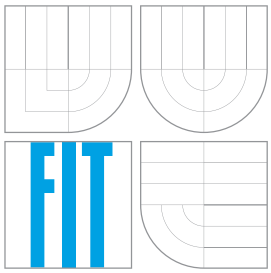
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

ROZŠÍŘENÍ BALÍČKOVACÍHO NÁSTROJE APT PRO KOMUNIKACI SE SYSTÉMEM SPACEWALK

SPACEWALK PLUG-IN FOR THE APT PACKAGING TOOL

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

ŠIMON LUKAŠÍK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. ALEŠ SMRČKA, Ph.D.

BRNO 2011

Abstrakt

Spacewalk je opensource nástroj pro správu Linuxových systémů. Spacewalk pracuje se systémy Fedora a Red Hat Enterprise Linux, na kterých je možné z něj stáhnout balíčky pomocí nástrojů yum a yum-rhn-plugin. Avšak pro systémy Debian takový nástroj neexistuje. Tato práce navazuje na diplomovou práci Lukáše Ďurfiny, který do Spacewalku přidal podporu pro Debian balíčky. Cílem této práce je vytvořit rozšíření nástroje apt-get, které umožní stažení balíčku ze Spacewalku na Debian systém.

Abstract

Spacewalk is an open source management solution for Linux systems. It works well with Fedora and Red Hat Enterprise Linux, where one can install and upgrade packages using yum and yum-rhn-plugin tools which download packages from Spacewalk. Unfortunatelly, for Debian, there is no such tool. This work reassumes Master Thesis of Lukáš Ďurfina, who added initial support for Debian in Spacewalk Server. The main goal is to write a plug-in for apt-get, which allows downloading the packages for a Debian system from Spacewalk.

Klíčová slova

GNU/Linux, Správa balíčku, Správa systémů, APT, YUM, Spacewalk

Keywords

GNU/Linux, Package Management, Systems management, APT, YUM, Spacewalk

Citace

Šimon Lukašík: Spacewalk Plug-in for the APT Packaging Tool, bakalářská práce, Brno, FIT VUT v Brně, 2011

Spacewalk Plug-in for the APT Packaging Tool

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Aleše Smrčky, Ph.D. a Mgr. Miroslava Suchého.

.....
Šimon Lukašík
May 18, 2011

Poděkování

Děkuji svému vedoucímu práce Ing. Aleši Smrčkovi, Ph.D. a mému konzultantu z firmy RedHat Mgr. Miroslavu Suchému za odbornou pomoc a podporu při vytváření práce.

Contents

Chapter 1

Introduction

The software has an unpleasant attribute that it tends to become obsolete. Thus, it needs to be replaced with a newer version many times in the life cycle. The process of replacement or so-called update raises numerous questions which need to be answered like: which tool is responsible for the update, if it is a single tool or do the different bits of software get managed by different tools, or how is the new content to be delivered. In addition, the tool installing the software should validate the content authenticity, and it should execute any necessary actions prior or after the update. The example of such action is a resolution of software's dependencies and conflicts.

The GNU/Linux distributions reply to the mentioned problems with a centralized package management system, where all the delivered bits are divided into the packages, considering the origin, the author or the internal logic, and managed by a single tool. Across the distributions, there is a variety of package management systems and package formats. Having the communities around the GNU/Linux and their desire to share a content, packages are usually distributed freely over the Internet by the means of publically available storages—repositories.

The upper mentioned model works well for the community, but in a corporate computing environments, there might be higher expectations and the demands for a tool which would provide an additional value answering the following questions: how to manage a huge number of machines with a less effort, how to verify and audit installed packages, at which point of time to execute the update, or how to set up the whole infrastructure to be updated in a relatively small time frame. The another question might be: how to do an accounting to charge the package consumers.

The Spacewalk server [1] is an example of a corporate ready solution of systems management. Spacewalk was designed to manage RPM based GNU/Linux distributions. This work reassumes and extends the Master thesis of Lukáš Ďurfina [2] who added initial support for Debian clients to the Spacewalk Server as a proof of concept.

The following chapter describes the technologies used in this work, such as Yum and Apt as instances of the package management tools, and the Spacewalk server. The description is not to be exhaustive, it only focuses on the features related to the work. Chapter 3 analyses the source code of the Apt in a detailed manner, including a mechanism of the content download and a repository management. The end of each section proposes a concept of solution. Chapter 4 introduces Apt-Spacewalk package and discusses the implementation details. Miscellaneous changes needed in existing projects are isolated in Appendices A, B and C.

Chapter 2

Related technologies

The technologies related to the work can be diveded into the three parts: the package management tools, the Spacewalk server, and the adapter enabling their communication. In the GNU/Linux distributions, there is a huge variety of package management solutions however the work focuses only on two specific ones: Yum and Apt. The Yum is a package updater for the RPM based distributions such as Fedora and Red Hat Enterprise Linux, while The Apt is designed for Debian packages.

The Spacewalk server is a system management system with an API[1] and a web interface. One of the Spacewalk's key features is the ability to store and manage packages and serve them as a repository to the *entitled* systems. Spacewalk has a large scale of capabilities beyond that, however this work focuses only on package and system management.

The last section of this chapter describes the adapter between Yum and Spacewalk server (`yum-rhn-plugin`), which is used by client system to download packages as well as metadata from Spacewalk. The adapter is implemented as an Yum plug-in, it authenticates with Spacewalk and forwards the bits from Spacewalk to the Yum.

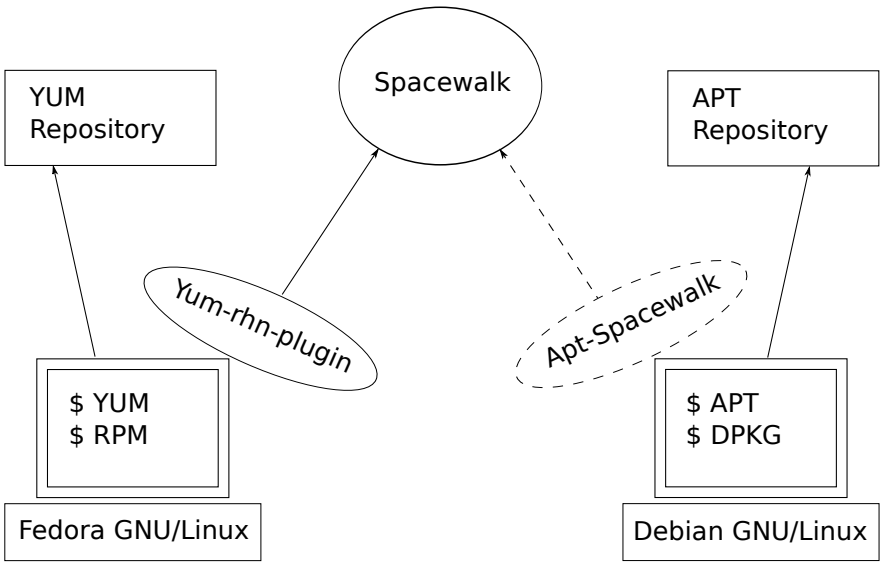


Figure 2.1: The technologies related to the work and the given aim—Apt-Spacewalk

[1] Application Programming Interface

2.1 Package Management Tools

In the GNU/Linux distrubutions, whole content is delivered within packages. There is no single package format. In fact, whole families of distributions are build around of each of package formats. The best known package formats are rpm, deb, tgz, pkg.tar.xz, and ebuild.

Besides the purposeful content, each package also contains a meta information. In spite of different formats, fundamental part of the meta information is present in every package. It applies to name, version, architecture, author, license, and dependencies[2]. Frequently, the three tier architecture of package management tool can be encountered.

1. Package Manager

   Package manager is the lowest layer of the package management system. It handles the extraction of bare packages, their installation, update, and removal. The tool usually understands dependencies and computes execution plan, however it will not proceed if some of the requirements is not full filled. This layer also tracks installed packages (instances: rpm, dpkg).

2. Package Updater

   Package updater manages software repositories, handles the content download, resolves inter-package dependencies and conflicts, and designs an execution plan (instances: Yum, Apt).

3. Graphical front-end

   Even though this layer is not essential it is often present and used by the desktop users (instances: Synaptic, PackageManager)

2.1.1 Yum Tool

Yum (The Yellowdog Updater, Modified) is the open source command line utility for software installation, update, and removal [3]. Yum is written in Python. It is modular, highly configurable and provides the interface for a plug-in development. A lot of additional functionalities are distributed over the `yum-utils` and Yum plug-ins. The `yum-utils` package contains collection of utilities and examples that make yum easier and more powerful to use [4].

As a backend for low-level package handling, Yum issues RPM Package Manager [5]. Unlike the backend, Yum determines inter-package dependencies (and conflicts) and automatically computes an execution plan for transactions. Software consumed by the transaction is acquired after the planning phase from the sources called repositories.

