VMController

A general purpose virtual machine controller

Rohit Yadav
IDD Part IV
07020003

CS-4303

Department of Computer Engineering
Institute of Technology
Banaras Hindu University

Project Guide:
Asst. Prof. Bhaskar Biswas

Submitted in partial fulfilment of the requirements
for the degree of Bachelor of Technology

DEPARTMENT OF COMPUTER ENGINEERING
INSTITUTE OF TECHNOLOGY
BANARAS HINDU UNIVERSITY
VARANASI-221005, (U.P.) INDIA

CERTIFICATE

This is to certify that Mr. Rohit Yadav, fourth year undergraduate student of the Department of Computer Engineering, Institute of Technology, Banaras Hindu University, has worked on his project entitled "VMController, a general purpose virtual machine controller" under my direct supervision during the period August 2010 – November 2010, the findings of which have been incorporated in this report. He has worked diligently, meticulously and methodically. The report submitted by him embodies the original work done by him during the development of the project. The project report has been found satisfactory and is approved for submission.

Bhaskar Biswas
Assistant Professor
Department of Computer Engineering
Institute of Technology
Banaras Hindu University

ACKNOWLEDGEMENT

I would like to thank my project guide Asst. Prof. B. Biswas for his support, patience and guidance. I'm also thankful to all the technical and non-technical staff of the Department of Computer Engineering for their co-operation.

VMController is based on *Boincvm Controller* authored by Mr. David Garcia Quintas in 2009 at CERN. Many thanks to David for his contribution, support, suggestions and emails.

Lastly, my thanks and appreciation to Dr. Ben Segal and his team at LHC++@Home, CERN for their interest and encouragement.

Rohit Yadav
Part IV, Integrated Dual Degree
Department of Computer Engineering
Institute of Technology
Banaras Hindu University

ABSTRACT

VMController is a multi-platform scalable infrastructure that provides means to interact with a guest system running inside a virtual machine instance of a hypervisor. It is written in Python and developed using a dozen of open source tools and packages. With VMController a client programmer can write a powerful application to control, manage, monitor, operate and administrate multiple virtual machines using XMLRPC APIs.

Keywords:

Hypervisor, virtual machine, controller, cloud computing, VirtualBox, VMWare, open source, XMLRPC, broker, file transfer, process execution, sandboxing, Python, Twisted, STOMP, dependency injection, DI, BOINC.

Contents

1. Introduction

VMController is a general purpose cross-platform virtual machine controller, based on *Boincvm Controller* [2][3] authored by *David Garcia Quintas* in 2009 at CERN [1]. The aim of this project is to create a multi-platform hypervisor-agnostic system that can automate host-guest virtual machine communications, provide interfaces to do things between guest and host systems like process execution, file transfer, resource management and monitoring etc. [6]

Before VMController and its prototype predecessor *Boincvm Controller* there was no software infrastructure which could provide an automated means to interaction and communication between a guest system running inside a virtual machine, and the host system, hence the motivation for this project.

VMController is written in Python and uses a lot of open source Python tools, packages and API bindings besides standard packages such as; Twisted, stomper, MorbidQ, CoilMQ, Inject, netifaces, uuid, distribute, zc.buildout and VirtualBox python sdk.

At its current state, VirtualBox is the only supported hypervisor but it is designed to support any other hypervisor with an exposed API. With VMController powerful programs can be written to control, manage, monitor, operate and administrate multiple virtual machines using XMLRPC APIs. VMController is open-sourced under BSD license. [4][5]

1.1 Objectives

VMController is developed with following objectives:

- Hypervisor agnostic.
- Open source.
- Cross platform and scalable.
- Easy to use, extend, build, package and deploy.
- Flexible message passing protocol.
- Highly decoupled.

