Towards better tools for the analysis and quality assurance of FOSS distributions

Ralf Treinen

PPS, Université Paris Diderot

April 26, 2011
Mancoosi at Paris-Diderot

This is joint work with:

Pietro Abate  
Jaap Boender  
Yacine Boufkhad  

Roberto Di Cosmo  
Jérôme Vouillon  
Stefano Zacchirolı
Contents

1. Context: Components and FOSS

2. EDOS and Mancoosi: formal analysis of package relationships

3. Possible evolutions of component repositories
Software Components

Proposed 1968 by Douglas McIlroy as a remedy to the “software crisis”.

Some characteristics of components:

1. Multiple-use
2. Encapsulated i.e., non-investigable through its interfaces
3. A unit of independent deployment and versioning
4. Composable with other components

Problem: conflict between (3) and (4):

- Components evolve independently of each other, . . .
- . . . but they still have to work together.
Components can exist at different level: objects in the sense of an object oriented programming language, plugins for a specific platform, software packages in a GNU/Linux distribution, ... 

Main reason for bundling software items (programs, libraries, documentation, ...) into packages: ease of deployment and installation.

Without Software packages we either would have
- one single large system image
- or compile and install every single program by hand

Sharing of functionality between open components, instead of autonomous and closed software packages.
The F/OSS infrastructure is particularly challenging:

- no central architect
- fast, distributed development
- strong interdependencies
- very large code base (Debian: > 30,000 packages)
- provide packages for several compute architectures at a time (Debian: currently 11 architectures officially supported)
- possibly provide packages for several operating systems (Debian: 2 released OS, 1 experimental OS)
Number of binary packages in Debian

- Version 6.0 (Feb 2011): 28,000 packages
Players in the F/OSS universe

- “Upstream” authors: independent, develop software
- (sometimes) intermediate software assemblers: CPAN (perl), texlive (TeX)
- Distribution editors: create a coherent software distribution (Debian, Ubuntu, Mandriva, …)
- Sysadmins: install distribution on machine; updates
- Users
Workflow for the Package Maintainer

- Get upstream program. Is it fit for release?
- Create/update a *source* package: format mostly useful for specific tools of the distribution (for instance: Debian)
  Compilation of source packages produces (in general several) binary packages.
- Testing, use automatic tools for assessing the quality (rare!)
- Publish both source package and binary packages.
- Automatic compilation for other architectures/OS
- Wait for bug reports ...
Source and binary packages are coming from individual maintainers

Is the quality of individual packages OK?

Is the quality of the *collection of packages* OK?

From time to time: freeze the packages, throw out the bad ones, fix coherence problems, make an official release of a complete collection. (this is always a major pain!)
The current state of Quality Assurance in F/OSS

- Tests, sometimes automatic, for compilation and installation.
- No automatic generation of test cases
- No usage of automatic verification tools
- Urgent need of automatic tools for quality assurance both of individual packages, and of the distribution as a whole.
Workflow for the Sysadmin

- Initial installation of a complete distribution
- Add new packages to an existing installation
- Upgrade individual packages, or all packages (new functionalities, bug fixes, security updates)
- Probably: remove packages.
Why is FOSS interesting?

- Components, however components also exist elsewhere.
- A problem of scale: Large number of components, rapid evolution.
- All the data is freely available to everyone.
- We want to contribute to the advancement of FOSS.
The Mancoosi Project

- Mancoosi: Managing the Complexity of the Open Source Infrastructure
- European Research Project in the 7th Framework
- Duration: Feb 2008 → Mai 2011
- Successor of the EDOS European project (Jan 2004 → Jun 2007)
Mancoosi Project Partners

- Caixa Mágica Software
- EDGE'T
- Mandriva
- pixart
- Universita' dell'Aquila
- Université Paris Diderot
- Université Nice Sophia Antipolis
- ILOG
- INESC ID Lisboa
- UCL
- TEL AVIV UNIVERSITY

Changing the rules of business
Concrete view of a package

A package consists of

- An archive of files that are to be placed on the target host (for instance a file `/usr/bin/ocaml`)
- Optionally some actions that are performed when installing, upgrading, or removing a package: create symbolic links, create or remove user and groups, (un)register documentation, update hashtables, restart or stop services, ...
Concrete view of packages (2)

