IST FP6-004779

PYPY

Researching a Highly Flexible and Modular Language Platform and Implementing it by Leveraging the Open Source Python Language and Community

STREP

IST Priority 2

Release Automated Testing Framework with html/pdf Reports and Runtime http-Server for Introspection Aiding Debugging and Understanding of PyPy Internals

Due date of deliverable: March 2006

Actual Submission date: XXX insert submission date here l

Start date of Project: 1st December 2005 **Duration: 2 years**

Lead Contractor of this WP: merlinux

Authors: Holger Krekel, Carl Friedrich Bolz

Revision: Draft

Project co-funded by the European Commission within the Sixth Framework Programme (2002-2006)

Dissemination Level: PU (Public)

Revision History

Date	Name	Reason of Change
Dec 01 2006	Carl Friedrich Bolz	created a rough structure, outline
Jan 16 2006	Holger Krekel	Draft of Executive summary
Jan 21 2006	Holger Krekel	refined exec summary, outline & abstract
Jan 28 2006	Carl Friedrich Bolz	Publishing of intermediate version on the web

Abstract

This report describes the purpose and implementation of PyPy's testing process. We particularly discuss unique features like ad-hoc distributed testing, ReStructured text/HTML reports, overall integrity tests and an HTTP server which allows to easily introspect failures. We also describe our development approach with respect to automated tests as a means to maintain good code quality.

Contents

1 Executive Summary

Development and research within the PyPy project relies and depends on automated testing and a test-driven development approach. For its complex technical testing needs, the project developed the `py.test` tool. It is possible to use `py.test` for more than just the PyPy code base. In fact, during the EU project period many individuals and companies adapted the usage of the tool.

`py.test` has been developed as a part of the "py lib", a library supporting diverse areas of development such as writing documentation, ad-hoc distribution of programs and tests and a flexible api to access conventional and transactional filesystems. The library has been separately released on XXX.

`py.test` internally uses PyPy for compiling Python code to Javascript and thereby validates both that PyPy's compiler can be used for compiling tested python code to a modern AJAX application, and that our testing tool is flexible enough to integrate this new reporting method.

The Python community traditionally used a Python version of the JUnit approach (itself derived from Smalltalk's SUnit) and several projects exist for enhacing and amending the testing process. `py.test` is a project-independent tool configurable for various testing needs. Its usage from the PyPy developers showcases its flexibility in adapting to complex testing needs. Recent developments also introduce ad-hoc distributed testing, which allows to distribute PyPy's thousands of tests to multiple computers. This is effectively speeding up the development process.

`py.test` aims at maximizing code-sharing between developers for their complex testing needs. Each project can amend or substitute parts of the collecting, execution and reporting process. Consequently, `py.test` not only tests code execution but also documentation and overall integrity.

Another important design goal is to minimize the overhead needed to implement a testing process suitable for the needs of the project. Particular emphasis lies on providing helpful and detailed feedback to the developer in case of test failures (XXX and a simple api?). There are several facilities to observe the full context and introspect a running program in case of such test failures.

PyPy's testing tool today is used increasingly by developers and organisations, and in fact there have been and are commercial opportunities for consulting about integration of it into new environments. The tool along with the "py lib" will continue a life of its own after the EU project period.

2 Introduction

2.1 Purpose of this Document

This document describes:

- The py.test and py lib release, its impact and features
- PyPy's testing approach and particular methods integrated into the testing process
- Technical details and references to several features, including distributed testing, generating HTML pages out of the test reports, a web server for viewing running tests and several so-called 'magic' features which enchance output in case of broken tests.

2.2 Scope of this Document

This documents `py.test` as published in the 0.9 release early 2007.

2.3 Related Documents

XXX not sure * something like D02.1, I guess

3 Motivation

XXX: Describe where unittest fails here

Python 'out-of-the-box' ships with a unit testing framework called 'unittest', which is (loosely) based on Java's JUnit, in turn based on Smalltalk's testing framework. Python's unittest is a very traditional, framework-ish library which expects users to write subclasses of a special TestCase base class, which provides some 'assert*' methods (e.g. assertEqual, assertFalse, assertRaises) to allow testing the code in 'test*' methods (user-implemented methods which have a name starting with 'test'), which are in turn called by the framework.

