Comparing Zope, Django and Rails

Marc W. Mengel

With application frameworks becoming more popular, I evaluated two popular packages, and compared them to decide what to recommend to our Computing Division developers; namely Django (a framework in the Python programming language) Zope (another framework, which significantly extends Python) and Rails (a frame work in the Ruby programming language). I found them to be roughly equivalent in functionality, and thereby recommend Django for our usage because our developer base is much more familiar with Python then Ruby.

Overview

I will first present the feature sets available in these packages, and contrast them where the feature is implemented differently.

1.1 Persistence

Two different persistence models are used in the packages reviewed; Object databases and Relational databases.

In Rails and Django, persistence is provided through object classes that inherit from a Model class, and represent rather directly tables in a relational database, and instances of those objects represent rows in those tables. Saving object changes to the database is explicit, and implemented using the obvious SQL INSERT or UPDATE mechanisms.

In Zope, persistence is provided through object classes that inherit from a Persistent class, and a single object database (ZODB) is used to store the data. A transaction model with the possibility of conflicts and retries is provided.

Our experience to date with the ZODB frameworks is that the performance is less than stellar, and that implementors have to build separate indexes (which are a great source of database conflict errors and retries) to speed up searching for data; whereas the relational database setups can have standard database indexes and query optimizations used to address performance issues with the persistent storage. [While one can access relational databases from within Zope, that is not the usual development/persistence model, and the support is not done in an object-oriented framework.]

Also, the relational database persistence packages (Django and Rails) have tools to reverse-engineer object definitions from existing databases, which makes it much easier to integrate directly with your existing database applications.

Supported Databases

The various packages support different databases:

django
- MySql
- Oracle
- Postgres
- sqlite3

Rails
- DB2
- Informix
- Interbase
- MySql
- Oracle
- Postgres

- sqlite
- DB2
- Sybase

Zope

- DB2
- Informix
- Interbase
- Gadfly
- MySql
- PostgreSQL
- Oracle
- Sybase
- SQLServer

However adding more database back-ends to django is fairly straightforward for databases with python DB API interfaces; one needs to find out how to query the database for the database introspection implementation.

1.2 URL mapping

All of these systems must map URLs to calls in the system. There are several approaches that are used.

django has a regular-expression mapping system which maps URLs to

Python function calls. This is a fast, low-overhead system; but it is by design not defined to what objects a given URL calls.

Zope puts object instances in container objects, and the container hierarchy has a root. This leads to path-based URL mapping where http://site/a/b/c/d maps to the "d" method on the object tied to 'c' in the object tied to 'b' in the object tied to 'a' in the root container object -- that is it works like a file-system.

Rails by default maps everything as http://site/controller/action/id where the controller is the name of the controller class, and the action is a template + method name. However, they've recently added a Routing object class which lets you define URL mappings.

1.3 Template system

All of these platforms provide some sort of templating system for providing a web-page or XML or text view of data. Zope goes so far as to make template rendering an implicit method of persistent objects, whereas the other platforms provide template rendering as a toolkit method which you can use explicitly in a method you write.

Zope comes with 2 template packages, DTML and TAL. DTML is an SGML template language, while TAL is an XML based one. TAL has one advantage above all the other notations here in that it can be loaded and saved cleanly in nearly all HTML editors (i.e. without damaging the active parts of the template), so a web designer using, say, DreamWeaver could edit the template and make it look nice. The disadvantage of both the Zope template notations is the difficulty in editing them, and the implied context in which they operate, which makes getting the right data into the template renderer sometimes difficult.

Ruby comes with 3 template notations, one (ostensibly) for web pages (although it can be used for plain text, also) , one for XML pages, and one for javascript pages. The latter two have many shortcuts designed to allow you to do less typing, or to integrate more nicely with the specific notation. All of them require knowing at least some Ruby syntax to understand/edit/modify the page templates.

Django uses one "universal" page template format, which has well defined notation and tags, but which one can extend with added tags and "filters".

Django wins for

- human editable templates and
- one template notation to learn, rather than 2 or 3

1.4 Table Maintenance

Django and Ruby both provide mechanisms for browsing and editing tables for which model classes exist; but they provide them differently.

Ruby has a "scaffold" generator script, which generates a code fragment and template files to browse and edit a given database table. The templates are then editable/customizable.

Django has an automatic "admin" tool, which generates screens on the fly from model classes. The behavior of this generation (and whether it is available at all) is modifiable by adding class data to the model class definition.

If you use packages like Archetypes in Zope, you get edit and view screens for your data types; but not otherwise.

1.5 interactive command-line mode

In Ruby and Django, you can launch an interpreter which is running in the environment which your web apps run in, and you can query and examine data, and render page templates, etc. with relative ease.

In Zope, with suitable imports and so on you can approximate the execution environment of your code in the web environment, but you must begin and end object database transactions, etc. by hand.

1.6 Session Management

All three packages provide session management tools; in Zope, the session management is done for you and you can simply refer to contex.REQUEST.session all over the place.

In Ruby and Django, you can initiate a session, choose where session data is stored, and get session data back later in the session.

This is the one category where Zope pretty much wins.

1.7 Forms package

All of these packages have some level of support for generating forms at a higher level, including:

- defining fields with various types and validation
- generating HTML for the form
- rendering validation errors along with the form

however each of them does this somewhat differently.

Zope has (with Archetypes and CMFFormController) a mechanism for generating forms, specifying validation scripts/tools and displaying errors discovered by validators. If you want a form which is not generated by an Archetypes data type, you have a somewhat complicated page template to write, or you can use PloneFormGen or Formulator...

Rails has forms calls that can be used in a page template to iterate through a form specification and render fields; and their scaffold tool will generate such templates from Models.

django has forms calls that directly render HTML for an abstract form/field type, as well as a form/model interface type that you can inherit from and specify a Model, where it will render the form for editing data for that table automatically.

Of these alternatives, I think django has the nicer interface.

1.8 Caching

All of the packages have middle-ware to support caching particular object lookups, generated pages, and (at least for Django) regions of page templates. They differ on how cache expiration is specified and how easy it is to invalidate caches. All support setting caching headers appropriately for upstream web caches (i.e. Squid cache, or Apache's disk/memory caches).

1.9 Programming Language

While Zope is written in Python, it has extended Python nearly to the point of being a different language.

Rails makes very heavy use of language features peculiar to Ruby, so that one needs to understand Ruby semantics very well to follow it. We have very few people at Fermilab with any Ruby experience.

Django is written in, and seems to promote, very straightforward Python code, and we have lots of experienced Python programmers here at Fermilab.

Advantage: Django.

1.10 Performance

Both Rails and Django appear to perform well, and are able to take good advantage of a fast database back-end.

Zope is hindered by its ZODB implementation, which has very poor write performance and is difficult to index effectively.

Conclusions

Overall these three packages cover a lot of the same territory. Zope (particularly with Plone) ships with an already implemented content management system, but if you don't want that included, you have three packages with some very similar attributes.

Therefore, considering the following:

- Performance -- advantage Rails and django over Zope
- existing DB integration -- advantage Rails and django over Zope
- Programming Language Training -- advantage: django
- Template simplicity -- advantage django

I think we conclude that of these packages, Django is the clear winner in our environment.

Recommendations

I recommend that we pursue Django as a platform for web software development in the computing division. Toward this end, we should:

- Get an official Fermi/Django release together [version 0.96 + updated oracle back-end]
- Install on development systems
- Training! Options include:

- Suitcase in class from someplace like: lamptraining or bignerdranch.
- PyCon conference has training as part of the conference. (we missed it here in Chicago in March...)
- Offer training internally