
New Generation of Linux Meta-installers

Paulo Trezentos¹ and Roberto DiCosmo² and Stéphane Laurière³ and Mário Morgado¹ and João Abecasis¹ and Fabio Mancinelli² and Arlindo Oliveira⁴

¹ Caixa Mágica, ADETTI, ISCTE / `Firstname.Lastname@caixamagica.pt`

² PPS, Paris 7 / `Firstname.Lastname@pps.jussieu.fr`

³ Mandriva / `fdechelle@mandriva.com`, `slauriere@mandriva.com`

⁴ INESC-ID / IST `aml@inesc-id.pt`

1 Abstract

One of the points that differentiates most Linux distributions nowadays is the meta-installer. Meta-installers are software programs that run over RPM / DEB / TGZ installation systems, providing for automatic resolution of dependencies. Meta-installers like Apt-get, Smart or Yum are used widely and are starting to have more features.

A framework for the new generation of meta-installers are being devised by a joint group of research and commercial partners of EDOS project (Environment for the development and Distribution of Open Source software)[1].

This article presents the new meta-installer features that are being developed and the ones that are still under research such as: rollback, new dependency solving and hardware support by RPM / DEB packages.

These features have as common point the fact they extend the packaging system.

2 Rollback transactions

The process of installing and upgrading software packages in a system may not always be successful.

Even where an upgrade is successful, it may be the case that the newer version doesn't meet quality requirements or breaks compatibility with other applications.

A **rollback** mechanism in a **meta-installer** offers an easy way to undo changes introduced by an upgrade or downgrade.

This mechanism was originally developed for Caixa Mágica on a fork of Apg-get RPM and is included in the latest version of that distribution. Changes introduced in this fork were recently ported back to mainline Apt-

get RPM⁵ as well as to Debian's Apt-get⁶. And an effort is underway to merge the `rollback` feature in upstream repositories[2].

This feature for the Apt-get meta-installer was developed under the EDOS[1] project (Environment for the development and Distribution of Open Source software). Documentation and source code are available at Caixa Mágica's Apt-rpm project site⁷ and public subversion repository⁸.

The rollback mechanism developed allows to backup and restore the configuration files so the system can return to the state that the system was before the upgrade operation. For example if a service stops to work after an upgrade due a configuration problem or a any other problem the administrator, with the rollback mechanism can restore not only the older versions of the package but also all the configuration files present when the upgrade/downgrade operation occurred.

With the introduced changes, for every `apt-get` operation an entry is added to a transaction log database. For each affected package the package name, the type of operation (install, upgrade, removal, rollback) and the software versions involved are recorded. Furthermore, files labeled as configuration files in the RPM / DEB packages are locally backed up. Dependencies that are automatically added by apt are included in the transaction log as well. This allows one to easily revert the installation of a package and dependencies installed in the same transaction.

In the current implementation, a rollback action involves downgrading and re-installing packages to meet a previous state (or snapshot, in the language of other versioning systems, like Subversion and CVS). Apt-get repositories are required to provide older versions of packages, as needed. Configuration files are restored from the local backups.

Since the transactions are taken as whole, every time there is a rollback to an older version of a package, even if the currently installed version of the package in the system is the same, other packages affected by the original transaction are reverted to their former state.

This mechanism allows one to obtain a log of apt-get activity as shown in Listing 1.

Listing 1. apt-get rollback-hist output sample

```

~ # apt-get rollback-hist
Reading Package Lists... Done
Building Dependency Tree... Done
+-----+
Transaction ID: 1
Packages:
      xLucas(3-cm.4)    REMOVE
      lucas(3-cm.1)    INSTALL
+-----+
Transaction ID: 2

```

⁵ <http://aptrpm.caixamagica.pt/repo/aptrpm/branches/rollback@apt-rpm/>

⁶ <http://aptrpm.caixamagica.pt/repo/aptrpm/branches/rollback@apt/>

⁷ <http://aptrpm.caixamagica.pt/>

⁸ <http://aptrpm.caixamagica.pt/repo/aptrpm/>

```

Packages:
  apt4rpm(0.69.3-0) REMOVE
  apt-server(0.5.15-cnc6-cm10.6) REMOVE
+-----+
Transaction ID: 3
Packages:
  OpenOffice_org-en-help(1.1.3-16.1) INSTALL
  OpenOffice_org-pt(1.1.3-16.1) INSTALL
  OpenOffice_org(1.1.3-16.1) INSTALL
+-----+

```

After browsing through the transaction log, a transaction ID can be chosen to rollback. In the Listing 2 we chose to rollback the Open Office installation (transaction ID 3).

Listing 2. Rollback operation over the OpenOffice package installed on transaction id 3

```

~ # apt-get rollback 3
Reading Package Lists... Done
Building Dependency Tree... Done
Starting Rollback
The following packages will be REMOVED:
  OpenOffice_org OpenOffice_org-en-help OpenOffice_org-pt
0 upgraded, 0 newly installed, 3 removed and 0 not upgraded.
Need to get 0B of archives.
After unpacking 241MB disk space will be freed.
Do you want to continue? [Y/n] Y
...

