Zounds, a dselect avatar for Zoularis

Author: Julien T. Letessier
Contact: mezis@users.sourceforge.net
Date: 2002-11-23
Revision: 1.2

Abstract

This document presents the conception of zounds, an enhanced clone of Debian's dselect binary package management tool. The user desired features will be described, and an informal external specification will be deduced therefrom. We will conclude with a few implementation guidelines.

Introduction

NetBSD's packaging system has quite a few advantages over other. Unfortunately, unlike the package managers present on other platforms, it lacks a user-friendly interface: all management, be it of source or binary packages, is done through a set shell commands and intensive filesystem browsing.

Debian GNU/Linux does have a relatively friendly package manager called dselect. It's an interactive, ncurses-based interface to Debian's low-level binary package management tool, dpkg, that allows to list, fetch and install/uninstall packages.

Our goal is to provide users of NetBSD's packaging system (and its multi-platform port Zoularis) with a similar program, without it's interface clumsiness.

Desired Features

Existing frontends to package managers

Notes about `dselect`

Debian's `dselect` is a useful tool, but it's far from perfection. A little survey among would-be system administrators reveals the following opinions on this program.

The dark side—what users dislike in `dselect`:

- the obtrusive help pages: e.g. the page that systematically pops up before accessing the package selection screen;
- the incoherent and unclear keyboard shortcuts: depending on the current screen, one has to press either the spacebar or return key to confirm or exit;
- the search facility: it's fast, simple (press 'slash', like in `less` or `vi`), and incomplete... only the package names and one-line descriptions can be searched;

– the unclear tree-structure / sorting rules of the package list: this is a pain since Debian has almost 10K packages, and the clumsiness of the search facility doesn't help;

– the clumsy package listing: column widths are constant, eventually hiding essential information; the detailed info panel seems obtrusive and sometimes useless;

– updating the list of available packages is sluggish.

– `dselect` doesn't deal with building packages from source.

The bright side—what users would like to see in dselect:

– a consistent interface: familiar shortcuts (e.g. those of `nano` or `emacs`), accessible help features (e.g. status-line help and/or static summary of commands at bottom of the screen as in `pine`);

– a more complete search feature: searching through package names, short and long descriptions, and file listing should be possible.

– an intelligent package list, with multi-criteria filtering and optional info panel.

This user survey will help us to determine the feature set we need in our avatar.

Other package manager frontends

`aptitude`**:** is a very complete manager, though it's interface is somewhat complex. It features a foldable tree-like representation of the package list, with advanced (multi-criteria) grouping, filtering and sorting possibilities, accessible through a `mutt`-like rule syntax. It even has "themes" that group a set of user defaults.

`deity`**:** is a fairly basic, menu-driven frontend to `apt`.

`synaptic`**:** is a `GTK+` frontend to `apt` (not tested - no documentation found)

`stormpkg`**:** idem.

Additional points of interest

Multiple repositories

Several package repositories can coexist: for instance, there can be a local repository on the user's machine (e.g. if he's using `pkgsrc`), an online repository (e.g. on SourceForge), and a third repository on a CD-ROM. Our avatar needs (like `dselect`) to be able to fetch/list from different repositories, and merge the information available from these various repositories.

Retreiving information from the various repositories should be kept as fast as possible (unlike the monolithic and large `Packages.gz` file used in Debian repositories).

`pkgsrc`-related considerations

Users will also want our tool to include a user-friendly frontend to Zoularis' package compiling system, `pkgsrc`. Thus, it could permit easy building and installation of binary packages from source packages. Since a first version of the avatar must be produced in a few months, this possibility will probably not be included.

User experience

A good package management tool should feature unobtrusive on-line help. It's typically the kind of utility you don't want to read a manual or man-page for.

Conceptual guidelines

Our tool is going to be a general package manager. It will allow to manipulate several classes of objects:

- packages (in the sense of software sets);
- package files;
- package tags;
- a package tree.