The Repository

Yum repository is a storage of defined structure containing packages and its indexes [6]. Indexes are known as repodata, named after the main index file (`repomd.xml`) repository metadata. Yum uses the indexes for accessing various information about RPM packages[3].

[2] With the exception of Slackware GNU/Linux packages (tar-balls) which do not track inter-package dependencies.

[3] Not only about single packages. Optional `comps` file holds the information regarding whole package sets.

Considering the fact that repository can be not only local but remote, the size of repodata and the number of retrieves should be minimal. That is also one of the reasons why structure of repository indexes has changed over time. Yum retrieves `repomd.xml` any time the cache timeout has been hit. The rest of indexes is updated on demand, when the checksum of the cached one and the remote one differs.

Yum repositories are configured in plain text files stored in `/etc/yum.repos.d/`. However, Yum provides an interface for a dynamic repository configuration.

The Plug-in Interface

Yum has a simple but powerful plug-in interface which allows external modules to add new features and/or modify Yum behavior [7]. Plug-ins are Python modules which are loaded at start and Yum invokes the plugin code in dozen of specified points so-called hooks. Plug-in is then supplied with the `conduit` interface to be able to modify the internal structures of the Yum.

2.1.2 Advanced Packaging Tool

The Apt (The Advanced Packaging Tool) is an open source set of libraries and user space tools for a package installation and removal [8]. The Apt has been designed to make use of dpkg[4] and provides further operations beyond it, such as dependency resolution, repository management, and package retrieval.

The Apt suite contains various executables, each for separate set of actions, for instance there is apt-cache binary for accessing the cached information. The common actions across these exacutables are provided by shared libraries: `libapt-pkg.so` and `libapt-inst.so`.

The apt-pkg library, as the name suggests, focuses mainly on packages. It contains mixture of classes: for acquiring a content, accessing a cache, the default resolver for a dependencies, and other aspects as work with checksums and a pinnig interface.

The apt-inst library is notably smaller and it provides only a few classes for handling file archives. According to the Wikipedia [10], the apt-inst project was meant to replace dpkg, but the project did not succeded. During a further development, a few functions from the library have been marked as obsolete. Both libraries are applied by an amount[5] of package management tools distributed with the Debian GNU/Linux.

The Configuration

The Apt configuration is stored in plain text files within the `/etc/apt/` directory. There are three sections of the configuration: repository definitions (`sources.list`), apt configuration (`apt.conf`), and pining settings (`preferences`).

Each section can be stored either in the file or in the directory of the same name with the `.d` extension (e.g. `/etc/apt/sources.list.d/`). Such a system of configuration can be also seen in other subsystems such as `/etc/modprobe.d/` or `/etc/sudoers.d/`. The files within the directory usually start with one or two digits indicating the priority of the file. Having the multiple files for each section eases the manipulation with the configurations for the third-party applications.

The configuration section (`apt.conf`) is organized in a tree of options [11]. Where each node of the tree represents a single option which can be assigned with a value. The node is

[4] Package management system delivered with the Debian GNU/Linux distribution and its derivatives [9].
[5] Aptitude, Dselect, Synaptic, Adept, Update notifier, Ubuntu Software Center, KPackage and others

described by a full path from a root node (the root itself is excluded from a path). The nodes on the path are separated by a double colon notation. E.g. `APT::Get::Assume-Yes` is the option within the Apt and the Get nodes. Options do not inherit anything from their parent.

The Repository

Likewise the Yum repository, the Apt repository consists of the packages and its indexes. However, the structure of Apt repository is more sophisticated in the way that it can hold multiple distributions and multiple components for the distribution.

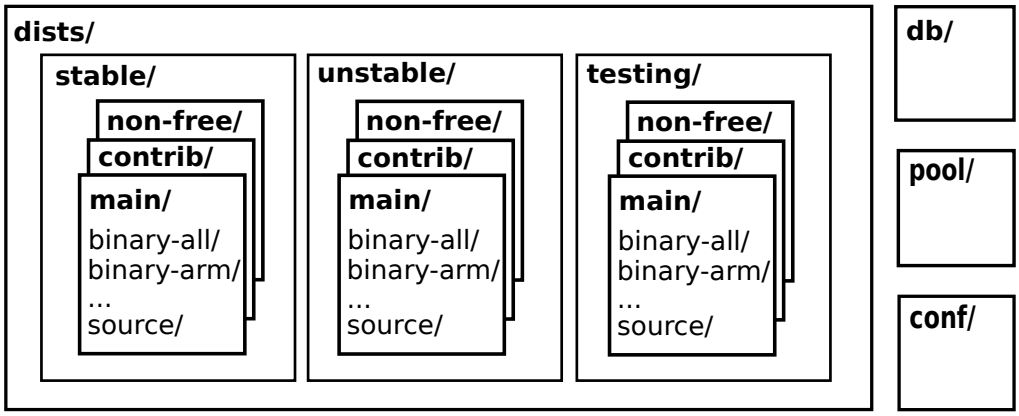


Figure 2.2: The structure of the Apt repository [12].

Figure 2.2 represents the standard directory tree of the Apt repository. In the `dists/` directory, there are either names of available distributions, distribution keywords (`stable`, `unstable`, `experimental`, `testing`), or the combination of both.

Within the subdirectories there is a `Release` file containing a list of metadata checksums for each of the component/architecture pair. The `Release` file is not essential for the Apt's workflow, though it is useful for verifying metadata authenticity. Metadata for packages (files named `Packages` and `Sources`) rest on the bottom of the directory tree. The packages are located in the separate directory (usually the `pool/`) and they are referenced from the metadata.

As illustrated on the figure, distributions and components are inseparable part of repository structure. In the other words, the Apt is able to assemble the URI[6] location based on distribution, configured components and machine's architecture. The repository definitions are stored in the `sources.list` in the following form [14]:

```
type protocol://provider/path distribution component1 component2
```

The type is either `deb` for installable packages or `deb-src` for Debian source packages. When updating, the Apt retrieves a metadata file for each of listed components according to the type—either `Packages` or `Sources`. The triplet of `protocol`, `provider`, `path` denotes a standard URI of repository root, the only constraint is that appropriate *acquire method* must exist for the given protocol.

[6] Uniform Resource Identifier, RFC 3986 [13].

The Interfaces for Plug-ins

Despite the fact that the Apt suite does not have a concept of plug-ins, there is quite a lot of third party applications co-existing with the Apt. These applications can either utilize the shared libraries of the Apt, register a new acquire methods, or make use of `Invoke` hook of the Apt.

The Acquire Methods

The Apt supports multiple network protocols: HTTP[7], FTP, FILE, RSH and more. Each one of the supported protocols is implemented as a separate binary which is executed by the Apt once the protocol is needed. These binaries are called *acquire methods* and they are located in the `/var/lib/apt/methods/` directory. The Apt chooses the method according to a `protocol` field from the `sources.list`.

The Apt communicates with the method through a pipe by an *Acquire Protocol*. The Acquire Protocol is text-based and it resembles HTTP. The protocol is described in a detail in Section 3.1.1. By introducing a new acquire method to the Apt, a plug-in can modify the behavior of the Apt on the network (for instance introduce a support for a new protocol to the Apt).

The Invoke Hooks

Another option of customizing the Apt's behavior is to use invoke hooks. The invoke hook is a point of a program execution where a snipet of another program might be executed. The Apt offers about 7 hooks on various occasions, details are described in the Table 2.1.

Name	Occasion
Apt::Update::Auth-Failure	When either Release file or a signature is missing
Apt::Update::Pre-Invoke	Before updating a cache (apt-get update)
Apt::Update::Post-Invoke	After the update of a cache
Apt::Update::Post-Invoke-Success	After successfull update of a cache
DPkg::Pre-Invoke	Before invoking dpkg
DPkg::Post-Invoke	After invoking dpkg
DPkg::Pre-Install-Pkgs	Before an installation of packages

Table 2.1: Apt's Invoke Hooks

The Apt hooks can be configured via the `apt.conf`. For example, the following line is to make an entry in the system log, with each attempt to update the Apt cache.

```
Apt::Update::Pre-Invoke { /usr/bin/logger 'The apt-get update'; };
```

The Python Bindings

The Apt's interface is also available from within a Python code. There is a `python-apt` package providing access to almost every functionality supported by the underlying `apt-pkg` and `apt-inst` libraries [16]. Besides the direct bindings, `python-apt` presents its own derivated objects for high level operations.

[7] Hyper Text Transfer Protocol, RFC 2616 [15].

We have to note here that Python is not the only one language with the binding to Apt libraries. For example `libapt-pkg-perl` package makes the library available from Perl programming language.

2.2 Spacewalk

A Spacewalk is a complex, web based, systems managing system, designed for GNU/Linux clients with the RPM package manager [17]. Besides the systems, Spacewalk is able to manage packages, errata, package channels, and custom non-rpm content. Spacewalk can administer client system from a source provisioning[8] through a while life cycle. Spacewalk is able to establish remote commands, remote package management, monitoring, and many other.

