PackageKit backend and AppStream integration for Software Center

Google Summer of Code 2011 application for *openSUSE Project* by Alex Eftimie.

Abstract

AppStream is an initiative of cross-distro collaboration, which aims creating an unified software metadata database, and also a centralized OCS (Open Collaboration Services) user-contributed content database.

By this project, a PackageKit backend and integration with constructed metadata will be added to Software Center, providing a solid software management client for AppStream.

Detailed Description

Background

Software Center (SC) has been proposed as a client for AppStream because it is an established and stable project (reaching version 3.2 and replacing other package management tools in ubuntu's default install). It also have cool features such as Zeitgeist (usage stats) integration, an recommendation system, featured applications, ratings and comments.

The current backend of SC is *AptDaemon*, that takes care of package management over a D-Bus interface. *AptDaemon* has been designed with an architecture similar to PackageKit, one reason beside not using PackageKit in the first time being incompatibility between it and the apt installation progress (a transaction being uninterruptible, configuration file conflicts cannot be easily solved).

PackageKit is project aiming to unify the distribution package management system, in a transparent way for the user. It consists of a daemon and distro-specific package management backends. PackageKit exposes over D-Bus an interface backend agnostic with methods for install/removing/upgrading packages.

Beside package installation, the AppStream architecture shows also Mirror, Compose server and OCS server components. The Compose server extracts metadata from desktop files inside packages (this is later reffered as *distromatch*), construct it and make it available to the Mirror. A Mirror should provide to the client (SC) the metadata and icons retrieved from the Compose server. Existing metadata, used for package installation/removal/dependencies, should not be altered.

Development of another client for AppStream has already started (openSUSE App Store or Bretzn). The client, based on `MeeGo Garage` and `libattica` is developed using the Qt framework and fits best the KDE desktop. Without duplicating the effort, Software Center can be the Gtk+, GNOME client.

Use Cases

Andrei is a begginer to openSUSE, and doesn't know nothing about software packages. The distribution ships with a branded Software Center, PackageKit-enabled (with `rpm` and `zypp` backends). Andrei can easily find and manage software applications.

Benefits

- the *AppStream* cross-distro initiative, by having a good software management client;
- the *openSUSE* distribution will have a simpler to use interface for package management; it is probably to optimistic to think that it will ship in the next November release, but at least it will be available for testing, and further integration;
- *Software Center* as a project will benefit from having a backend abstraction; this is planned, but was not implemented yet.
- other distributions, that until now could not adopt Software Center for package management because of its only apt backend;