```

This mechanism is already available in Caixa Mágica Linux version 11. The default `apt-get rpm` included in this distribution supports the Rollback mechanism described in this section.

One area of interest for the future development of the rollback mechanism includes providing further guarantees about the state of the system after a rollback. For example, files changed by installation scripts, currently, are not properly restored on a rollback, unless these files are also marked as configuration files in the packaging sub-system. This is a general problem that will probably be addressed in RPM and DEB. There is community interest in the subject, as was demonstrated in responses received while submitting changes upstream and in a recent discussion in the Fedora development list⁹.

3 Package installability problem

Package in a distribution are associated to meta-data that describes several properties of the package itself and that is used by automatic tools to perform common operation on it. One of the most important meta-data information is the definition of the inter-package relations. Such relations basically gives, for a given package, two kinds of information:

- *Dependencies*: what a package needs in order to function correctly.

⁹ <http://thread.gmane.org/gmane.linux.redhat.fedora.devel/47862>

- *Conflicts*: what are the packages that prevents its correct functioning.

Linux distributions provide huge package bases with thousands of packages that are interconnected by those inter-package relations. In a distribution like Debian we can find more that 200.000 of them. It is clear that checking if there are problems in this dense network is not easy.

In particular, what could happen is that, due to the particular configuration of these relations, some of the packages might be “broken”, i.e., not installable, because its dependency requirements could not be fully satisfied.

Checking if a package is broken or not implies the exploration of the dependency network and the extrapolation of a “solution” for the installation problem, i.e., a set of packages that when installed satisfy all the dependency requirements and do not contain any conflict among them.

We have developed a set of tools (i.e., `edos-rpmcheck` and `edos-debcheck`) that use formal techniques and given a package repository and the inter-package relations are able to find, if it exists, a solution to the installation problem for a given set of packages.

The tools basically implements a boolean satisfiability problem (SAT) solver that works on an encoding of the information found in the package repository as a boolean formula¹⁰. The existence of such an encoding also proves that the problem of checking the installability of a package is a hard one (i.e., it is NP-Complete).

If a solution that satisfies this formula, where a given package P is encoded to as to be installed, exist then there exists an installation for that package P . Otherwise there is a problem in the inter-package relations network that prevent the package from being installed.

The tools are able to spot what is the cause of the failure, giving the distribution editor a precise hint on where to look for solving the problem.

Figure 1 shows the output of `edos-debcheck` run on a snapshot of the Debian `testing` distribution. It is possible to see two typical problems that occur in package repositories. For example the package `firefox-dbg` is not installable because it depends on a version of the package `firefox` that is not available in the package repository.

The other failure, is a more complex one, and is due to a chain of dependencies that ends with a package `gforge-common` that conflicts with some other package that is part of this dependency chain, and so is required for the correct functioning of the package to be installed.

4 Hardware support

Hardware support for new devices by Linux kernel is one of the week points of Linux operating system.

¹⁰ The interested reader could refer to [3] for a complete description.

```

$ zcat Packages.gz | debcheck -failures -explain
Parsing package file... 2.4 seconds 19180 packages
Generating constraints... 3.4 seconds
...
firefox-dbg (= 1.5.dfsg+1.5.0.7-2): FAILED
The following constraints cannot be satisfied:
  firefox-dbg (= 1.5.dfsg+1.5.0.7-2)
    depends on firefox (= 1.5.dfsg+1.5.0.7-2) NOT AVAILABLE
...
gforge-theme-starterpack (= 3.1-1): FAILED
The following constraints cannot be satisfied:
  gforge-theme-starterpack (= 3.1-1)
    depends on gforge-common gforge-common (= 4.5.14-19)
  gforge-common (= 4.5.14-19)
    conflicts with gforge-theme-starterpack (= 3.1-1)
...
Checking packages... 0.5 seconds

```

Fig. 1. edos-debcheck execution on the Debian's testing repository meta-data

Hardware support is by nature supported directly by the kernel of Linux Operating system.

This happens by the inclusion of drivers, also know in Linux world as modules, in kernel packages.

However, not all existent modules are presented there because:

- They might not be available at Linux kernel release time
- Hardware vendor might not want to publicize the source code which was necessary in case of inclusion in Linux package
- Kernel maintainers such as Linus Torvalds or Andrew Morton might not want to include the driver in Linux kernel source code tree because of license, code maturity or other kind of issues.

There is a wide range of solutions found by hardware vendors like Intel, Nvidia or ATI, for publishing support for their hardware outside kernel tree. Such as making available:

- Compiled modules for a reduced number of kernel versions and suggest user to manually copy to kernel directory (Zyrex ADSL modems,...).
- Shell script with ncurses interface that compiles support and installs it in kernel directory (Nvidia,...).
- Tarball with configure, make, make install
- Module source code packaged in a RPM / DEB and compiled in instalation time
- Pre-compiled module packaged in a RPM / DEB (Atheros - Madwifi)

All of them might have advantages and disadvantages but RPM / DEB precompiled usage is the most promising approach.