This Section briefly describes history, architecture and deployment model of Spacewalk. To understand the details of architecture, one might find helpful to learn the history and the purpose first. Spacewalk features and capabilities are not described in a detail[9] with the exception of *Software Channels* and Debian support which are essential for the work.

2.2.1 A Short History of the Spacewalk

The project was started in June 17, 2008 by opensourcing Red Hat Network Satellite [18]. However its genesis dates back to 2000, when the *Red Hat Network* was openned up for public [19]. Since then, Red Hat Network, abbreviated to *RHN* and also known as a Hosted, serves as a publicly available mechanism for maintaing Red Hat Enterprise Linux machines. Besides of the updates and the distributin of installation media, the Hosted allows customers to manage their systems and subscriptions.

Centralized architecture of Hosted used for a package delivery leads to a high load of Hosted and high Internet bandwith usage at customers with multiple subscribtions. To address the problem Red Hat Network Proxy has been relased. It is deployed at customers where it caches the packages to speed up a secured download [20]. The Proxy can also be an assistant for building separate network segments.

Although the solution was suboptimal. Management capabilities were limited by design and the content delivery system was highly dependent on the Internet access. Such dependency was not acceptable for local area neworks disconnected from the Internet, e.g. because of the security concerns.

In February 2002, Red Hat released Red Hat Network Satellite, providing users much greater functionality and allowing customization [20]. The first releases of the RHN Satellite was based om the codebase of the Hosted. Later then, developement of the Hosted and the RHN Satellite has dispart and each team has concentrated on a different set of features. By opensourcing RHN Satellite, Spacewalk project was brought to life and it became an upstream of the next releases of RHN Satellite (the 5.3.0 and higher).

2.2.2 Spacewalk Deployment Model

Figure 2.3 shows the comparison of the deployment models focusing on the management and the capabilities of the content delivery. Customer A uses a direct connection to the provider while customer B has the management system (Spacewalk in this case) deployed in his

[8] The automated installation of the operating system

[9] Spacewalk features and capabilities are well documented on the project wikipage [17].

private network[21]. Having all the machinery in the private network gives the customer B extra abilities like monitoring, which might not be doable through the wide area network (the Internet in this case). In addition, customer B can be separeted completelly from outer world, which might be needed for high security deployments.

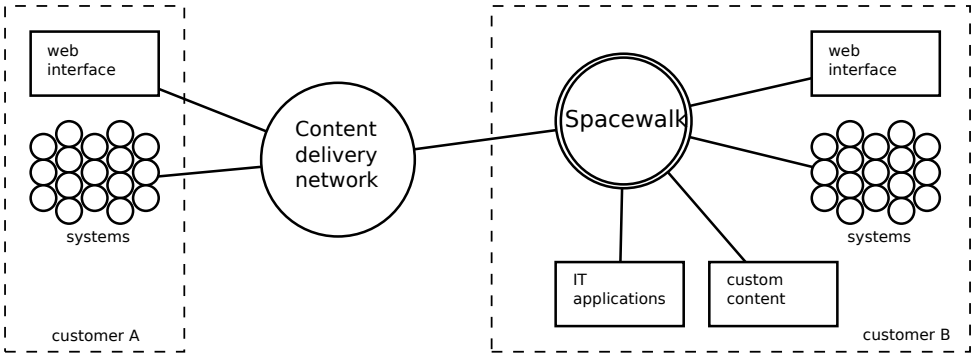


Figure 2.3: The comparison of the deployment models [21].

2.2.3 Server Architecture

Spacewalk consists of the server code deployed on the Apache HTTP server, the backend tools, the deamons, and the database [22]. The most of the Spacewalk's functionalities are available through the Apache handlers (including the web interface), although the `root` access to the underlying machine might be needed for some of the management actions. E.g. whole stack can be turned on and off by `rhn-satellite` command-line tool.

HTTP Server

The code deployed on the Apache HTTP server compound of several programming languages, each of stacks has separate responsibilities. There is a Python code, which handles the communication with the clients, a Perl code serving a minority of the Web pages [23], and finally a Java rendering most of the Web pages and providing frontend API calls. The Apache handlers communicates outwards by `XML-RPC`[10], `HTTP` and `HTTPS` protocols.

The client systems invoke the remote procedure calls (`XML-RPC`), which are dispatched by Python backend stack [25]. This model is used for most of the client actions, the example might be the registration of a new client which involves many of the backend calls. However, the `XML-RPC` is not the only mean of the client communication, clients may use the `HTTPS` for the package download, `jabber` for real-time messaging, `ssh` for the monitoring or `tftp` in the very beginning of the client's life cycle[11].

The Perl and the Java parts form the Spacewalk's web user interface, exposing the most of the Spacewalk's management capabilities. The Java also serves the frontend `XML-RPC` interface, providing a considerable part of web's management capabilities. The custom IT applications (on Figure 2.3) are utilizing just the frontend API. The long-running actions

[10]The protocol for remote procedure calls encoded in XML [24].

[11]The `tftp` protocol is used during the network boot of a client, when the system is being provisioned (`Kickstarted`).

(scheduled eiher on a Web or XML-RPC interface) are inserted into the queue and executed asynchronously by the Taskomatic deamon [26].

Deamons

Beside the Database, Apache, and Tomcat, the Spacewalk runs various deamon services. The already mentioned one is the Taskomatic which is responsible for scheduling asynchronous actions. The RHN-Search deamon covers the indexing and search of the database.

The following daemons are not essential, but they are used for the additional and support communication with the client systems. The Jabberd daemon provides the jabber server and the Osa-dispatcher uses it for talking to clients in real-time. The Cobbler is used for the provissioning and interoperates with Spacewalk through the database, XML-RPC, and JSON API. Finally, the Monitoring Deamon as the name suggests covers the monitoring of the clients.

Backend tools

A backend tools are command-line tools on the Spacewalk server providing additional management functionality to the Spacewalk administrator. These tools can manipulate a database, export and import software channels, generate various reports, or upgrade the Spacewalk.

Database

All the data stored by Spacewalk server are integrated in a single database instance. The Spacewalk server supports two database backends: Oracle and PostgreSQL. Traditionally, Spacewalk runs on Oracle database, but since June 2008, there have been an effort to support the PostgreSQL as well [27]. At present, PostgreSQL port is considered to be functional [28].

The information about PostgreSQL database backend might be important for this work, because the repository of Debian distribution does not fit to the Oracle 10g Express Edition, which has a limited tablespace to 4 GiB. With the PostgreSQL backend there is no such limitation.

2.2.4 Concept of Software Channels

Packages managed by Spacewalk server are grouped to the sets called software channels. Channels are associated to the clients and their purpose is to define the content intended for the client. A software channel can be either a base or a child. The client system can be associated with one base channel and a number of its child channels.

From the client perspective, the channel is very similar to any other software repository. Thus, Spacewalk has to provide equivalent repository metadata[12] for each of managed channels. However, the difference between channels and standard repositories is a client authorization. Spacewalk does not serve the content, unless the client is authenticated and authorized to consume it. Each client request is evaluated respectively considering the client system, the channel and the package identifier.

[12]The process of metadata generation is is described in the Appendix A.1.

2.2.5 Initial State of Debian Support

Lukáš Ďurfina in his Master thesis [2] introduced Debian support to Spacewalk. In order to register and manage Debian clients, both sides, server and client, have been modified. Since then, Spacewalk is able to manage Debian binary packages and Debian channels. Debian clients can be registered with Spacewalk using activation keys and a basic set of remote actions can be executed. The package managements capabilities has not been fully completed.

Server side changes have been merged with upstream project on spring 2010. However, the changes in the client code as well as packaging information were not published. Ďurfina has only provided built `.deb` packages. The builts were based on the client code from December 2009. The rebase of one of the core client packages (`rhn-client-tools`) is described in the Appendix B.

2.3 Yum RHN Plug-in

A package yum-rhn-plugin is an adapter between Yum and Spacewalk. If the client system is registered with the Spacewalk, yum-rhn-plugin activates and forwards the communication between Yum and Spacewalk (Figure 2.1). The plug-in is written in Python and it uses `rhn-client-tools` library for high-level operations with Spacewalk server. In the plug-in workflow, one can find three key responsibilities:

- Transform Spacewalk's software channels to Yum's repositories (Section 2.3.1).
- Attach authentication when Yum make a use of these repositories (Section 2.3.2).
- Refresh *Package profile* after each transaction (Section 2.3.3).

2.3.1 Repository Management

When initiated, yum-rhn-plugin authenticates to Spacewalk and gathers the information about entitled software channels. For each channel, plug-in creates `RhnRepo` object which is then forwarded to the Yum as a `YumRepository`. By overriding YumRepository object, plug-in slightly modifies inner structures and behavior of the Yum.

