FUDCon 2008, Boston

Luke Macken
lmacken@redhat.com
[ Overview ]

- What is TurboGears2
- TurboGears1 vs. Pylons
- Changes since 1.0
- Why not merge with Pylons?
- SQLAlchemy vs. SQLObject
- Genshi vs. Kid
- WSGI & Middleware
- DBSprockets & DBMechanic
- Testing
- Diving in...
[ whoami ]

- Software Engineer at Red Hat, Inc.
- Member of the Fedora Infrastructure, Release Engineering, and Security Response Teams
- Maintain ~50 packages for Fedora and EPEL
  - Including the TurboGears stack
- Hack on various projects: bodhi, liveusb-creator, PackageKit, yum, TurboGears, func, myfedora, etc...
What is TurboGears2?

- A complete reinvention of TurboGears
- Reimplementation of the TG1.0 API on top of Pylons/Paste
- Provides a fully customizable WSGI stack
- Takes advantage of many new pre-existing components
Both support SQLAlchemy & SQLObject
Both support a wide variety of templating engines, but each have their own preferences. (Kid/Genshi in TG and Mako/Myghty in Pylons)
Both use FormEncode for validation
Both will be using ToscaWidgets in the future
Many other similarities...
Different dispatching mechanisms
- TG uses CherryPy's object dispatching
  - Each path segment becomes an attribute lookup
- Pylons uses Routes pattern matching against the full URL
- TG dispatches to a function that is invoked by CherryPy
- Pylons dispatches to a WSGI application
[ TurboGears1 vs. Pylons ]

• Different controllers
  • TG uses decorators to alter the functionality of your methods
  • In Pylons, you create subclasses to implement controller-specific logic

• Framework Features
  • Pylons: Implemented as WSGI middleware
  • TG: Function decorators

• TG1.0 is heavily tied into CherryPy
[What has changed since 1.0?]

- Deciding to work very closely with Pylons
  - Built the TG1.0 on top of Pylons & Paste
- Using paster instead of the tg-admin wrapper
- Uses the Genshi templating engine by default, instead of Kid
- Uses SQLAlchemy instead of SQLObject by default
- ToscaWidgets
[ Why not merge with Pylons? ]

- Different philosophies
  - Pylons
    - Defaults are chosen for performance and flexibility
    - Gives loose recommendations, but is committed to staying ORM and template agnostic
  - TurboGears
    - Wants to provide a “full stack” out of the box

- “TG is to Pylons as Ubuntu is to Debian”
SQLAlchemy > SQLObject

- Much more efficient SQL queries
- Supports composite keys
- Amazing documentation
- Very active upstream community
Genshi is an intentional re-write of kid
APIs are almost identical
Internally, Genshi is much simpler and faster
Provides full XPath support
Provides useful error messages!
Much larger and more active community
[ WSGI ]

- Web Server Gateway Interface (PEP #333)
- A framework independent specification for how web servers can interact with Python callables
- A standard way for web applications to talk to web servers
- "Think of it as the servlet spec for the Python world" -- Jason Briggs
- "WSGI is a series of tubes" -- Ian Bicking
def wsgi_app(environ, start_response):
    ''' Hello world WSGI application.

    :environ: The WSGI environment. Allows us to get at all kinds of request information.
    :start_response: A callable used to set the server response status and headers.

    Returns an iterable. This allows us to send chunked responses back to the user as they become available.
    '''

    start_response('200 OK', [('content-type', 'text/html')])
    return ['Hello world!']
{ 'HTTP_HOST': 'localhost',
  'PATH_INFO': '/',
  'QUERY_STRING': '',
  'REQUEST_METHOD': 'GET',
  'SCRIPT_NAME': '',
  'SERVER_NAME': 'localhost',
  'SERVER_PORT': '80',
  'SERVER_PROTOCOL': 'HTTP/1.0',
  ...
}
It's just a WSGI application

Doesn't do anything alone, but works in between the request and your application

Essentially the WSGI equivalent of a Python decorator

Instead of wrapping one method in another, you're wrapping one web-app in another
from subprocess import Popen, PIPE

class CowsayMiddleware(object):

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

    def __call__(self, environ, start_response):
        for response in self.app(environ, start_response):
            out, err = Popen(['cowsay', response], stdout=PIPE).communicate()
            yield '<pre>%s</pre>' % out
class HelloWSGIWorldApp(object):

    def __call__(self, environ, start_response):
        start_response('200 OK', [('content-type', 'text/html')])
        return ['Hello WSGI world!']