Package tags

Package tags give hints about the status and characteristics of a package. Packages wear two types of tags (functional and thematic), described below; unless otherwise stated, a package can wear any set of tags.

functional tags: give details about the local status of a package. Each of the lines below features mutually exclusive tags; note that not installed appears in two different lines, though it's really the same tag.

- *installation status*: not installed, installed, auto-installed or broken;
- *user desire*: install, upgrade, remove;
- *freshness*: up-to-date, obsolete, not installed.

thematic tags : give info about what service a package provides.

- *NetBSD categories*: archivers, audio,... www, x11;
- *Functionality*: application, library, server, utility, documentation, environment (etc.);
- *Additional categories*: to be defined.

Package objects

Package objects are defined by the following characteristics:

- a name (e.g. bison);
- a list of pre-dependencies (the packages this one depends upon);
- a list of post-dependencies (the packages which depend on this one);
- a short and a long description text;
- a homepage URL;
- a maintainer & maintainer email address;
- a tag list;
- a list of package file references.

Package files

Package files represent the actual (versionned) distribution of a package. They are defined by:

- a package reference;
- a version+revision number (e.g. 1.35nb1);
- a strong hash key;
- an (ordered) list of availability locations.

Note that since dependencies and descriptions generally evolve with versions of the package, package objects and pacckage files will probably not be distinguished.

Package tree

Let's start by noting that the package tree we intend to use is not just the same as the package set, plus a tree structure. There will be duplicates.

Each of the tags is given a tag level (positive integer), which will help building the package tree. For instance, we could decide that the installation status tags are level 1, functionality tags are level 2, NetBSD categories are level 3, and additional categories are level 4.

The tree root has depth zero, and has no label. Internal tree nodes (hereafter refered to as nodes) a labeled with a package tag, which taglevel is equal to the node's depth. Other tree nodes (leaves) are labeled with a (reference to a) package.

Tree construction rules:

- given a tree node N, the subtree rooted in N contains all the packages which wear all of the tags N and its ancestors are labeled with;
- the children nodes of a node at depth d are labeled with each of the $(d+1)$-level tags.

The program should use a sensible default for taglevels, as they can complicate the user's tree browsing. Taglevels should also probably be user-setable, and a 'flatten' functionnality ought to be included, to let the tree forget about a level of tags by merging all the subtrees at a given level.

Informal external specifications

In this section, we'll present the external specifications of *zounds*, or at least enough of them to give useful guidelines for the implementation.

The repository

The repository specification used by zounds is described in a separate document, The Zounds Repository. This document should be available from the Solarpack project page [1]

[1] http://solarpack.sourceforge.net

Interface components

Here we'll detail the external aspect of the graphical interface; these are general rules that must be followed, but freedom will be given to the implementor to enhance this with implementation-specific possibilities (such as color) and make choices over the details (keyboard shortcuts, etc.)

Note that *zounds* functionality should also be available from the command-line, with an interface in the spirit of classical tools like `apt`.

States

State is the term used to refer to what the interface is currently used for. The following states will be used:

- *browse* describes when the user is perusing the package tree and acting on packages;
- *install* is when `zounds` is performing actions without user interaction (e.g. fetching, installing, removing packages);

The main view

The main view of zounds is divided in three parts: a short header, presenting general information (the program name, the active state and its status); a footer, composed of a status line (which can also be used as a mini-buffer for user input) and an online-help pane that presents the available commands at all times; and the main panel, the contents of which vary depending upon the current (active) state.