2. Architecture

The following diagram captures a bird's eye view of VMController's architecture:

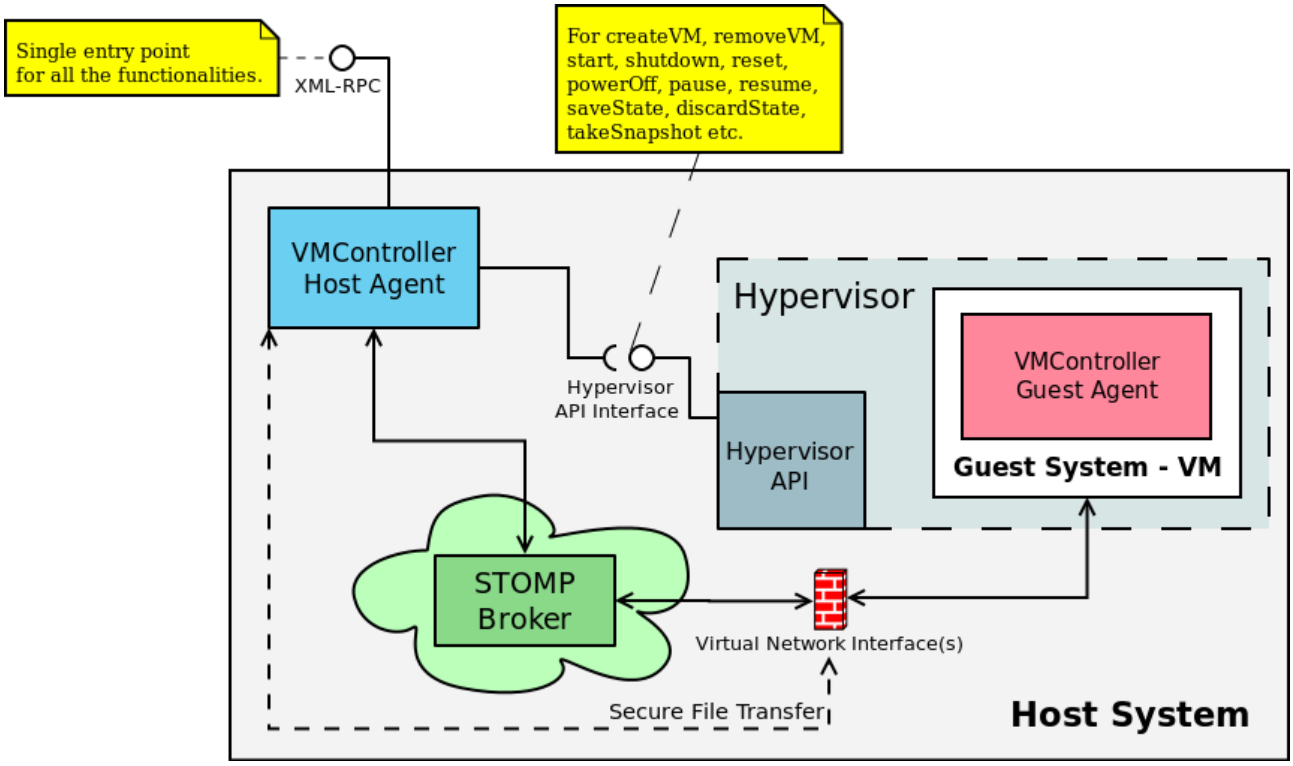

Fig 1. VMController's Architecture

VMController has three main components:

- VMController Host agent which runs on the host system.
- VMController Guest agent which runs on the guest system, running inside a VM.
- STOMP Message Broker which is basically a STOMP server with which the agents (clients) pass messages between each other.

The hypervisor has an exposed API interface with which the Host agent can perform operations like creating a VM, starting a VM, saving a VM's state etc. The Guest agent connects with the STOMP broker, the Host agent and/or the outside world via the virtual *host-only-adapter* and NAT network interface provided by the hypervisor. The Host agent has an XMLRPC interface which is a single entry point to all the functionalities.