2.3.2 Authentication

There are two types of client authentication, one for a remote procedure calls and another for `HTTPS` requests. The former is established with credentials from a `systemid` file and settles a transient token for a subsequent `HTTPS` communication. The remote procedure calls are encapsulated in `rhn-client-tools` library and bonded by `XML-RPC`. protocol.

The transient token also refered as *loginInfo* is a structure generated by server. It contains key-value pairs. One of the items is *X-RHN-Auth*, which is an MD5 digest generated from all the other items and the server secret salt. The *X-RHN-Auth* is a signature which verifies the authenticity of the client [29].

The mechanism not unlike *Basic Authentication* [30] is used for `HTTPS` connections to Spacewalk. The *loginInfo* attached to HTTPS request is called *Authentication headers*. The following is the example of HTTPS request amended with the *Authentication Headers*. Especially note the URI (line 7) which points out the location of the channel metadata on the Spacewalk server.

```
wget --header='X-RHN-Server-Id: 1000010298' \
     --header='X-RHN-Auth-Server-Time: 1285423201.81' \
     --header='X-RHN-Auth: RhIkjPZRjUjeIxN2zoEMEw==' \
     --header='X-RHN-Transport-Capability: follow-redirects=3' \
     --header='X-RHN-Auth-User-Id: ' \
     --header='X-RHN-Auth-Expire-Offset: 3600.0' \
     https://example.com/XMLRPC/GET-REQ/epel-i386-5/repodata/repomd.xml
```

2.3.3 Package profile Update

For each client Spacewalk tracks the list (*Package profile*) of installed packages with versions and installation dates. Obviously the list of packages changes with each Yum transaction and therefore needs to be refreshed on Spacewalk side. The yum-rhn-pligin binds to the `posttrans_hook` of the Yum, and at that time the information about installed packages is sent to the Spacewalk.

2.3.4 Yum-rhn-plugin's Workflow

The sequence diagram in Figure 2.4 illustrates the signals sent between Yum and Spacewalk being transformed by Yum-rhn-plugin. Yum communicates with the plug-in by methods invocation, which gets transformed either to XML-RPC call or HTTPS GET request.

At the very beginning, Yum asks plug-ins for a list of available repositories by invoking the `prereposetup_hook`. As a result, the plug-in logs in with the Spacewalk server to access the `loginInfo` structure containing the Authentication headers. In the next query, Spacewalk returns a list of entitled software channels and the plug-in composes an `RhnRepo` object per each of software channels. The `RhnRepo` class is specialization of `YumRepository`, overriding the `get()` method to handle the authenticated download.

Yum calls the `get()` method of a particular repository object whenever it needs to acquire content, for instance the index file `repomd.xml`. The `RhnRepo.get()` compiles the HTTPS request attaching the authentication headers from the `loginInfo`. Spacewalk back-end tier validates the headers and only authorize the request if the client system is entitled to consume the content. As a result, Yum is supplied with the desired content.

When the transaction completes, Yum invokes plug-in again through `posttrans_hook`. Within the hook plug-in refreshes the package profile on the Spacewalk server through the XML-RPC call, using support functions from `rhn-client-tools` library.

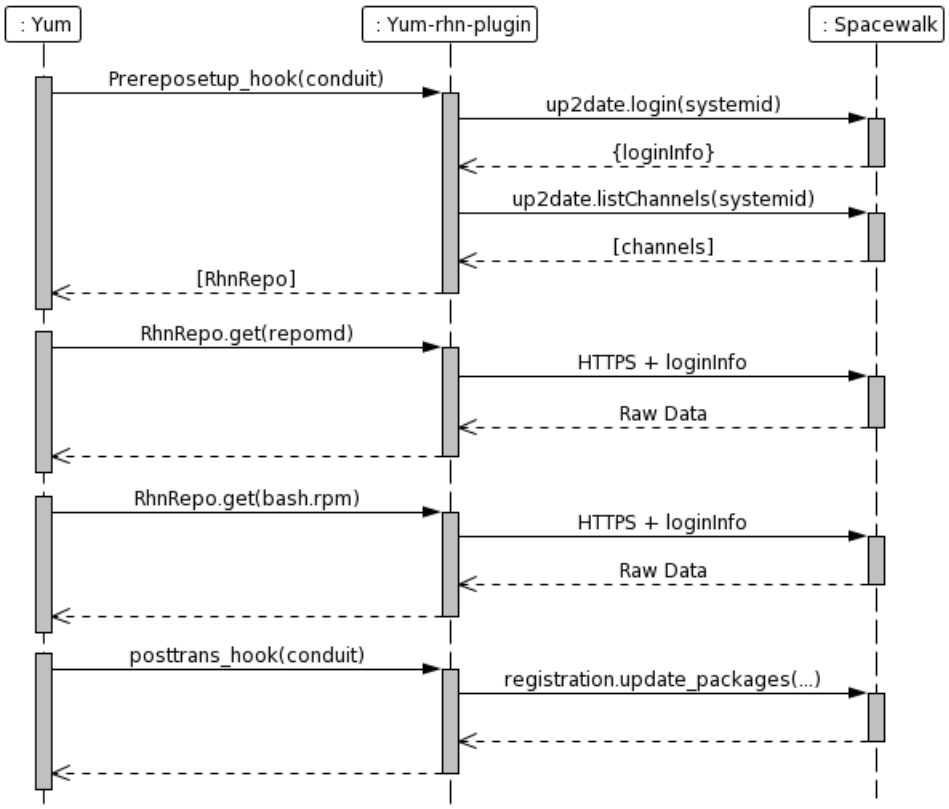


Figure 2.4: The communication betwen yum, yum-rhn-plugin and Spacewalk backend tier during `yum update` command (an update of specific packages).

Chapter 3

Analysis of The Apt Code

The goal of this work is to provide an adapter similar to yum-rhn-plugin for the Apt. Such adapter should not be bound to any specific Apt frontend like apt-get, aptitude or synaptic. Ideally all the frontends will use the adapter in a way transparent for the user. Such requirement implies the adapter operating fully-enclosed in the Apt primitives. This chapter describes parts of the Apt architecture related to the adapter's workflow.

3.1 Content Acquisition

This section describes a mechanism used by the Apt to fetch a content from the repositories. The mechanism is called Acquire and the interface is available in libapt-pkg[1]. To understand the mechanism, see the following Python script, which is to fetch the Release file from a Debian repository.

```
1 from apt import apt_pkg
2 acquire = apt_pkg.Acquire()
3 acquireFile = apt_pkg.AcquireFile(acquire,
4     'http://ftp.cz.debian.org/debian/dists/unstable/Release')
5 acquire.run()
```

The first line includes the apt_pkg module, which is a low-level binding for libapt-pkg library. The Acquire object (line 2) represents a queue and a scheduler for download. To optimize parallel download the Acquire object contains multiple discretive queues served in parallel. Each item registered to download is classified to the proper queue according to the URI.

There are two modes of classification: *access* and *host*. The access mode creates named queues based on the protocol identifier from URI. The host mode considers both, the protocol and the domain name. Either way the URIs with a different protocol are grouped to different queues. User is allowed to configure the mode.

Each file registered to the download is represented by *AcquireItem* class or its specialization, e.g. AcqureFile. The constructor of AcquireFile (on the line 3) will automatically register with the Acquire object appending the URI location to one of the upper mentioned queues.

[1]The name scheme in Python module differs from the one used in the shared library. This work prefers the Python variant, which is more human-readable.

When the Acquire is being run (line 5), it creates *AcquireWorker* object for each queue. The AcquireWorker is responsible for the download of a given queue. Whereas, the *AcquireMethod* is responsible for a download of a single item. The AcquireWorker executes the acquire method as a subprocess. The inter-process communication between the worker and the method is covered in the *Acquire Protocol* and transfered via UNIX pipe. The implementation of the protocol is encapsulated in AcquireWorker class (for a master end) and AcquireMethod class.

All the methods in the Apt suite are separate executables with the common behavior inherited from the AcquireMethod. Each one of the specializations implements only the `Fetch` function used for downloading a single URI.

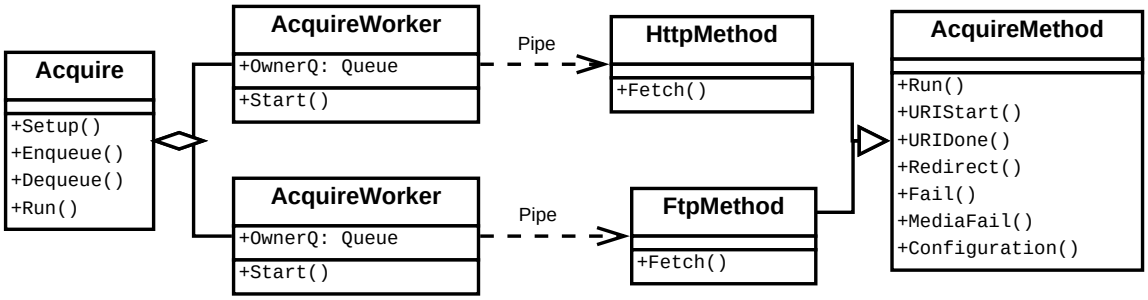


Figure 3.1: The class diagram ilustrates the communication between Worker and Method.