A package has prerequisites:

- System resources (disk space, ...)
- A certain version of a certain operating system
- File system structure (existence of, and access rights to certain directories)
- Availability of software libraries in a specific version
- Executability of other stand-alone tools
A package contains *metadata*:

- A package provides a certain functionality that is denoted by the name of the package, probably refined by the version number.
- A package may also provide even more abstract functionality (*feature, virtual package*), i.e. *web-browser*.
- All prerequisites are expressed through relations to other packages (or virtual packages), or possibly other meta-data i.e. space consumption of the package.
A concrete example of metadata

Package: hevea
Installed-Size: 2112
Maintainer: Debian OCaml Maintainers
  <debian-ocaml-maint@lists.debian.org>
Architecture: all
Version: 1.10-5
Depends: gs, netpbm (>= 2:9.10-1), ocaml-base-nox-3.10.2, tetex-bin | texlive-base, tex-common (>= 1.10)
Suggests: hevea-doc
Description: translates from LaTeX to HTML, info, or text.
Homepage: http://hevea.inria.fr/
Tag: implemented-in::ocaml, interface::commandline, ...
An more complex example

Package: myspell-hu
Architecture: all
Source: magyarispell
Version: 0.99.4-1.1
 Provides: myspell-dictionary, myspell-dictionary-hu, myhungarian
Depends: dictionaries-common (>= 0.10) | openoffice.org-updatedicts
Suggests: openoffice.org
Conflicts: openoffice.org (<= 1.0.3-2), myhungarian
Model (simplified)

Names, Versions and Constraints

- Set \( N \) of names
- Set \( V \) of versions: total and dense order
- Set \( \text{Con} \) of constraints: \( = v, > v, < v, \ldots \) where \( v \in V \)

A package \((c, v, D, C)\) consists of

- a package name \( n \),
- a version \( v \),
- a set of dependencies \( D \in \mathcal{P}(\mathcal{P}(N \times \text{Con})) \),
- a set of conflicts \( C \in \mathcal{P}(N \times \text{Con}) \),

A repository

is a set of packages, such that no two different packages carry the same name.
An \( R \)-installation is a set \( I \subseteq R \) with:

**abundance** For each element \( d \in p.D \) there exists \( (n, c) \in d \) and a package \( q \in I \) such that \( q.n = n \) and \( p.v \in [[c]] \).

**peace** For each \( (n, c) \in p.C \) and package \( q \in I \), if \( q.n = n \) then \( q.v \notin [[c]] \).

**flatness** For all \( p, q \in I \): if \( p \neq q \) then \( p.n \neq q.n \).

**Installability**

\( p \in R \) is \( R \)-installable if there exists an \( R \)-installation \( I \) with \( p \in I \).
Is a installable in R?

Repository R

Package: a
Version: 1
Depends: b, c, d

Package: b
Version: 17

Package: c
Version: 42
Conflicts: b
Is a installable in R?

<table>
<thead>
<tr>
<th>Repository R</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Package:</strong> a, <strong>Version:</strong> 1, <strong>Depends:</strong> b, c</td>
</tr>
</tbody>
</table>

| Package: b |
| Version: 17 |

| Package: c |
| Version: 42 |
| **Conflicts:** b > 15 |

Is a installable in R?
<table>
<thead>
<tr>
<th>Package</th>
<th>Version</th>
<th>Depends</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package a</td>
<td>1</td>
<td>b &gt;= 18, c</td>
</tr>
<tr>
<td>Package b</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>Package b</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>Package c</td>
<td>42</td>
<td>b &lt;= 17</td>
</tr>
<tr>
<td>Package</td>
<td>Version</td>
<td>Depends</td>
</tr>
<tr>
<td>---------</td>
<td>---------</td>
<td>--------------------</td>
</tr>
<tr>
<td>a</td>
<td>1</td>
<td>b, c</td>
</tr>
<tr>
<td>b</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>42</td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>87</td>
<td></td>
</tr>
</tbody>
</table>

**Conflicts**: b > 15
(Package,version) = Propositional variable
(package installed = value true)

Complete installation = propositional model

Modeling dependencies: $p \rightarrow \phi$ where $\phi$ is a positive formula

Package $p$ is not available: $\neg p$.