3. Design and Implementation

3.1 Development

For development and testing, a custom assembled server with following configuration was used:

- CPU: Intel Core i7 930 x86_64, 3 Ghz
- RAM: 4 GB DDR3, 1066 Mhz
- Storage: 1 TB 7200RPM HDD
- Host OS: Ubuntu GNU/Linux 2.6.35-22 x86_64

VMController is written in Python [7][21]. The project is hosted at Google Code [4] and open sourced under the BSD license [5]. The project hosting provides a distributive version control system, issue tracker, wiki and file hosting.

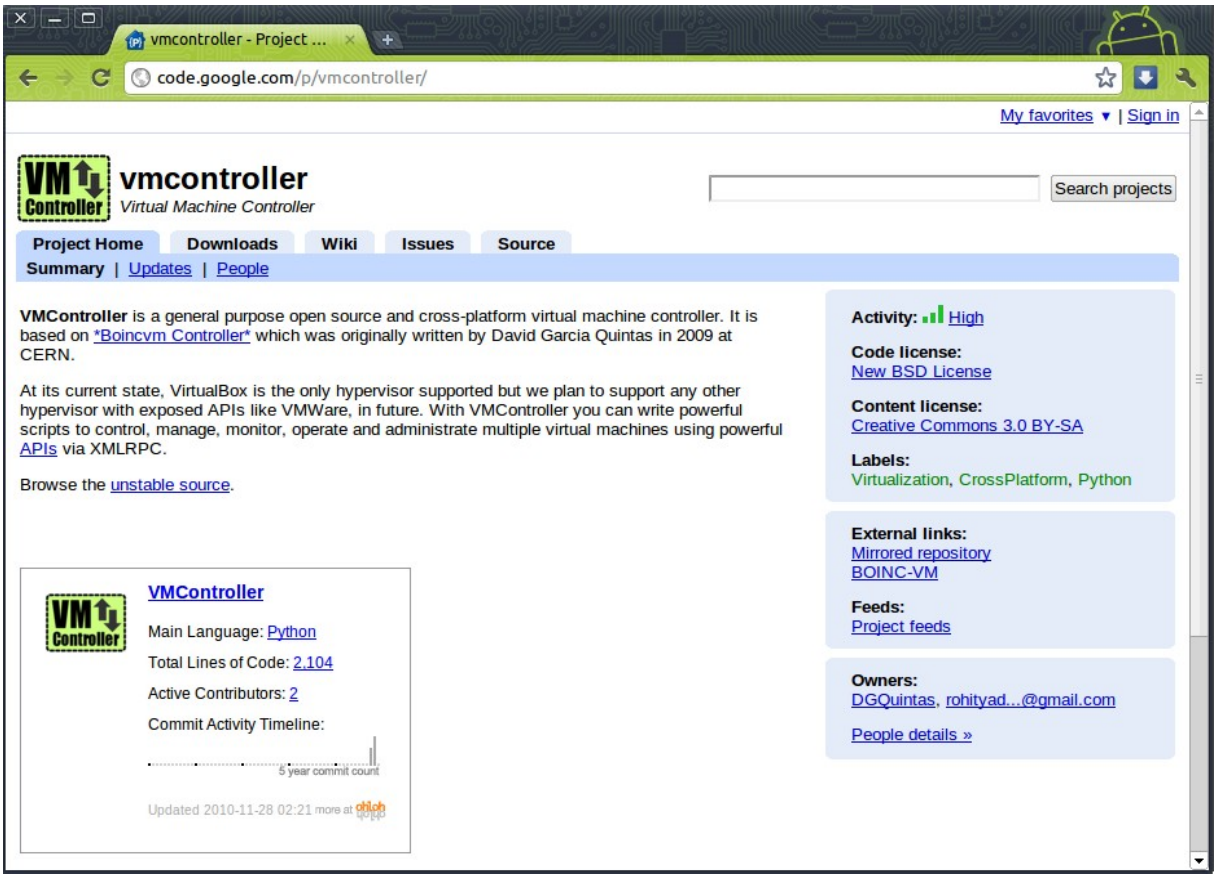

Fig 2. Project Hosting at Google Code

3.1.1 Version control

For version controlling Mercurial *hg* [8] a cross-platform, distributed revision control tool is used. One can get a local copy of the repository with this command:

```
hg clone https://unstable.vmcontroller.googlecode.com/hg/ vmcontroller
```