There are several problems with this approach:

- Unpleasant API
- Hard to enchance (ie. way we perform documentation tests)
- Relatively uninforming results (without all the 'magic')
- Lacking certain features (such as distributed testing)
- No configuration at all

Design goals:

- No, or almost no boilerplate (lightweight)
- No, or almost no API
- Very flexible and easy to enchance
- Split into abstraction layers (XXX not sure about this one) (I'd say leave it out - Guido)
- Highly configurable (conftest)

4 py.test Feature highlights

4.1 Basic Features

- typical feature list + usage description, can probably more or less be taken from docs
- can perform tests without boilerplate:

```
def test_myfunc():
  assert myfunc(10) == 20
  assert myfunc(20) == 40
```

- assert reinterpretation
- automatic tests collection
- generative tests
- keyword-select
- --exec
- stdout/stderr capturing
- recursion detection, useful tracebacks, local variables
- setup/teardown hooks on all levels
- can perform tests in a distributed manner (multiple hosts running different tests)
- the output can be customised
- can do 'boxing' (sandboxing, tests don't share context) and such

4.2 Customization and Configuration

Conftest is a flexible (XXX and confusing (lol - Guido)) way of configuring py.test.

4.3 Distributed Testing

4.4 Reporter class and different reporters

- rest reporter
- web reporter

4.5 Boxing

XXX

4.6 Apigen

5 Usage of Testing Framework in PyPy

- apptests, documentation tests, tests on pypy-c
- automated, nightly tests - very useful, quality tool
- Distributed Testing

6 Comparison with other Python testing tools

XXX maybe under motivation

7 Release remarks

XXX describe release(s), including the SVN-based work in 2005/2006

8 Future Development

The `py.test` tool has been incrementally developed over the last years (by means of using test-driven approach itself) and many projects and developers started using it from the publically accessible codespeak subversion repository. It has been presented at multiple Python Community conferences and beginning 2007, the first full release was prepared and published. Several outside developers stated that they enjoy the ability to follow the development of the tool and have hardly encountered problems just using the "latest" repository version - the same approach taken for developing the PyPy interpreter and compiler code base.

The development of the `py lib` along with its tools will continue after the EU project phase, and there are commercial opportunities on extending, consulting and teaching `py.test` and the benefits of using a thoroughly employed testing process and tool, relaying the experiences obtained during the execution and implementation of the PyPy EU project.

9 Glossary of Abbreviations

The following abbreviations may be used within this document:

9.1 Technical Abbreviations:

AST	Abstract Syntax Tree
CPython	The standard Python interpreter written in C. Generally known as "Python". Available from www.python.org.
codespeak	The name of the machine where the PyPy project is hosted.
CCLP	Concurrent Constraint Logic Programming.
CPS	Continuation-Passing Style.
CSP	Constraint Satisfaction Problem.
docutils	The Python documentation utilities.
F/OSS	Free and Open Source Software
GC	Garbage collector.
GenC backend	The backend for the PyPy translation toolsuite that generates C code.
GenLLVM backend	The backend for the PyPy translation toolsuite that generates LLVM code.
Graphviz	Graph visualisation software from AT&T.
Jython	A version of Python written in Java.
LLVM	Low Level Virtual Machine - a compiler infrastructure available from University of Illinois at Urbana-Champaign
LOC	Lines of code.

Object Space	A library providing objects and operations between them, available to the bytecode interpreter via a well-defined API.
Pygame	A Python extension library that wraps the Simple Direct-media Library - a cross-platform multimedia library designed to provide fast access to the graphics framebuffer and audio device.
pypy-c	The PyPy Standard Interpreter, translated to C and then compiled to a binary executable program
ReST	reStructuredText, the plaintext markup system used by docutils.
RPython	Restricted Python; a less dynamic subset of Python in which PyPy is written.
Standard Interpreter	The subsystem of PyPy which implements the Python language. It is divided in two components: the bytecode interpreter, and the standard object space.
Standard Object Space	An object space which implements creation, access and modification of regular Python application level objects.
VM	Virtual Machine.

9.2 Partner Acronyms:

DFKI	Deutsches Forschungszentrum für künstliche Intelligenz
HHU	Heinrich Heine Universität Düsseldorf
Strakt	AB Strakt
Logilab	Logilab
CM	Change Maker
mer	merlinux GmbH
tis	Tismerysoft GmbH
Impara	Impara GmbH