if __name__ == '__main__':
    from wsgiref.simple_server import make_server

    app = HelloWSGIWorldApp()
    app = CowsayMiddleware(app)

    httpd = make_server('', 8000, app)
    httpd.serve_forever()
[ WSGI Middleware ]

< Hello WSGI world! >

\ /  ^__^  \
(oo)\_________
(____)\        )\/
||-----w ||
||      ||
[ Paste Registry ]

- Registry for handling request-local module globals sanely
- Manages thread-local request-specific objects
- Ensures that your module global is always properly set depending on the current request
- Provides a StackedObjectProxy which is popped/pushed during the request cycle so that it properly represents the object that should be active for the current request
from paste.registry import RegistryManager, StackedObjectProxy

# WSGI app stack (setup for you by TurboGears2)
app = RegistryManager(yourapp)

# Inside your wsgi app
myglobal = StackedObjectProxy()
class YourApp(object):
    def __call__(self, environ, start_response):
        obj = someobject  # The request-local object you want to access
                       # via yourpackage.myglobal
        environ['paste.registry'].register(myglobal, obj)

• This allows you to import your package anywhere in your WSGI app or in the calling stack below it and be assured that it is using the object that you registered with the RegistryManager
Error Traceback:

Exception: OMGEXCEPTION!!!

URL: http://127.0.0.1:8080/explode
Module weberror.evalexception.middleware 364 in respond view
>> app_iter = self.application(environ, detect_start_response)
Module reponse.who.middleware 105 in _call view
>> app_iter = app(environ, wrapper.wrap_start_response)
Module w.core.middleware 30 in _call view
>> return self.wsgi_app(environ, start_response)
Module paste.registry 334 in _call view
>> app_iter = self.application(environ, start_response)
Module w.core.middleware 48 in wsgi_app view
>> return request_response(self.application)(environ, start_response)
Module whecb 1228 in get_response view
Module whecb 1196 in call_application view
Module w.core.resource Injector 56 in Injector view
>> app_iter = app(environ, determine_response_type)
Module beaker.middleware 75 in _call view
>> return self.app(environ, start_response)
Module beaker.middleware 147 in _call view
>> return self.wrap_app(environ, session_start_response)
Module routes.middleware 99 in _call view
>> response = self.app(environ, start_response)
Module pylons.wsgiapp 117 in _call view
>> response = self.dispatch(controller, environ, start_response)
Module pylons.wsgiapp 308 in dispatch view
>> return controller(environ, start_response)
Module tuccon.tucbase 39 in _call view
>> return TController.__call__(self, environ, start_response)
Module pylons.controllers.core 198 in _call view
>> response = self._dispatch_call()
Module pylons.controllers.core 153 in _dispatch_call view
>> response = self._inspect_call(func)
Module pylons.controllers.core 92 in _inspect_call view
>> result = self._perform_call(func, args)
Module tg.controllers 450 in _perform_call view
>> self, controller, params, remainder=remainder
Module tg.controllers 99 in _perform_call view
>> output = controller(*remainder, **dict(params))
Module tuccon.controllers.root 21 in explode view
>> raise Exception, "OMGEXCEPTION!!!"
WebError

Exception: OMGEXCEPTION!!

URL: http://127.0.0.1:8080/exp conco
Module weberror eval_exception.middleware 364 in respond
Module repoze who middleware 105 in __call__
Module tw.core middleware 30 in __call__
Module pasteregistry 34 in __call__

```
Execute Expand

app_iter = None

Exception('OMGEXCEPTION!!',)

environ {'CONTENT_LENGTH': '0', 'CONTENT_TYPE': '', 'HTTP_ACCEPT': 'text/html,application/xhtml+xml,application...

expected False

reg <paste.registry.Registry object at 0x264f956>

self <paste.registry.RegistryManager object at 0x256f590>


```

Module tw.core middleware 48 in wsgi_app

```
Module webob 1228 in get_response
Module webob 1196 in call_application
Module tw.core resource 56 in injector

```
Module tw.core middleware 75 in __call__
Module tw.core middleware 147 in __call__

```
Module routes.middleware 99 in __call__
Module pylons.wsgiapp 117 in __call__
Module pylons.wsgiapp 308 in dispatch

```
Module pylons.wsgiapp 308 in dispatch
Module pylons.wsgiapp 308 in dispatch

```
Error Traceback:

>>> Exception: OMGEXCEPTION!!1

URL: http://127.0.0.1.8080/explode

Module weberror evaluation middleware:364 in respond

Module repoze.middleware:105 in _call_

Module bw.core.middleware:30 in _call_

Module paste.registry:334 in _call_

>>> type(self)
<class 'paste.registry.RegistryManager'>

Execute | Expand
---
app_iter: None