3.1.2 Build System and Documentation

For building the project, *zc.buildout* [9] is used. A build script evokes the *bootstrap.py* from zc.buildout, which reads a *buildout.cfg* configuration file and creates a script named *buildout* in */bin* folder of the source directory. Then this script scans the source code, downloads all the dependencies from Internet and creates entry console scripts for VMController Host and Guest modules. To build, we run this command while in root source directory:

```
./build
```

Or:

```
python bootstrap.py -d
bin/buildout -v
```

For automatic API documentation generation, *epydoc* [20] is used. Epydoc scans the source code and builds documentation. To automatically build the documentation in */docs/api*, run the following command while in root source directory:

```
epydoc –config=docs/epydoc.cfg
```

An online documentation can be found here:

```
http://packages.python.org/vmcontroller/
```

3.2 Dependency Injection

Besides the general design patterns like Command, Iterator, Strategy, Factory, Facade etc. *Dependency Injection* (DI) [10] is most prominently used design pattern in the implementation of VMController using *Python Inject* [15] to inject objects which can be accessed anywhere in application scope. DI is a technique that indicates to a part of a program which other parts it can use, i.e. to supply an external dependency to a software component and it separates behaviour from dependency resolution, thus decoupling highly dependent components.

3.3 Message Passing

Message passing provides a means of communication between the host and the guest system. VMController has an XMLRPC interface in the Host agent, STOMP based client implementation for message passing in both Host and Guest agents and a STOMP broker which runs on the host system.

3.3.1 Networking

Python based event-driven network programming framework, *Twisted* [13] is used to implement asynchronous networking in VMController. VirtualBox provides virtual network interfaces through which VMController Host and Guest agents can interact. Two kind of interfaces are used for a guest system; a *NAT* interface for general networking and a *host-only-adapter* interface which is a hybrid between the bridged and internal networking modes through which the virtual machines can talk to each other and the host. The NAT is used for STOMP messaging and general access to Internet and the host-only-adapter is used for special operations like file transfer.

The MAC address of the hypervisor managed virtual network card is a common piece of unique identifier known by both the guest and host systems. Therefore, MAC addresses are used as identifiers. System's network interfaces and their addresses are found using *netifaces*. Every virtual machine has a UUID [19] associated with it, which is used by VirtualBox for its various API calls; in VMController Python package *uuid* is used for that purpose.

VMController host agent functionalities are made accessible through an XMLRPC interface. XMLRPC aims to provide a simple yet fully functional, standard and multi-platform mechanism of communication between this host agent and the outside world.

3.3.2 Broker

Message passing is implemented on a lightweight STOMP protocol [16]. The broker is a STOMP server that processes STOMP messages between the Host and Guest VMController agents. *Stomper* [17] which is a python client implementation of transport layer neutral STOMP protocol is used to extend message passing implementation in Host and Guest modules of VMController. Tailor made STOMP messages are used which have the structure: {HEADERS, BODY}. The HEADERS holds identifiers and the BODY holds the custom message.

VMController has inbuilt support for two STOMP brokers: MorbidQ and CoilMQ. *MorbidQ* [18] is the default broker and is favoured over CoilMQ because it is a lightweight reusable Twisted module. The broker runs on the host system and listens at the default STOMP port 61613.

3.4 Hypervisor

By design VMController is hypervisor agnostic, the present version supports VirtualBox only. VirtualBox [11] was chosen because it is a mature free software with a big user base and an exposed programming interface, Microsoft COM on Windows and Mozilla XPCOM on all the other host operating system. VMController uses VirtualBox's Python SDK [12] which provides a VirtualBoxManager instance

with which it can control a running instance of the VirtualBox.

```
from vboxapi import VirtualBoxManager
_vboxmgr = VirtualBoxManager(None, None)
```

3.5 XMLRPC API Specification

A total of 30 APIs are identified and implemented. They are classified as creational, behavioural, state based, host-guest based and miscellaneous.