The following figure shows an example of the main view's aspect in the browse state. The main panel contains a representation of the active subtree, with the active subnode or package hilighted, and an optional detailed info pane:

```
+------------------------------------------------------------------------+
| Zounds 0.1.0                        BROWSING       current tree node   |
+------------------------------------------------------------------------+
|                                     |                                  |
|                                                                        |
|                                     |                                  |
|           main panel                           optional                |
|                                     |       "detailed info"            |
|         (current node view)                     panel                  |
|                                     |                                  |
|                                               (for long package        |
|                                     |        descriptions,             |
|                                                dependency lists,...)   |
|                                     |                                  |
+------------------------------------------------------------------------+
| status line (shows last action / prompts / mini-buffer)                |
+------------------------------------------------------------------------+
| available commands, "a la pine"                                        |
+------------------------------------------------------------------------+
```

The tree representation:

```
--- (parent node)

--- subnode 1
--- subnode 2

-+- expanded subnode 1
 |
 |  package line 1
 |_ package line 2

-+- expanded subnode 2
 |
 |-+- other subnode
 | |
 | |  package line 3
 | |_ package line 4
 |
 |  package line 5
 |_ package line 6

--- subnode 3
--- subnode 4

package line 7
package line 8
package line 9
```

Implementation notes

Guidelines

A package manager is base software. It shouldn't have too many requirements, if any; and these requirements should be established standards. We propose to use `python` (an object-oriented, interpreted langage) for development, because it's simple, scalable and reasonably fast (compared to other interpreted languages like `perl` or `tcl`). It features efficient bindings to `ncurses`, the *de facto* standard for console-based UIs.

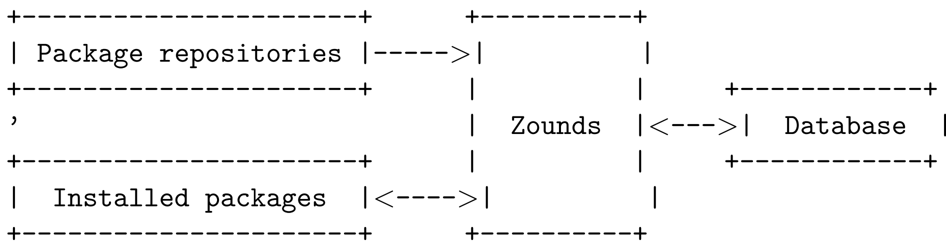
Python, being an interpreted (or *scripting*) language, allows fast and efficient development of applications; since it has the features of most modern languages (such has exception handling, signal support), it allows to develop reliable software components. Furthermore, it is currently considered as the fastest free interpreted language (including Java). Finally, it has a standard library that's impressive in size, and extremely completely documented.

Design Overview

Zounds operates mainly by interacting with a database. It uses a classical model of database (or *cache*) queries, eventually followed by rollbacks (cancel operations).

In other words, performing package management operation with zounds can be seen as either modifying the desire tags of known packages, or performing actual system-level installations and removals of software sets.

The following simple graphic sums up how Zounds interacts with its environment:



Zounds uses the metadata stored inside repositories [2] and the available information from the locally installed packages to keep its package cache (database) up-to-date. Each possible package exists in this cache in the form of a `Package` object; each of these objects wears appropriate functionnal tags at any time, indicating its installation status and the desire specified by the user.

When the user specify his desire to commit the desires he specified, Zounds uses this tags to bring the local installed package pool in conformity with the cache's tags, i.e. installs or removes packages accordingly.

User and Developer documentation

For now, Zounds lack a proper user and developer guide. This is not a critical issue, though; the code is very well documented with doc-strings (similar to Java's `javadoc` system), and a full API documentation can easily be extracted with tools such as `pydoc`, which comes with the standard Python distribution.

For users (administrators and repository maintainers), the program's README file documents the tools present in the Zounds distribution; this should probably be enough, as the Zounds tool itself comes with a self-explanatory command-line help.

Interface

Much work has to be done. For now, Zounds only implement a fairly basic command-line interface; in the future it will also have a `curses` based interface like described above.

Contents

- Introduction
- Desired Features
- Conceptual guidelines
- Informal external specifications
- Implementation notes

[2] see *The Zounds repository format*.