3.1.1 Acquire Protocol

The text protocol between AcquireWorker and AcquireMethod can be sniffed by interleaving the pipe chain with a tapping program. For instance, the HTTP method can be replaced with the folowing script which will be activated by an attempt to download a content through HTTP.

```
1 #!/bin/bash
2 tee -a methodlog.in \
3   | /usr/lib/apt/methods/http.distrib \
4   | tee -a methodlog.out
```

The captured log of communication[2] resembles the HTTP protocol. Likewise the HTTP *response*[3] the messages in the log start with *Status-Line* folowed by the headers. Status-Line consists of the numerical identifier known as *Status-Code* and the *Reason-Phrase*. Unlike the HTTP response, the acquire protocol message does not contain a message body. Raw data are stored directly on a disk and the protocol transmits only filenames.

Table 3.1 gives a detail of the message types within the protocol. The message types marked with a symbol * are esential for a successfull download, however different methods may use a different set of messages.

At the start, each method prints the *100 Capabilities* message, declaring its characteristics and eventually requesting the *601 Configuration* message from the worker. After the capabilities and configuration is exchanged, worker may issue the download by *600 Acquire* message. Such a request must include URI location and filename of destination

[2]Example can be found in Appendix D.
[3]HTTP terminology is defined in RFC 2616 [15].

object. While fetching the URI, method notifies the master about progress with 1, 2 or 4 houndred messages. These messages always consists of URI regarding the initial request. Methods with a *pipeline* capability may dispatch multiple URIs at once.

Code	Phrase	Send by	Brief description
100	Capabilities	Method *	Declare method's characteristics
101	Log	Method	Arbitrary or debug information
102	Status	Method	Report details of progress
103	Redirect	Method	Download has been redirected
200	URI Start	Method	Download has started
201	URI Done	Method *	Download has been successfully completed
400	URI Failure	Method	Download has failed
401	Failure	Method	General failure
403	Media Failure	Method	Need to change physical media (CD/DVD)
600	Acquire	Worker *	Request for download
601	Configuration	Worker	Declare worker's configuration
603	Media changed	Worker	Media has been changed upon request

Table 3.1: Message types of Acquire protocol.

3.1.2 Spacewalk Acquire Method

From the Spacewalk perspective, each HTTPS request must be amend with authentication headers, which can be achieved by a special acquire method. Call the method *spacewalk*. The method communicates with the Apt through upper mentioned protocol and transfers the requests to the Spacewalk server.

Since all the well known Apt frontends use apt-pkg for the content download, introduction of a new method will be transparent to them. Apt-pkg library will choose the Spacewalk Acquire Method for any URI, which has the protocol equal to *spacewalk*

3.2 Repository Management

As described in Section 2.1.2, the Apt repositories are configured statically in a text file and Apt caches metadata in the `/var/cache/apt/` directory. Considering the fact that Apt-get has a special command for downloading repository metadata to the cache, one can presume that the metadata will not change very often.

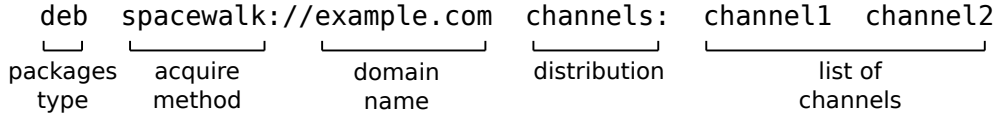
For instance if one runs `apt-get install tcsh` command, the Apt looks up for which of repositories configured in the sources.list is the cache available. The Apt attempts to install the package, only if it is present in the cache. Apt-get will neither update the cache automatically nor challange any plug-in to adjust the sources.list in the run-time.

The Apt lacks for the concept of dynamic software repositories, which is in contrast with the typical Spacewalk use cases. On Spacewalk web user interface, additional repositories might be associated to the client system at any time.

3.2.1 Updating the Cache

Even thought the metadata of the Apt repository are spread into the multiple files, the most important one is a `Packages` file which holds the information about all the packages for given

repository. When updating the cache, the Apt downloads `Packages` for each configured component in the sources.list. Which implies the fact that multiple repositories associated to the client in Spacewalk can be configured to the Apt as components. The following is a hyphotetical line in the sources.list of the client registered to Spacewalk (example.com) associated with two channels: channel1 and channel2.



Keyword *deb* tells us that repository contains binary packages. Second item specifies Spacewalk's domain name and the acquire method which is to be used. The third item originally represents the distribution and the item is mandatory, but Spacewalk does not have the concept of ditributions similar to Debian. We decided to use word *channels:* as a human readable keyword, which does not have any other meaning for the communication. The rest of the line is the listing of associated software channels.

3.2.2 URI Translations

When downloading the metadata, the Apt automatically composes URI for a given line in the sources.list. The exemplary line from Section 3.2.1 will result in the following URIs.

```
1 spacewalk://example.com/dists/channels:/channel1/binary-i386/Packages.gz
2 spacewalk://example.com/dists/channels:/channel2/binary-i386/Packages.gz
```

However, the Spacewalk server provides the content under the different locations.

```
1 https://example.com/XMLRPC/GET-REQ/channel1/Packages.gz
2 https://example.com/XMLRPC/GET-REQ/channel2/Packages.gz
```

On the server side, there is a Python handler (`/XMLRPC/GET-REQ/`) which verifies the authentication and serves the content only to the authorized clients. The URI inconsistence could be redeemed by a proper configuration on Server side[4] or even better on the client side by Spacewalk Acquire Method.

3.2.3 Dynamic Repositories

A dynamic repository is a repository, which can be configured under certain conditions in the run-time, e.g. by some sort of plug-in. In contrast to Yum, Apt does not support the dynamic repositories. The Apt stores the configuration in the plain-text file (`sources.list`). The problem is that the `sources.list` is often read by third-party programs. These programs use either primitives in apt-pkg library or they parse the `sources.list` on their own. Neither the Apt nor the third-party applications do expect repositories to change dynamically.

[4] For instance by Apache's `mod_rewrite`.

Possible Solutions

1. Modify sources.list synchronously.

   (a) Dynamic repository feature could be implemented in a synchronous action, which will activate any time apt-pkg is asked to read sources.list. The problem with this approach is that sources.list are frequently read by non-root processes (as M. Vogt from debian.org wrote on deity Debian mainling list cf. [31]). Additionally, even when the new repository is successfully added, the Apt would most probably not have the repodata for it.

   (b) A step better solution would be to bind with the action which updates the repository cache of the Apt. In that case, the sources.list and the repository cache will be consistent. The update of cache is much less frequent compared to reading the sources.list and it is always executed by a process with a `root` priviledges.

   (c) Another option would be not to modify sources.list at all but extend the procedure which reads it. Such procedure is available in apt-pkg library. However, this solution will not be transparent for users and other tools parsing the sources.list on their own.

2. Modify sources.list asynchronously.

   Either a cronjob or a deamon can be set up. Spacewalk has multiple client deamons which can be adjusted to update the sources.list timely. The deamon rhnsd polls every 4 hours with Spacewalk to pick up remote actions. Osad is the deamon which talks to Spacewalk in a real time on jabber protocol. Finally, the rhncfg deamon deploys configuration files. Unfortunatelly, an for asynchronous action there is no guarantee that it will be run. The problem with the deamons is that they are not necessarily up and running. Consider also the pooling interval, which is 4 hours by default, it may fail or get postponed by another 4 hours.

3. Conceal multiple repositories from the Apt.

   When the cache is being updated particular acquire methods are asked to download the metadata. The acquire method could query Spacewalk for configured repositories, download metadata for each, concatenate them to the single file and forward it to the Apt.

   However, this will not be transparent for the user, who will never know from which exact repository the new package comes from. In addition, the pinning feature would not be used in between single Spacewalk repositories.

4. Modify the Apt to allow dynamic repositories.

   A quite intrusive change would be to introduce a new type of repository defininition in the sources.list. Such change would require a big change set for the Apt and also some of the third-parties.

The Proposal

After the discussion on the Apt mailing list [31], we have decided to take the option 1b and modify the Apt to allow plug-ins to alter the sources.list prior to the cache update. In other words, any time when user issues `apt-get update` (or an equivalent) sources.list will be adjusted.

The Apt already has the *APT::Update::Pre-Invoke* hook which is run before the cache is updated. Plug-ins can utilize the hook and amend the source.list, however the Apt reads the sources.list prior to the hook and does not reload it after then. Meaning that the cache is populated by the repositories defined in the sources.list prior the Invoke hook. Proposed change is to force the Apt to reload the sources.list after the hook and before the cache update.