3.5.1 Creational

- **createVM:** Creates a VM with a given name and path to VM disk image.
- **removeVM:** Remove a VM with a given name.

3.5.2 Behavioural

- **start:** Starts a VM.
- **shutdown:** Shuts down (ACPI) a VM.
- **sleep:** Puts a VM with a given name to sleep (ACPI).
- **reset:** Reboots a VM.
- **powerOff:** Turns off power of a VM.
- **pause:** Pauses a running VM.
- **resume:** Resumes a paused VM.

3.5.3 State Based

- **getState:** Gets state of a VM with a given name.
- **saveState:** Saves state of a running VM.
- **discardState:** Discards previously saved state of a VM.
- **takeSnapshot:** Takes snapshot of a VM and names it to a user specified value.
- **restoreSnapshot:** Restores a snapshot of a VM.
- **deleteSnapshot:** Deletes a snapshot of a VM.

3.5.4 Host-Guest Based

- **ping:** Sends a ping message to guest agent running inside a VM with a given timeout. Default timeout = 5 seconds.
- **runCmd:** Runs a command in a VM with arguments, environment, path and file for stdin.

- **listFinishedCmds:** Returns a dictionary of finished commands ran inside a VM with details.
- **getCmdResults:** Returns result of a command execution inside a VM.
- **getCmdDetails:** Returns command execution details like resource, CPU, memory usage.
- **cpFileToVM:** Copies a local file in the host system to a location inside a VM.
- **cpFileFromVM:** Copies a file from a VM to a local path in the host system.

3.5.5 Miscellaneous

- **help:** Returns a help string with brief information about available APIs.
- **listVMs:** Lists all the available VMs with the hypervisor.
- **listVMsWithState:** Lists all the available VMs with the hypervisor with state information.
- **listRunningVMs:** Lists all the VMs with the state *Running*.
- **listSnapshots:** Lists all the available snaphots of a VM.
- **getIdsToNamesMapping:** Returns a dictionary with ID as keys.
- **getNamesToIdsMapping:** Returns a dictionary with VM names as keys.
- **getPerformanceData:** Gets performance data of a VM.

3.6 Implementation

VMController is implemented and divided into three Python packages:

- vmcontroller.host: Host agent package.
- vmcontroller.guest: Guest agent package.
- vmcontroller.common: Common package between host and guest packages.

In both host and guest packages, a StompProtocol is instantiated by a StompProtocolFactory. This StompProtocol interacts with BaseStompEngine which is extended in host package as HostStompEngine and in guest package as VMStompEngine. Both of these classes have a set of words defined, HostWords for host package and VMWords for guest package, which define how to read and react to a STOMP message. The MsgInterpreter interprets a message from StompEngine and acts the words. Both the Host and VM classes have access to the words and StompEngines. The Host class in HostServices handles the HostXMLRPCService. A class diagram of the same is shown in Fig 3.