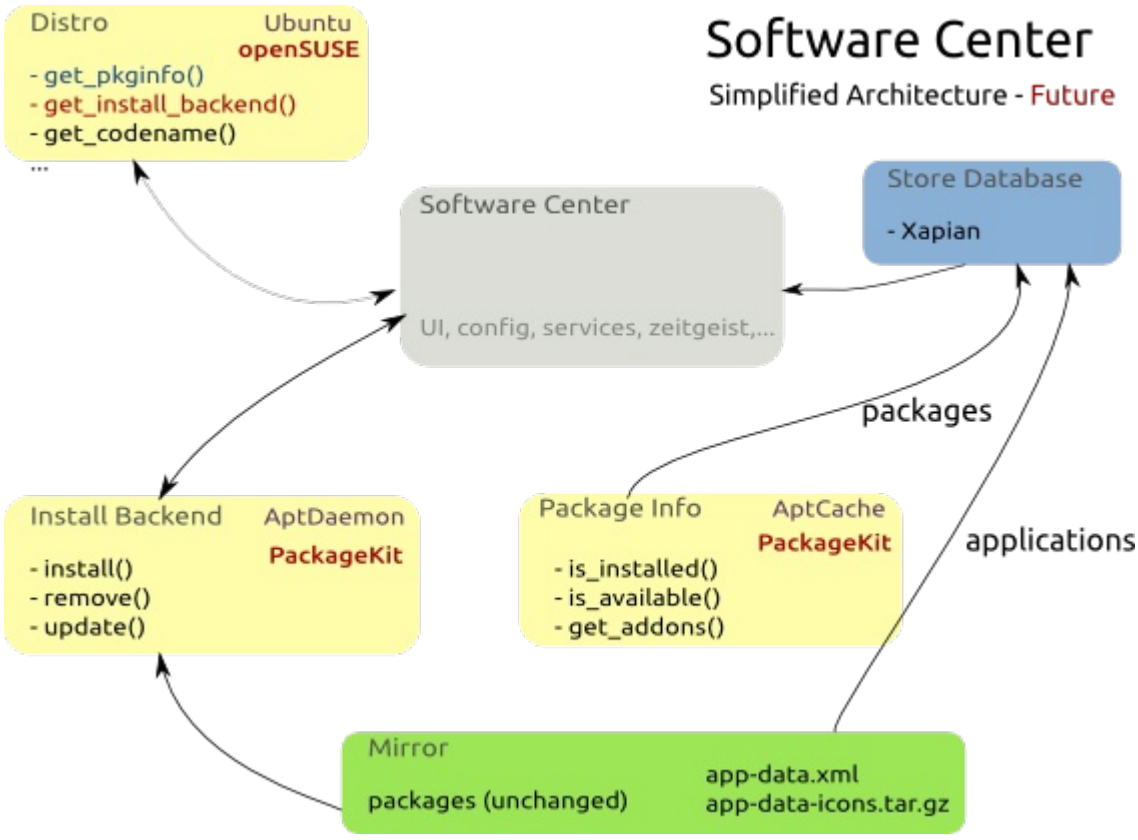
Caveats

Software Center has been designed and developed to fit Ubuntu. To change it is not an easy task, there will be features needing to be disabled/hidden when SC will run on other distribution (for example authorization and purchasing applications). This should be addressed by keeping in touch with SC upstream, and pushing changes gradually, so that they are accepted.

Making Software Center and PackageKit work together may raise problems of design compatibility. For example, if during a package installation, a conflict appears and PackageKit solves this conflict asking the user to chose and, in a similar situation, AptDaemon would avoid prompting the user - changes would be needed in Software Center GUI or in PackageKit. This kind of problems will appear after playing a little more with code/reading documentation and familiarizing with PackageKit workflow, and should be resolved using mentor and community feedback.

Technical Details

After analysing the source code, and discussing with the developers, this is a simplified design scheme:



Two main changes are required: first, **use the install/remove PK backend**, and second, create **a xapian database equivalent with the current `apt-xapian-index`, to populate it with data from *distromatch***.

The first, PK install backend, should be straightforward to implement, due to similarities between AptDeamon and PackageKit. Work has already been started in Michael Vogt `pk-abstraction` branch. The backend will extend a `BaseInstallBackend`, implementing the following methods:

```
class BaseInstallBackend(object):

    def upgrade(self, pkgname, appname):

    def remove(self, pkgname, appname):

    def remove_multiple(self, pkgnames, appnames):

    def install(self, pkgname, appname):

    def install_multiple(self, pkgnames, appnames):

    def apply_changes(self, pkgname, appname):

    def reload(self, sources_list):
```

The second, Xapian database populated with AppStream distromatch data, will need proper design and integration. Currently, a distromatch mirror can provide a `tar.gz` archive containing xml metadata and icons. These should be parsed and put into the database SC uses to present applications to the user.

Currently the search database (Xapian) is built from apt using this function (in `softwarecenter/db/update.py`):

```
def update(db, cache, datadir=APP_INSTALL_PATH):

    # index desktop files in $datadir/desktop/*.desktop
    update_from_app_install_data(db, cache, datadir)

    # index files in /var/lib/apt/lists/*AppInfo
    update_from_var_lib_apt_lists(db, cache)
```

First call fetches application information from `.desktop` files. This should be replaced by getting this data from the Mirror. The second call updates db with data stored in an apt specific format. This must be replaced by a PackageKit method of getting the package information for the current repository backend.

Another functionality needed is downloading *distromatch* data from the Mirror. Also, SC design requires an Package Information class (currently implemented as AptCache), with the following interface:

```
class BasePkgInfo(object):

    def is_installed(self, pkgname):

    def is_available(self, pkgname):
```

```
def get_addons(self, pkgname, ignore_installed):

def open(self):

def ready(self):
```

This will be implemented directly by wrapping over PackageKit interface for package information.

In addition to these main changes, there will be many small things scatterred around the code, relying on the current ubuntu-only setup, that will need to be discovered and fixed.