A good hardware support in a Linux system should respect the following requirements:

- Module instalation should be transparent to the user and should occur when new hardware is detected, for instance, by HAL layer.
- Module may be in different repository locations (Linux distribution repository, hardware vendor repository, community repository like PLF or ContribWare) and should be discovered when the repositories are active.
- The mapping between the hardware and the package with the module with support for it should not be centralized, for instance, in a local hardware list.

Since a compiled module only works with the kernel version where it was compiled (with the exceptions that we mention ahead), we have three dimensions that result in a large number of needed packages:

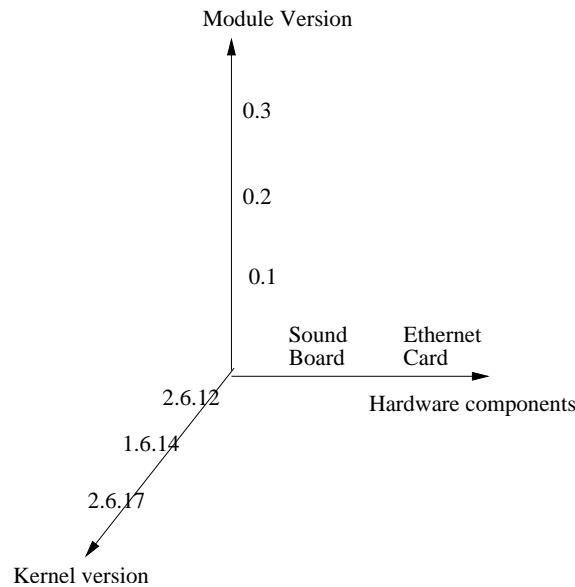


Fig. 2. Dimensions of needed packages

Regarding to figure 2 one axis may be eliminated: the kernel version.

If the module is compiled with kernel symbols presented in the system, then the module will be installable in all future kernel versions where the needed symbols were not changed.

The kernel symbols are variables and functions inside the kernel and they have specific information attached (like a kind of hash that helps to identify if a change occurred).

The packaging of a module could then be made, having as *depends*: the kernel symbols needed and as *provides*: the symbols exported.

This is always being done in some packages with modules such as Madwifi.

5 Future work

The future work may not only pass by this three hot topics but other like using P2P in instalation process.

5.1 Rollback

Rollback in meta-installers is a reality and is being shipped with Linux distributions.

It has however to be enhanced in the following points:

1. Reduce the granularity of the rollback from Transaction to Package
2. Display differences comparing different configuration files stored like in SVN or maybe even using SVN/CVS engine.
3. Port this feature to other meta-installer tools (in progress)

5.2 Dependency Solving

A lot of achievements were accomplished recently in this area but more can be done.

New research lines are being pursued in other to understand what kind of algorithms could be used to allow a better dependency solving mechanisms.

A promising field is applying Pseudo-Boolean Optimization (PBO) to the problem. In this case, the constrain problem would be optimized applying the following variables:

1. Number of installed packages that will be removed (less is better)
2. Total number of packages that will be installed and are not presently in the system (less is better)
3. Freshness (newer packages) of a solution compared with other solutions (newer are better)
4. Total amount of MB downloaded for a solution (less is better)

5.3 Hardware Support

Is it expected that hardware support requires multiple dimensions: kernel version, hardware model, and module version. Then, and since was proved by EDOS that package dependency is a NP-complete problem[3], the complexity will increase further.

But in order to work in a efficient way, other points must be assured:

1. Standard way to present dependencies, provides and conflicts for all hardware supporters are respected
2. Mapping between the detection of the hardware and the retrieving of the package in a distributed way
3. Robust meta-installer tools that support the large name of metadata presented in the repository
4. Tools to help compile and generate modules for different kernel versions that even might not be installed in the local machine using a chroot environment (enhancement of Debian pbuilder)

6 Conclusions

EDOS project is not focused in client-side meta-installers. Nevertheless, the knowledge generated inside of the project was enough to one embody the idea that much of Linux enhancements should - and will - be made through better meta-installers.

These client-side tools complement perfectly the software tool chain that are being made available by the EDOS project.

As stated in future work section, those are only the first steps in client-side and a lot of research effort should be focused in this subject to assure that Linux instalation becomes better, faster and easier.

References

1. Edos project. <http://edos-project.org>.
2. J. Abecasis. Rollback functionality for apt-rpm. <http://lists.laiskiainen.org/pipermail/apt-rpm-laiskiainen.org/2006-November/000542.html>.
3. J. Boender, R. DiCosmo, B. Durak, X. Leroy, M. Lijour, F. Mancinelli, T. Milo, M. Morgado, D. Pinheiro, R. Suarez, R. Treinen, P. Trezentos, J. Vouillon, and T. Zur. Wp2 - formal management of software dependencies - deliverable 2.2. <http://edos-project.org/xwiki/bin/download/Main/Deliverables/edos-wp2d2.pdf>.