```
e: Exception('OMGEXCEPTION!!1',)

environ: {'CONTENT_LENGTH': '0', 'CONTENT_TYPE': '', 'HTTP_ACCEPT': 'text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8'}

expected: False

reg: <paste.registry.Registry object at 0x264f610>

self: <paste.registry.RegistryManager object at 0x256f590>

start_response: <bound method StartResponseWrapper.wrap_start_response of <repoze.who.middleware.StartResponse object at 0x236f510> >
```

Module bw.core.middleware:48 in _call_

Module bw.core.middleware:1288 in get_response

Module bw.core.middleware:1196 in call_application

Module bw.core.resource_injector:56 in _injector

Module bw.core.resource_injector:75 in _call_

Module bw.core.resource_injector:147 in _call_

Module routes.middleware:99 in _call_

Module pylons.wsgiapp:117 in _call_

Module pylons.wsgiapp:99 in _call_

Module pylons.wsgiapp:79 in _call_

Module pylons.wsgiapp:59 in _call_

Module pylons.wsgiapp:39 in _call_

Module pylons.wsgiapp:17 in _call_

Module pylons.wsgiapp:9 in _call_

Module pylons.wsgiapp:1 in _call_

Module pylons.wsgiapp:1 in _call_

Module pylons.wsgiapp:1 in _call_

Module pylons.wsgiapp:1 in _call_
class RegistryManager(object):
    """Creates and maintains a Registry context
    RegistryManager creates a new registry context for the registration of
    StackedObjectProxy instances. Multiple RegistryManager's can be in a
    WSGI stack and will manage the context so that the StackedObjectProxies
    always proxy to the proper object.
    The object being registered can be any object sub-class, list, or dict.
    Registering objects is done inside a WSGI application under the
    RegistryManager instance, using the `''environ['paste.registry']''`
    object which is a Registry instance.
    """

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

def __call__(self, environ, start_response):
    app_iter = None
    reg = environ.setdefault('paste.registry', Registry())
    reg.prepare()
    try:
        app_iter = self.application(environ, start_response)
    except Exception, e:
        # Regardless of if the content is an iterable, generator, list
        # or tuple, we clean-up right now. If its an iterable/generator
        # care should be used to ensure the generator has its own ref
        # to the actual object
        if environ.get('paste.evalexception'):
            # EvalException is present in the WSGI stack
            expected = False
            for expect in environ.get('paste.expected_exceptions', []):
                if isinstance(e, expect):
                    expected = True
            if not expected:
                # An unexpected exception: save state for EvalException
                reg.cleanup()
                restorer.save_registry_state(environ)
        raise
    except:
        # Save state for EvalException if it's present
        if environ.get('paste.evalexception'):
[ Beaker ]

- Web session and general caching library
- Handles storing for various times any Python object that can be pickled with optional backends on a fine-grained basis
  - Backends include file, dbm, memory, memcached, and database (SQLAlchemy)
- Signed cookie's to prevent session hijacking/spoofing
- Multiple reader/single writer lock system to avoid duplicate simultaneous cache creation
- Extremely customizable
• Arbitrary caching

```python
from tg import TGController, expose
from pylons import cache

class Example(TGController):
    def _expensive(self):
        # do something expensive
        return value

    @expose()
    def index(self):
        c = cache.get_cache("example_cache")
        x = c.get_value(key="my key",
                        createfunc=self._expensive,
                        type="memory",
                        expiretime=3600)
```

[Beaker]
from pylons.decorators.cache import beaker_cache
from tg import TGController, expose

class SampleController(TGController):
    # Cache this controller action forever (until the cache dir
    # is cleaned)
    @expose()
    @beaker_cache()
    def home(self):
        c.data = expensive_call()
        return "foo"

    # Cache this controller action by its GET args for 10 mins to memory
    @expose()
    @beaker_cache(expire=600, type='memory', query_args=True)
    def show(self, id):
        c.data = expensive_call(id)
        return "foo"
[ Beyond Middleware ]

- DBSprockets
  - Provides a simple way to generate web content from available database definitions
  - Utilizes ToscaWidgets and SQLAlchemy
Automatically create a ToscaWidget form based on an SQLAlchemy model

```python
from dbsprockets.primitives import makeForm
from myProject.myModel import User
loginForm = makeForm(User,
    identifier='myLoginForm',
    action='login',
    limitFields=[user_name, password])
```
Automatically create a ToscaWidget form based on an SQLAlchemy model

