in one class as you like[2]. That is a very common scheme in Zope

[2] As long as the language supports multiple inheritance, which Python does.

2. Many Zope 2 classes only exist because they are to be "mixed in" into application classes. Some of them even carry a `Mixin` suffix in their name.

Subclassing has a weakness in practice: you have to create a new class for every change in functionality. Often, you find your mix-in classes rely on functionality intended to be implemented by subclasses, which is an error-prone scheme. Moreover, changing even presentation behaviour would mean writing Python code (a new subclass) even if only an HTML template has to be customized.

Delegation

Delegation is the other concept. Instead of inheriting functionality from superclasses, work is delegated among several separate objects called *components*. Each component takes a responsibility in a complex action. For instance, a content component is only responsible for storing data while a presentation component is responsible for presenting it. When a component is no longer satisfactory, whether in general or for your particular application, it can be replaced by a better implementation. For example, a presentation component that determines the way a content component is presented to the web browser could be exchanged without any modifications to the underlying content component. The Component Architecture is the tool by which Zope can provide this flexibility and power.

In order to be exchangeable for another implementation, components need to clearly define what kind of functionality they expect from other components and what kind of functionality they provide in return. In the real world, this is often called a *contract*. Objects must have a clear contract in order to be components. Like other systems, the Component Architecture uses interfaces to express such contracts (see below). We can therefore define components as *objects with interfaces*.

We commonly divide components into three categories: Model, View, and Controller (MVC). The model holds the data. It knows about the data's structure, as well as where and how to store it. The model is analogous to what we often call a *content component*. The view presents the data to the user (e.g. as a GUI element or a web page) and the controller processes data in order to change the model or the view. In the Zope Component Architecture, *adapters* can be used to implement controllers and views. They are a generic way of extending existing components with data processing or presentation capabilities. Apart from adapters, the Component Architecture also knows *utilities*. These are components fulfilling a certain task without the context of another component, for example database look-ups or indexing and searching.

The following sections of this chapter will uncover more details on each of those component types.