Timeline

To fulfill these requirements, I will folow this timeline (the timing is estimative):

prior official coding period

- discuss design architecture with S-C upstream and my mentor; deobfuscate ambiguities and participate in decision taking;

milestone 1 (first and 2nd week)

- get the latest SC running in Debian (currently, version 2 works, but version 3 is broken, due to ubuntu dependencies);
- identify and fix as many crossdistro issues as posible, such as dependencies, removal/abstraction of ubuntu specific parts;

milestone 2 (3rd to 5th week)

- familiarize with PackageKit workflow, especially with the `python-packagekit` bindings;
- implement a PackageKit install backend, extendind the `BaseInstallBackend` class;
- get SC running with the PK *install* backend in Ubuntu; write unit tests for the install backend; assure that both the `AptDaemon` and `PackageKitInstallBackend` pass these tests;

milestone 3 (6th to 8th week)

- understand how Xapian works, and create a database for *distromatch* data;
- deploy a personal Mirror storing composed *distromatch* metadata and icons;
- implement an AppStream *package info* backend, extending the `BasePkgInfo` class;
- figure out metadata fetching;

milestone 4 (9th week -)

- develop an openSUSE `Distro` class, and integrate it with the previous two new backend classes.
- work on cross distro fixes needed for a smooth integration;

After the 4th milestone, work should continue on integrating the resulted software with openSUSE distro, testing the resulted code and fixing bugs.

I think that using a higher level interface (PK Python bindings) for interacting with D-Bus, is a good idea; when PK D-Bus protocol changes, the bindings will also be updated upstream, and Software Center will benefit from this changes aswell, without requiring modifying the source code.

My development will rely on feature-based iterations. Each iteration will be completed when the features are implemented, and tests written for them pass. All changes should be made so that upstream adoption will be easy.

I plan to use `Bazaar` as a version control system, since this is the VCS used by Software Center, and using it will make easier pushing my changes back to the main development branch. If, for some reason, bazaar isn't agreated, my backup VCS is `git`.

Why Me

I am a student at the University Politehnica of Bucharest, in the first year of the Networks Security Master. For five years I've been using Linux as my primary operating system, and I'm passionate about Free Software and Open Source. Being a ROSEdu[1] member, I have gained experience on team work, nettiquette and community values.

My programming language of choice is Python, but I also do C , bash and PHP. Another areas of interest lately, are system administration and distribution packaging.

I have basic to intermediate knowledge of GTK+ and the GNOME desktop; one project I started is a thumbnailer for `.deb` packages[2]; my Graduation Thesis project was a PyGTK application (an paper exam generator and automatic corrector - using OpenCV); unfortunately that project isn't open yet.

My Python experience can be also reflected by my contribution to *World of USO*[3] open source project. This is a web based game for first year students, helping them check their gained knowledge in a fun way. I have participated at designing a new architecture mapped on Django, and wrote most of the actual code.

I watched Software Center grow, from the beginnings. On the first release, I contributed the Romanian translation. Some time ago, I also used a snapshot of it as a reference for understanding how a PyGTK application should work, where should the files be placed in the filesystem directory, and other best practices. Recently, I proposed a few patches to SC [4], [5] and [6]. The Software Center Team is a great team and responded positively to my questions; I'm keeping contact with them via IRC and email.

What I am expecting to learn during this summer programme, is: working with the community, participating at an cross-distro initiative, using technologies (such as Xapian, D-Bus), and great open source experience.

I am confident that I have the required skills and enthusiasm to be the perfect student for this project.

Availability

For me, this summer is reserved for GSoC; since always I wanted to apply, but I have never got an available summer. Either working on something else, either travelling, but never free to be part of the programme.

This GSoC project will be my main activity during the summer; the first two weeks (May 23 to June 5) it will be the Exam period at the university, so I estimate having up to 20 hours per week left for working on GSoC, and maybe taking a few days off); after that, I can commit for a 40 hours per week work time.

Contact Information

- *IRC:* alex3f on Freenode
- *e-mail:* alex@rosedu.org

- *XMPP:* alexeftimie@gmail.com

Links

[1] Romanian Open Source Education

[2] http://launchpad.net/deb-thumbnailer

[3] http://projects.rosedu.org/projects/wousodjango/

[4] #704719

[5] #575688

[6] #744655