```python
from dbsprockets.primitives import makeTable, getTableValue
from myProject import User

value = getTableValue(User)
table = makeTable(User, '/',
    omittedFields=['user_id', 'created', 'password'])
table(value=value)
```

<table>
<thead>
<tr>
<th></th>
<th>user_name</th>
<th>email_address</th>
<th>display_name</th>
<th>town</th>
</tr>
</thead>
<tbody>
<tr>
<td>edit</td>
<td>asdf</td>
<td><a href="mailto:asdf@asdf.com">asdf@asdf.com</a></td>
<td>asdf</td>
<td>Arvada</td>
</tr>
</tbody>
</table>
**[DBMechanic]**

- A stand-alone TurboGears controller for database administration

```python
from model import metadata
from dbsprockets.dbmechanic.frameworks.tg2 import DBMechanic
from dbsprockets.sapprovider import SAPProvider

class RootController(TGController):
    dbmechanic = DBMechanic(SAPProvider(metadata), '/dbmechanic')
```

### Tables
- test_table
- permission
- visit
- user_group
- visit_identity
- group_permission
- tg_group
- tg_user
- town_table

#### tg_user

<table>
<thead>
<tr>
<th>AddRecord</th>
<th>TableView</th>
<th>TableDef</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>edit</td>
<td>delete</td>
<td>1</td>
</tr>
<tr>
<td>edit</td>
<td>delete</td>
<td>2</td>
</tr>
<tr>
<td>edit</td>
<td>delete</td>
<td>3</td>
</tr>
</tbody>
</table>
from paste.fixture import TestApp
app = TestApp(config)

class TestTGController:
    def test_index(self):
        response = app.get('/
')
        assert 'Hello WSGI World' in response
[ Ok, lets dive in... ]
ubunut brainstorm

The Ubuntu community has contributed 9720 ideas, 43549 comments, 970330 votes

Brainstorming
- Most popular today
- Most popular this week
- Most popular this month
- Most popular ideas ever
- Latest ideas
- Latest comments
- Random ideas
- Ideas being worked upon
- Implemented ideas
- Search Ideas

Most popular ideas

Fix Suspend and Hibernate
Written by frighten the 28 Feb 08 at 17:22. Category: System. New
Suspend and hibernate still seems to be a big issue based on forum posts. Really focus on fixing it, even with proprietary drivers.
See the 218 comments >>

Provide a simple graphical interface to manage _any_ type of network connection
Written by Alan Pope the 28 Feb 08 at 13:50. Category: Internet & Networking. New
At the moment it's possible to manage traditional wired and WiFi connections using Network Manager. To connect via a modem, a 3G/GPRS card, over bluetooth to a cell phone or via USB to another device requires that the user installs extra packages, and does a fair amount of configuration that isn't found in Network Manager.

A single unified tool should be provided which allows the user to connect to a network (or internet) via any supported method. It would also be useful to provide an extension to this tool to manage firewall rules and network connection sharing.
See the 99 comments >>

Power Management
Written by jsmart the 28 Feb 08 at 16:49. Category: Others. In development
Ubuntu needs to go green. PowerNap, Lesswatts and other tools have finally hit the Linux scene to pave the way for better power management. It needs to be said, "If you want your battery to last longest, or have your energy bill be the lowest, you better use Ubuntu Linux."
See the 65 comments >>
[ We can do better ]

- Enter **Manas**
- Definition: *Intellect, part of the mind that thinks, source of all discrimination; ego-consciousness*
- Real-time comet widgets, powered by Orbited
- Uses jQuery for all of the javascripty aJaxy hotness
- Powered by TurboGears2
from model import metadata
from sqlalchemy import *
from sqlalchemy.types import *

idea_table = Table("idea", metadata,
               Column("id", Integer, primary_key=True),
               Column("title", UnicodeText, unique=True),
               Column("timestamp", DateTime, nullable=False, default=func.now()),
               Column("author", UnicodeText, nullable=False),
               Column("description", UnicodeText, nullable=False),
               Column("karma", Integer, default=0))

comment_table = Table("comment", metadata,
               Column("id", Integer, primary_key=True),
               Column("author", UnicodeText, nullable=False),
               Column("timestamp", DateTime, nullable=False, default=func.now()),
               Column("text", UnicodeText, nullable=False),
               Column("idea_id", Integer, ForeignKey('idea.id')))  
tag_table = Table("tag", metadata,
               Column("id", Integer, primary_key=True),
               Column("name", UnicodeText, nullable=False, unique=True))

idea_tags = Table('idea_tags', metadata,
               Column('idea_id', Integer, ForeignKey('idea.id')),
               Column('tag_id', Integer, ForeignKey('tag.id')))
from sqlalchemy.orm import mapper, relation

class Idea(object): pass
class Comment(object): pass
class Tag(object): pass

mapper(Idea, idea_table, properties={
    'comments': relation(Comment, backref='idea'),
    'tags': relation(Tag, secondary=idea_tags),
})
mapper(Comment, comment_table)
mapper(Tag, tag_table)
from tw.api import Widget, JSLink, js_callback, WidgetsList
from tw.forms import TextField, TextArea
from tw.jquery.activeform import AjaxForm
from formencode.validators importNotEmpty

class NewIdeaForm(AjaxForm):
    success = js_callback('idea_success')
    
class fields(WidgetsList):
        title = TextField('title', validator=NotEmpty)
        tags = TextField('tags', validator=NotEmpty)
        description = TextArea('description',
                                validator=NotEmpty)
        manas_js = JSLink(link='/javascript/manas.js')

class IdeaWidget(Widget):
    template = 'genshi:manas.templates.ideawidget'
    params = ['idea']
new_idea_form = NewIdeaForm('new_idea_form', action=url('/save'))

class RootController(BaseController):

    @expose('manas.templates.new')
    @authorize.require(authorize.not_anonymous())
    def new(self):
        pylons.tmpl_context.new_idea_form = new_idea_form
        return {}
[ Template ]

<html xmlns="http://www.w3.org/1999/xhtml"
     xmlns:py="http://genshi.edgewall.org/">
  <head><title>Submit a new idea</title></head>
  <body>
    <h2 class="notice">Got an idea?</h2>
    ${tmpl_context.new_idea_form()}
  </body>
</html>
Got an idea?

Title
Replace RPM with Conary

Tags
rpm, package management

Description
Blah blah blah

Submit
[ Saving ideas ]