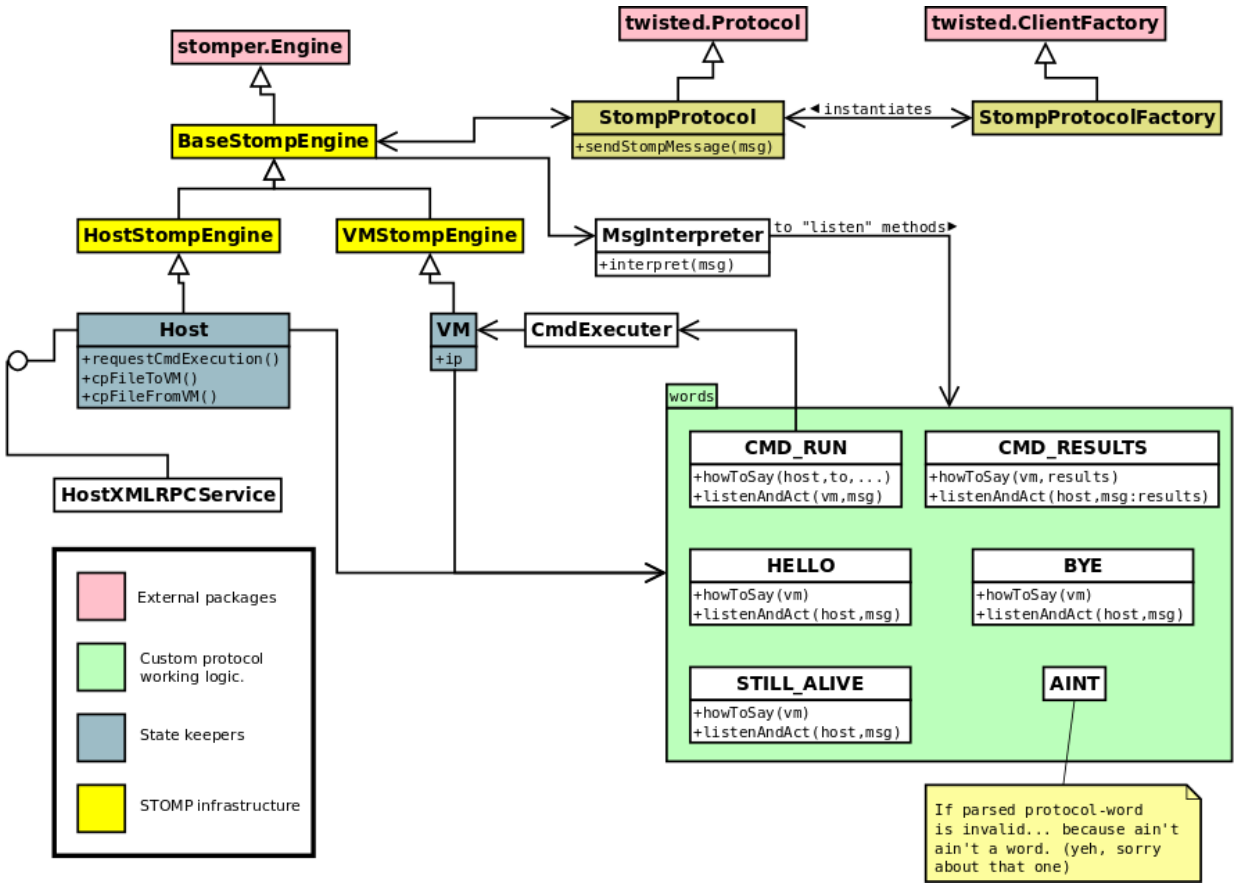

Fig 3. VMController Class diagram

3.7 Testing

Unit testing is used for testing various components of VMController. *Trial* [14] from Twisted Framework, *twisted.trial*, is used in all the test scripts for unit testing. These test scripts can be found in */tests* of the root source directory. To run tests, in the root source directory use this command:

```
trial tests/
```

4. Conclusions

The development of VMController has reached beta level. Almost all of its APIs are working correctly. The software is packaged [22] as Python eggs and is available from PYPI:

Host: http://pypi.python.org/pypi/vmcontroller.host
Guest: http://pypi.python.org/pypi/vmcontroller.guest
Common: http://pypi.python.org/pypi/vmcontroller.common

One can now install VMController using easy_install or pip, for example:

```
easy_install vmcontroller.host
```

or,

```
pip install vmcontroller.host
```

4.1 Applications

VMController is aimed at automation of the BOINC-VM [6] system. On successful deployment, in BOINC-VM, the VMWrapper should be able to execute VMController and using the XMLRPC APIs create a VM, start it, execute jobs inside the guest system and get back results.

4.2 Further Work

Ruggedisation and testing on all major operating systems. Some new set of interfaces. Support for VMWare. Dependency free packaged installer for Windows, Mac and Linux.