Anyway, the change must not affect only the Apt itself, alas it should be transparent to all the frontends. In the apt-pkg, there is a ListUpdate function, which is the top level routine for updating the cache. Brief look on the code of Apt-get, Aptitude and Synaptic proofs that they use the function in order to update the Apt's cache. The desired behavior can be enforced by a trivial patch (Appendix C).

Chapter 4

Implementation and Automated Tests of Apt-Spacewalk

This chapter describes the implementation of Apt-Spacewalk and the creation of the automated tests. The Apt-Spacewalk is a small adapter forwarding the communication between the Apt and the Spacewalk server. Therefore, the testing section focus on the communication between the components and the package management life-cycle.

4.1 Implementation of Apt-Spacewalk

Apt-Spacewalk—the plug-in for the Apt should be a standalone Debian package. The plug-in will comprise own acquire method for the communication with Spacewalk and two scripts: first one to update the sources.list and another for refreshing the package profile.

4.1.1 Spacewalk Acquire Method

The Spacewalk acquire method is an executable in `/usr/lib/apt/methods` directory. It communicates with the Apt on standard input/output and with Spacewalk through HTTPS and XML-RPC protocols. The communication has been already implemented by third-parties in Apt (C++) and rhn-client-tools (Python) packages.

The spacewalk acquire method is written in Python programming language and utilizes rhn-client-tools library for all the communication with Spacewalk. Apparently, the inter-process communication with the Apt is rewritten from C++ to Python.

Design of the Solution

Design of a proposed solution is similar to all the other acquire methods. Base class `pkg_acquire_method` covers the communication with the Apt. While its specialization `spacewalk_method` implements the `fetch()` method, which is responsible for the download. The class `pkg_acquire_method` (Figure 4.1) is inspired by `AcquireMethod` (Figure 3.1) from the apt-pkg library, but it contains only the subset of functionality which is usefull for Spacewalk.

The interaction with Spacewalk comprise several actions and most of them are required to be run only once even for multiple requested URIs. We have to point here that the most of these actions are similar to the yum-rhn-plugin workflow. In order to fetch the content, client must log-in, look-up for associated channels, preparare the authenticated headers,

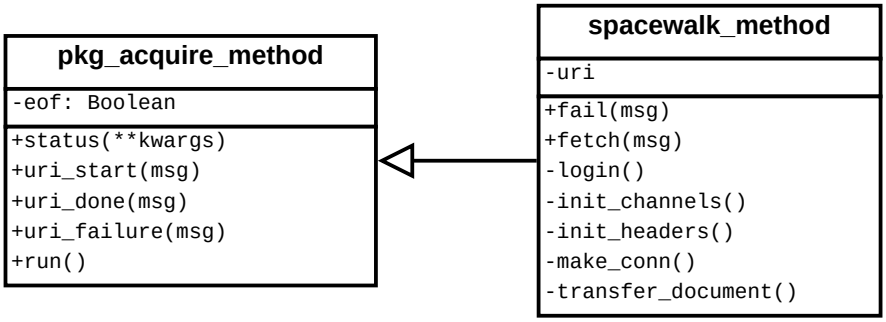


Figure 4.1: Class diagram of Spacewalk Acquire Method.

verify the HTTPS certificate, and establish a keep-alive connection. The mentioned actions are good candidates to be be extracted to the private methods.

Python Specifics

We have to note here that the Python 2.x bufferes the operations on the standard output and the `print` command is not guaranteed to be implemented as a single syscall[1]. On the other hand, parser on the Apt side (`acquire-worker`) requires the messages to be delivered timely and in one piece. The acquire-worker expects a whole message to be read by a single syscall.

Futher investigation of Python 2.x implementation reveals that command `print 'Hello'` requires two syscalls, first one for the string 'Hello' and second one for a new line. Fortunatelly, the same output can be generated using only one syscall. The first possibility is to use `os.write()` function, which is a direct binding to the syscall. Or another possibility is to use print command ended by a comma (`print 'Hello\n',`). The comma forces Python to not append the new line. In the final solution, We prefered the later, because of human redability.

4.1.2 Pre Invoke

As discussed in Section 3.2.3, The Apt-Spacewalk will bind to *APT::Update::Pre-Invoke* hook and adjust the configuration in the sources.list. It logs in with Spacewalk and retrieves the list of channels the client is subscribed to. The result is written to the separate file `/etc/apt/sources.list.d/spacewalk.list`. The file must remain in the correct syntax even if the client gets unsubscribed.

The Pre-Invoke script utilizes apt-pkg library for parsing and modifying the sources.list and rhn-client-tools library for communication with Spacewalk.

4.1.3 Post Invoke

Post Invoke is the simpliest part of the Apt-Spacewalk. It does only a part of the tool `rhn-profile-sync` does. It collects the list of currently installed packages and pass them forward to the Spacewalk by a single XML-RPC call.

[1]System call `write` sends the data to the buffer using primitives of underlying Operating System [32]. In our case the buffer is the standard output (`stdout`).

4.2 Automated Tests

The automated tests are considered to be the essential part of every software project. This section will describe the creation of tests for Spacewalk's Debian client. It discusses the possibilities of creating the test for multiple components in heterogenous environments, optimizing the run-time/coverage ratio. The end of section introduces a functional test which covers a basic package management scenario of Debian client.

4.2.1 Choosing the Test Technique

Given the work introduced small changes in numerous components, the tests should cover all of them, including their communication and interoperability. The changes affected the following packages: apt-spacewalk, apt, spacewalk-java, spacewalk-backend, and rhn-client-tools.

Functional vs. Non-functional Tests

This section focuses mainly on the functional tests, consider the fact that in the Spacewalk case, the non-funcional test techniques such as stress and performance tests apply rather to the specific set-up. For example, it is possible to stress a single component within the Spacewalk (e.g. as database) or to stress the whole stack by the particular scenario, but not vice versa.

There is a problem with non-functional test of Spacewalk that the most interesting ones might be hard to execute because of unsatisfiable requirements. Consider the following scenario: a huge amount of machines communicationg with the Spacewalk in the same time. In the real world, there might be thousands of client machines spread across the planet with different connectivity contactiong the Spacewalk through numerous Spacewalk Proxies. Such scenario could be the real bottleneck for the given Spacewalk installation, alas it is very hard to execute in the test laboratory. Saying nothing about routine retrials.

The changes introduced by this work are not the first line candidates to be a performance drawback for Spacewalk. Naturally, the Apt might be slowed down by the Apt-Spacewalk due to the additional network communication, but the slowdown might be hard to assert in a general. The work introduces is a new functionality across the multiple components and therefore the functional tests are needed first.

Considering Unit, Component, and Integration Levels

The tests can be also divided by the size of its target. The Unit tests aims on the small piece of software or its API, they are good choise for the developers of the monolitic systems. Component tests are a step above and focuses on the components as a whole. Finally, the integration tests verify the interoperation between multiple components often also simulating the uncertain environment.

The Spacewalk versioning system contains unit tests and a few component tests. However, a notable amount of the tests are outdated and not conforming the current source code. If the unit tests was in the good shape it would make a complete sense to amend them. The Apt-Spacewalk is a package too small for the unit tests, in addition, its main purpose is to enable the communication between other systems. Hence, the integration test which automates the communication between Spacewalk, Apt, rhn_check, and possibly

even more components would give us much more evidence about the functionality of whole scenarios. Such an integration test could be also used as regression test.

The Test Framework

The tests are written in the UNIX shell and the choosen test framework is the *Beakerlib* [33] Bash library. The Beakerlib provides functions which simplifies the logging and assertions from bash. In a shell, the most common assertions could be the check for a return value of command-line tool.

4.2.2 Apt-Spacewalk Sanity Test

The test is to be deployed and run on Debian GNU/Linux client. Ideally, the integration test is dependent on a very few variables, in any case it should not be dependent on the particular set-up nor the content. The implemented test passes and the log from the test run is present on attached CD.

The Apt-Spacewalk Sanity test covers installation, update, and removal of the Debian packages from the Spacewalk server. All the actions are being test with both base and child channels for multiple package architectures. First of all, the test creates the content within the given Spacewalk server. Consequently, it registers itself with the Spacewalk and consumes the content. Each action with Spacewalk is followed by assertions for expected results.

The test cases:

- Creation of the Debian channels
- Registration with the activation key
- Package push to the Spacewalk server
- Package and hardware profile synchronization with Spacewalk
- Creation and download of Debian repository metadata
- Installation and update against the Spacewalk
- Schedule and pick-up of remote action to install/update packages
- Purge of a Debian content

Chapter 5

Conclusion

The work introduced the Apt-Spacewalk package which handles the communication between Apt and Spacewalk server. With the provided package and the changes in existing projects (described in Appendices A and B), Debian-based clients are able to install and update packages from Spacewalk server. The Apt-Spacewalk package has been handovered to community [28] and the reported problems were addressed.

Patch for the Apt to support dynamic repositories has been submitted upstream, but the conversative Apt community has not responded at all even upon a reminder. The Apt-Spacewalk tool remains usable, even without applying the dynamic repositories patch. However, under certain circumstances, some actions require to be run twice in a row to succeed. Spacewalk community distributes both, patch and patched version of the Apt.