```python
@expose('json')
@validate(new_idea_form)
@authorize.require(authorize.not_anonymous())
def save(self, title, description, tags):
    if pylons.tmpl_context.form_errors:
        return dict(idea='fail')
    idea = Idea(title=title, description=description,
                author=pylons.tmpl_context.identity.user_name)
    DBSession.save(idea)
    for tag in tags.split(',',):
        tag = Tag(name=tag)
        DBSession.save(tag)
        idea.tags.append(tag)
    DBSession.commit()
    flash("Your idea was successfully created!")
    return dict(idea=idea)
```
[ Real-time widgets ]

- Powered by Orbited
  - Web server designed for real-time applications
  - Allows for asynchronous server-push messages to be sent to clients
  - Cross browser compatible
  - Highly scalable
from tw.api import Widget, js_function, JSLink
from tw.jquery import jquery_js
orbited_js = JSLink(link='http://localhost:8000/_/orbited.js')
manas_js = JSLink(link='javascript:manas.js')

class LatestIdeas(Widget):
    params = ['id']
    template = 'genshi:manas.templates.latestideas'
javascript=[orbited_js, jquery_js, manas_js]
    include_dynamic_js_calls = True

def update_params(self, data):
    super(LatestIdeas, self).update_params(data)
    event_cb = js_callback('function(data) {
$.each(data, function(i, item) {
  $('<div/>').hide().append(
    $('<a/>' + item['title'])
  ).prependTo("#%s_data").slideDown();
});

event_cb("%s_data").slideDown();

self.add_call(js_function('connect')(event_cb, data.user, '/orbited', 0))
from pyorbited.simple import Client
orbited = Client()
 orbited.connect()

@expose()
def join(self, user):
    if (user, '0') not in self.users:
        self.users.append((user, '0'))

        # Throw the latest entries at the user
        for idea in DBSession.query(Idea).order_by('timestamp')[:10]:
            orbited.event(['%s, 0, /orbited' % user],
                           [{'id': idea.id, 'title': idea.title}])

        return 'ok'

def save(self, title, description, tags):
    # Save the idea
    # ...
    orbited.event(self._user_keys(),
                  [{'id': idea.id, 'title': idea.title}])

    return dict(idea=idea)

def _user_keys(self):
    return ['%s, %s, /orbited' % (user, session)
            for user, session in self.users]
Questions? 
[ References ]

- TurboGears2
  - http://turbogears.org/2.0/docs/index.html
- TurboGears and Pylons (A technical comparison)
- Paste
  - http://www.pythonpaste.org
- DBSprockets
  - http://code.google.com/p/dbsprockets/
- Orbited
  - http://www.orbited.org/
- ToscaWidgets
  - http://toscawidgets.org
- SQLAlchemy
  - http://www.sqlalchemy.org
- Ubuntu Brainstorm
  - http://brainstorm.ubuntu.com