5. Reference

[1] David Garcia Quintas, BOINC Virtual Machine Controller Infrastructure, CERN, 2009, http://boinc.berkeley.edu/trac/attachment/wiki/WorkShop09/VMControllerInfraestructure_DGQ.pdf

[2] David Garcia Quintas, A host <-> guest VM communication system for Virtual Box, 2009, http://boinc.berkeley.edu/trac/wiki/VirtualBox

[3] Boincvm Controller Repository, http://bitbucket.org/dgquintas/boincvm

[4] VMController, Project Hosting, http://code.google.com/p/vmcontroller

[5] BSD License: http://www.opensource.org/licenses/bsd-license.php

[6] Ben Segal, David Weir, Kevin Reed et al, General considerations for "Running apps in virtual machines, BOINC, 2008, http://boinc.berkeley.edu/trac/wiki/VmApps

[7] Python, Official Website, http://python.org

[8] Mercurial, Distributed Version Control, http://mercurial.selenic.com

[9] Tarek Ziade, zc.buildout: Expert Python Programming, Packt Publishing, 2008: 167-181

[10] Martin Fowler, Inversion of Control Containers and the Dependency Injection pattern, 2004, http://martinfowler.com/articles/injection.html

[11] VirtualBox, Open-source desktop Hypervisor, http://virtualbox.org

[12] VirtualBox SDK documentation, http://www.virtualbox.org/sdkref/index.html

[13] Twisted, event-driven network programming framework, http://twistedmatrix.com

[14] Trial, Unit testing with Trial, http://twistedmatrix.com/documents/current/core/howto/testing.html

[15] Python Inject, Python dependency injection, http://code.google.com/p/python-inject

[16] STOMP, Streaming Text Oriented Message Protocol, http://stomp.codehaus.org

[17] Stomper, Transport neutral client implementation of the STOMP protocol, http://code.google.com/p/stomper

[18] MorbidQ, STOMP publish/subscribe server, http://www.morbidq.com/trac/wiki/Start

[19] UUID, Universally Unique Identifier, http://tools.ietf.org/html/rfc4122

[20] Epydoc, Automatic API Documentation Generation for Python, http://epydoc.sourceforge.net

[21] Mark Pilgrim, Dive into Python, Apress, 2004, http://diveintopython.org
[22] Packaging Python apps, http://diveintopython3.org/packaging.html

6. Glossary

Advanced Configuration and Power Interface (ACPI): A specification to provide an open standard for unified operating system-centric device configuration and power management.

COM: Component Object Model is a binary-interface standard for software componentry introduced by Microsoft. It is used to enable interprocess communication and dynamic object creation in a large range of programming languages.

Hypervisor: A hypervisor, also called virtual machine monitor (VMM), is one of many virtualization techniques which allow multiple operating systems, termed *guests*, to run concurrently on a host computer. It presents to the guest operating systems a virtual operating platform and monitors the execution of the guest.

Middleware: A computer software that connects software components or some people and their applications.

Network Address Translation (**NAT**): The process of modifying network address information in datagram (IP) packet headers while in transit across a traffic routing device for the purpose of remapping one IP address space into another.

Message Oriented Middleware (MOM): A software infrastructure focused on sending and receiving messages between distributed systems.

Python: An interpreted, general-purpose high-level programming language which supports multiple programming paradigms, primarily but not limited to object oriented, imperative and, to a lesser extent, functional programming styles.

SDK: Software development kit.

Streaming Text Oriented Message Protocol (STOMP): is a simple text-based language-agnostic protocol, designed for working with Message Oriented Middleware. It provides an interoperable wire format that allows Stomp clients to talk with any Message Broker supporting the protocol.

STOMP Broker: A STOMP server with which STOMP clients can communicate to pass STOMP messages.

Unit Testing: A method by which individual units of source code are tested to determine if they are fit for use.

Virtual Machine (VM): A software implementation of a machine (i.e. a computer) that executes instructions like a physical machine.

XMLRPC: A remote procedure call (RPC) protocol which uses XML to encode its calls and HTTP as a transport mechanism.

XPCOM: Cross Platform Component Object Model (COM) is a cross-platform component model from Mozilla.