5.1 Further Work

Spacewalk is a complex system build around the rpm. Considering the current community which is focusing mainly on the functionality of rpm based systems, there will always be segments which will work for rpm but not for deb. Further work should focus on the features required by the community at most. The challenging ones are:

- Abillity to fetch the content of remote Debian repository into the Spacewalk channel.
- Support of the Debian clients in the Spacewalk Proxy server.
- Spacewalk to generate *pdiff* metadata known from Debian repositories.
- Abillity to store and manage Debian source packages.
- Spacewalk client tools become a part of standard Debian distribution.
- Inter-Spacewalk synchronization of Debian channels.
- Provisioning of Debian clients using the `cobbler` tool.

Bibliography

[1] Spacewalk. Homepage, http://spacewalk.redhat.com, [Online, 2011-04-30].

[2] L. Ďurfina. Native Support for DEB Packages in Spacewalk. Master thesis, FIT BUT, 2010.

[3] Yellow Dog Updater, Modified. Homepage, http://yum.baseurl.org, [Online, 2011-02-24].

[4] Seth Vidal. Yum Utils. Fedora Package Database, https://admin.fedoraproject.org/pkgdb/acls/name/yum-utils, [Online, 2011-05-02].

[5] RPM Package Manager. Homepage, http://rpm.org/, [Online, 2011-03-04].

[6] How to Setup Your Own Package Repository. Project Wikipage, http://yum.baseurl.org/wiki/RepoCreate, [Online, 2011-02-26].

[7] Writing Yum plugins. Project Wikipage, http://yum.baseurl.org/wiki/WritingYumPlugins, [Online, 2011-02-26].

[8] D. Burrows. Modelling and Resolving Software Dependencies. Unpublished, Debian, 2005.

[9] Debian New Maintainers' Guide. Tutorial, http://www.debian.org/doc/maint-guide/, [Online, 2011-02-27].

[10] Advanced Packaging Tool. Wikipedia, http://en.wikipedia.org/wiki/Advanced_Packaging_Tool, [Online, 2011-03-24].

[11] Apt.conf. Man Page, http://linux.die.net/man/5/apt.conf, [Online, 2011-03-06].

[12] A. Isotton. Debian Repository Howto. Tutorial, http://www.debian.org/doc/manuals/repository-howto/repository-howto, [Online, 2011-02-28].

[13] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform Resource Identifier (URI): Generic Syntax. RFC 3986 (Standard), January 2005.

[14] Sources.list. Man Page, http://linux.die.net/man/5/sources.list, [Online, 2011-02-27].

[15] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Draft Standard), jun 1999. Updated by RFCs 2817, 5785.

[16] Python-Apt v0.7.100 Documentation. API Documentation, http://apt.alioth.debian.org/python-apt-doc/, [Online, 2011-03-03].

[17] Spacewalk. Project Wikipage, https://fedorahosted.org/spacewalk/wiki, [Online, 2011-04-30].

[18] J. M. Rodriguez. Introducing Project Spacewalk. Announcement, https://www.redhat.com/archives/spacewalk-devel/2008-June/msg00001.html, [Online, 2011-05-02].

[19] History of Red Hat Linux. WWW Page, http://fedoraproject.org/wiki/History, [Online, 2011-05-02].

[20] Red Hat Network. Wikipedia, http://en.wikipedia.org/wiki/Red_Hat_Network, [Online, 2011-04-30].

[21] Red Hat Network Architectural Overview. WWW Page, http://www.redhat.com/red_hat_network/deploydetail/, [Online, 2011-02-28].

[22] Spacewalk Architecture. Project Wikipage, https://fedorahosted.org/spacewalk/wiki/Architecture, [Online, 2011-02-23].

[23] Spacewalk – Perl Stack. Project Wikipage, https://fedorahosted.org/spacewalk/wiki/PerlStack, [Online, 2011-04-25].

[24] D. Winer. XML-RPC Specification. http://www.xmlrpc.com/spec, January 1999.

[25] Spacewalk – Python Backend Architecture. Project Wikipage, https://fedorahosted.org/spacewalk/wiki/PythonDocumentation, [Online, 2011-04-25].

[26] Spacewalk – Taskomatic. Project Wikipage, https://fedorahosted.org/spacewalk/wiki/TaskoMatic, [Online, 2011-03-15].

[27] J. Pazdziora. Spacewalk on PostgreSQL. Presentation, http://www.adelton.com/docs/spacewalk/spacewalk-on-postgresql, [Online, 2011-04-30].

[28] M. Suchý. Spacewalk 1.4 Released. Announcement, https://www.redhat.com/archives/spacewalk-announce-list/2011-April/msg00000.html, [Online, 2011-04-30].

[29] Description of The API Used by Up2date, version 2. Spacewalk Documentation, http://git.fedorahosted.org/git/?p=spacewalk.git;a=blob_plain;f=backend/doc/api.txt, [Online, 2011-03-25].

[30] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, and L. Stewart. HTTP Authentication: Basic and Digest Access Authentication. RFC 2617 (Draft Standard), jun 1999.

[31] Š. Lukašík, , J. M. Navarro M. Suchý, and M. Vogt. Writing a Spacewalk plugin for Apt-Get. Mail Thread, http://lists.debian.org/deity/2011/02/msg00134.html, [Online, 2011-02-16].

[32] Write System Call. Man Page, http://linux.die.net/man/2/write, [Online, 2011-05-01].

[33] Beaker library. Homepage, https://fedorahosted.org/beakerlib, [Online, 2011-04-29].

Appendix A

Server Side Changes

In order to allow Debian clients, there are changes needed in the Spacewalk server codebase. Including the Java part, which generates the metadata for channels, and the backend part serving the content to the clients. All the described changes has been commited to the upstream project.

A.1 Taskomatic

Taskomatic [26] is a daemon which serves as a scheduler for asynchronous tasks needed by Spacewalk server. It is written in Java and deployed on Apache Tomcat. Taskomatic has a cron like ability for scheduling tasks on regular basis.

One of the tasks is `ChannelRepodata` and it is used for generating metadata for Software Channels. The task is scheduled on top of every minute and it looks up the channels with outdated metadata in order to regenerate them. The `ChannelRepodata` task utilizes the `RepositoryWriter` class to create the metadata for a given channel. Lukáš Ďurfina has modified the `RepositoryWriter` class to generate metadata not only for the Yum but also for the Apt.

A.1.1 Problem Definition

The problem with the solution in question was that the repository metadata was partialy incorect and their creation worked only under certain conditions. In addtion, the functionalities of Yum and Apt part were not equivalent.

The Spacewalk server stores the generated metadata on a disk in separate directories. Thus, `RepositoryWriter` needs to make sure that the appropriate directory exists. The `RepositoryWriter` has also a responsibility for the deletion of the metadata, and decision whether the metadata needs to be regenerated. Nevertheless, such capabilities were not present in the Debian part of the Taskomatic code.

The another problem was the content of generated *Packages* file. The *Packages* contained file path for each package equal to the location on the Spacewalk disk. On the other hand, Apache server was not configured to server the content from that path location. And even if the Apache was configured to do so, the result would be in contrary to the concept of the Software Channels (2.2.4).

A.1.2 Proposed Solution

Since the creation of Yum and Apt repository metadata differs, I propose to introduce a separate class for each. When the task `ChannelRepodata` starts it will look up the channel architecture and choose either `DebRepositoryWriter` or `RpmRepositoryWriter` class. The original `RepositoryWriter` class will remain as the abstract base class. The process of metadata deletion could be rewritten in more general way, with the ability to remove any metadata. The generalized code should become a part of abstract base class.

Additionaly, the Apt's metadata creation must be modified to point out the correct URI location for packages. The URI must involve the `/XMLRPC/GET-REQ/` handler followed by the channel's name. The URI is then used by Apt for the content download and `/XMLRPC/GET-REQ/` handler on Spacewalk side will cover the authentication and authorization of HTTPS requests.

A.2 Spacewalk Backend

Spacewalk backend is a part of the Spacewalk server writen in Python and deployed on Apache. Beyond others, the backend is responsible for handling the `XML-RPC` calls and HTTPS requests from clients. For example it might be the client registration, retrieval of metadata or packages, or the rhnpush of a package to the server. The part of the Spacewalk backend which serves the packages and channel metadata needs to be modified to allow the Debian content.

The handler, which serves the content listens on the `/XMLRPC/GET-REQ/` URI. The handler sticks to the principle that everything what is not allowed explicitelly is forbidden. Aparently, the Debian content belongs to the forbidden group. The proposed change is to allow the Debian content (patterns: `Packages.gz` and `*.deb`). The changeset is quite small and isolated. We have to note here that the similar change might be needed in the Spacewalk Proxy code.

Appendix B

RHN Client Tools

Rhn Client Tools is a set of libraries and executables used on the clients for elementar actions with Spacewalk server. The current implementation is distribution dependent. This appendix briefly describes the Debian port of the package.

B.1 Problem Definition

B.1.1 Description of Package

Rhn Client Tools package contains exacutables and Python modules used on a client side for a basic set of actions with the Spacewalk server. The modules encapsulate remote communication with Spacewalk as well as common client actions: parsing the configuration file, work with local package management tool, and collection of the information about the local machine—hardware and distribution related.

The executables allows registering with Spacewalk (`rhn_register`, `rhnreg_ks`), refreshing profile information (`rhn-profile-sync`), management of associated software channgels (`spacewalk-channel`), and picking up remote actions (`rhn_check`).

B.1.2 Debian Port

Library is distributed on Fedora GNU/Linux and its derivates. Certain parts of the library are distribution dependent. For example the library imports `python-rpm` and utilizes it as the only one packaging system.

Moreover, the library parses the information about installed distribution from an rpm package *providing* a redhat-relase and the information about system architecture from `/etc/rpm/platform` file. The configuration is stored in `/etc/sysconfig/` directory. Also note the documentation strings and output which is intended for Red Hat users. These listed facts, while they are native on Fedora, pose unsatisfied dependencies on Debian. Fortunatelly, The rest of the rhn-client-tools library is independent on the underlying distribution.

B.1.3 Current Solution

Lukáš Ďurfina in his Master thesis [2] provided a Debian package of `rhn-client-tools`. His package uses `python-apt` as package manager and parses distribution information from `/etc/issue` file.

The main problem with this package was that the Ďurfina's changes were not tracked in any versioning systems, which makes the maintainance hard. Furthermore, the package was still dependent on `python-rpm` and over the year diverged significantly from the upstream version.

B.2 Proposed Solution

In an ideal case, rhn-client-tools package would be compiled from a single source code for both platforms: rpm and deb. The package should neither require `python-rpm` on Debian nor `python-apt` on Fedora. More or less the implementations on Fedora and Debian are known. The problems is to integrate them to a single peace of software.

It could be achived by polymorphism (either in run-time or in build time) or by interleaving with if-else statements. Since the most of the rhn-client-tools library is not object oriented, polymorphism needs to be made not on the class level but on the module level. With such an approach, introducing a new module for each inconsistence, there is a risk for amount of modules to grow surprisingly. On the other hand, interleaving with if-else struts might lead to deep-nested if-else branches. Viable solution is to combine both concepts.

B.2.1 Implementation

When doing an change in the library code, the very first step should be to run the unit tests and maybe improve their coverage ration to involve also the planned change. The pity with rhn-client-tools package is that the unit tests are not maintained. During a work I have amended a few test cases, but most of them remained untouched.

The fundamential part of changes is a newly introduced `platform` module. The module contains only one function: `getPlatform` returning string equal to either `deb` or `rpm`. All the platform dependent bits from the library behave accordingly to given platform. The value of platform is stored statically and set up during the package build.

The next step would is to cut of the dependencies on a package manager. Let's take `rpmUtils` module as an example. The module is dependent on `python-rpm` and provides high level functions above it. The same functionality could be achieved on the Debian with a help of `python-apt` module. The usage of both modules (the `rpmUtils` and the new one `debUtils`) should transparent to the user. And the transparency could be achieved by polymorphism on the module level. The abstract module `pkgUtils` can look as follows:

```
1 from platform import getPlatform
2 if getPlatform() == 'deb':
3     from debUtils import *
4 else:
5     from rpmUtils import *
```

Some of the remaining modules in the library were also dependent on `python-rpm`, although the polymorphism on the module level might not be optimal for them, as it will lead to numerous and very small modules and lot of duplicity. The rest of platform dependent code in the rhn-client-tools package could be enterleaved with the local if-getPlatform-else branches.

Appendix C

Patch for the Apt

The patch has been proposed on the Apt's mailing list [31] and even thought the communication on the list was quite active before the patch was sent, no one has ever responded to the proposal.

```
1  === modified file 'apt-pkg/algorithms.cc'
2  --- apt-pkg/algorithms.cc       2011-02-10 16:51:44 +0000
3  +++ apt-pkg/algorithms.cc       2011-02-20 13:03:51 +0000
4  @@ -1458,12 +1458,17 @@
5      if (Fetcher.Setup(&Stat, _config->FindDir(``Dir::State::Lists'')) == false)
6         return false;
7
8  +   // Run scripts
9  +   RunScripts(``APT::Update::Pre-Invoke'');
10 +
11 +   // Refresh source lists, Pre-Invoke might change content
12 +   if (_config->FindB(``APT::Update::List-Refresh'',true) == true)
13 +       if (List.ReadMainList() == false)
14 +           return false;
15 +
16     // Populate it with the source selection
17     if (List.GetIndexes(&Fetcher) == false)
18          return false;
19 -
20 -   // Run scripts
21 -   RunScripts(``APT::Update::Pre-Invoke'');
22
23     // check arguments
24     if(PulseInterval>0)
```

The patch introduces a new boolean option—`APT::Update::List-Refresh` (line 12). If the option is set sources.list will be reloaded after `APT::Update::Pre-Invoke` and before the cache update.

The changeset has been tested with the common Apt frontends without observing any problems. Nonetheless, I admit that the patch has two hypothetical problems, which cannot be avoided unless the public API is changed. It changes behavior of the hook, which will be run even if the sources.list is empty, and it can change the value of the `List` reference (line 13).

Appendix D

Acquire Protocol Log

The following is the example of the Acquire Protocol (Section 3.1.1). This particular log is a snippet of the communication between Apt and Apt-Spacewalk tools captured during the `apt-get update` command. The lines introduced by '<' character were send by the `Spacewalk Acquire Method` (Section 4.1.1), while the lines introduced by '>' were sent by the Apt.

```
< 100 Capabilities
< Version: 1.0
< Single-Instance: true
<
> 600 URI Acquire
> URI: spacewalk://s01.lab/dists/channels:/Release.gpg
> Filename: /var/lib/apt/lists/partial/s01.lab_dists_channels:_Release.gpg
> Index-File: true
>
< 102 Status
< Message: Logging into the spacewalk server
< URI: spacewalk://s01.lab/dists/channels:/Release.gpg
<
< 102 Status
< Message: Logged in
< URI: spacewalk://s01.lab/dists/channels:/Release.gpg
<
< 102 Status
< Message: Waiting for headers
< URI: spacewalk://s01.lab/dists/channels:/Release.gpg
<
< 400 URI Failure
< FailReason: HttpError404
< Message: 404  Not Found
< URI: spacewalk://s01.lab/dists/channels:/Release.gpg
<
> 600 URI Acquire
> URI: spacewalk://s01.lab/dists/channels:/Release
> Filename: /var/lib/apt/lists/partial/s01.lab_dists_channels:_Release
> Index-File: true
>
< 102 Status
< Message: Waiting for headers
```

```
< URI: spacewalk://s01.lab/dists/channels:/Release
<
< 400 URI Failure
< FailReason: HttpError404
< Message: 404  Not Found
< URI: spacewalk://s01.lab/dists/channels:/Release
<
> 600 URI Acquire
> URI: spacewalk://s01.lab/dists/channels:/main/binary-i386/Packages.gz
> Filename: /var/lib/apt/lists/partial/s01.lab_dists_channels:_main_binary-i386_Packages
> Index-File: true
> Last-Modified: Sat, 01 Jan 2011 14:59:41 GMT
>
< 102 Status
< Message: Waiting for headers
< URI: spacewalk://s01.lab/dists/channels:/main/binary-i386/Packages.gz
<
< 200 URI Start
< Last-Modified: Tue, 21 Dec 2010 20:13:00 GMT
< URI: spacewalk://s01.lab/dists/channels:/main/binary-i386/Packages.gz
< Size: 1830
<
< 201 URI Done
< MD5-Hash: a7f25012454d56c1835fa54ce100a291
< MD5Sum-Hash: a7f25012454d56c1835fa54ce100a291
< URI: spacewalk://s01.lab/dists/channels:/main/binary-i386/Packages.gz
< Last-Modified: Tue, 21 Dec 2010 20:13:00 GMT
< Filename: /var/lib/apt/lists/partial/s01.lab_dists_channels:_main_binary-i386_Packages
< SHA256-Hash: f8db564d338475503140bcc7b83f1651e59ce32e00de4910c278e3dc70228fcf
< Size: 1830
<
```

Appendix E

Content of the CD

The attached CD contains source code of Apt-Spacewalk package, separate patches, final packages, and Apt-Spacewalk sanity test.

```
.
|-- logs
|   `-- acquire-methods
|-- packages
|-- README.txt
|-- src
|   |-- apt-spacewalk
|   |   |-- debian
|   |   |-- src
|   |   `-- test
|   `-- patches
|       |-- apt
|       |-- client
|       |-- server
|       `-- test
`-- tex
```