Dependency theory $D$: dual Horn theory:
Models are closed under union

$p$ is installable w.r.t. $D : D \land p$ satisfiable.

Since $D$ is dual Horn: $p$, $q$ co-installable iff $p$ installable and $q$ installable (so far).
Modeling conflict relations

**conflicts**

- A package $p$ may be in conflict with several other packages $q_1, q_2, \ldots$
- Conflict theory C: $\{\neg(p \land q_1), \neg(p \land q_2), \ldots\}$
  (neither Horn nor dual Horn)
- $p$ is installable: $p \land P \land C$ is satisfiable.

**A result from EDOS [ASE 2006]**

Installability of packages (measured in the number of packages) is NP-complete.
Modeling virtual packages

**virtual package**

If packages $p_1, \ldots, p_n$ provide a virtual package $q$:

$$q \rightarrow p_1 \lor \ldots \lor p_n$$

**Exclusivity constraint**

- Package $p$ both provides $q$ and conflicts with $q$.
- For every package $p' \neq p$ that provides $q$: $\neg(p \land p')$.
- Use case: allow only one package that provides a functionality, for instance *mail-transport-agent*. 
Written by Jérôme Vouillon in 2005, using SAT-solver technology.
Computes, for a complete distribution, *all* non-installable packages with explanation.
And it does this in *a few seconds*.
Integration into pkglab, an interactive system to explore package repositories of package-based software distributions.
### Usage in Debian

#### Web service edos.debian.net

Uninstallable packages in testing/main 17–23 June 2008:

<table>
<thead>
<tr>
<th>Date</th>
<th>alpha</th>
<th>amd64</th>
<th>arm</th>
<th>armel</th>
<th>hppa</th>
<th>i386</th>
<th>ia64</th>
<th>mips</th>
<th>mipsel</th>
<th>power</th>
</tr>
</thead>
<tbody>
<tr>
<td>23/06</td>
<td>367(7)</td>
<td>14(2)</td>
<td>217(4)</td>
<td>348(21)</td>
<td>369(9)</td>
<td>12(4)</td>
<td>48(3)</td>
<td>267(3)</td>
<td>269(3)</td>
<td>21(3)</td>
</tr>
<tr>
<td>Δ</td>
<td>+0/−0</td>
<td>+0/−0</td>
<td>+0/−1</td>
<td>+0/−0</td>
<td>+0/−0</td>
<td>+0/−0</td>
<td>+0/−0</td>
<td>+0/−0</td>
<td>+0/−0</td>
<td>+0/−0</td>
</tr>
<tr>
<td>22/06</td>
<td>367(7)</td>
<td>14(2)</td>
<td>218(4)</td>
<td>348(21)</td>
<td>369(9)</td>
<td>12(4)</td>
<td>48(3)</td>
<td>267(3)</td>
<td>269(3)</td>
<td>24(4)</td>
</tr>
<tr>
<td>Δ</td>
<td>+0/−0</td>
<td>+0/−0</td>
<td>+0/−0</td>
<td>+0/−0</td>
<td>+0/−0</td>
<td>+0/−0</td>
<td>+0/−0</td>
<td>+0/−3</td>
<td>+0/−3</td>
<td>+0/−0</td>
</tr>
<tr>
<td>21/06</td>
<td>367(7)</td>
<td>14(2)</td>
<td>218(4)</td>
<td>348(21)</td>
<td>369(9)</td>
<td>12(4)</td>
<td>48(3)</td>
<td>270(4)</td>
<td>272(4)</td>
<td>24(4)</td>
</tr>
<tr>
<td>Δ</td>
<td>+0/−0</td>
<td>+0/−0</td>
<td>+0/−0</td>
<td>+0/−0</td>
<td>+0/−0</td>
<td>+0/−0</td>
<td>+0/−0</td>
<td>+0/−0</td>
<td>+0/−0</td>
<td>+0/−0</td>
</tr>
<tr>
<td>20/06</td>
<td>367(7)</td>
<td>17(3)</td>
<td>221(5)</td>
<td>357(24)</td>
<td>369(9)</td>
<td>12(4)</td>
<td>48(3)</td>
<td>270(4)</td>
<td>272(4)</td>
<td>24(4)</td>
</tr>
<tr>
<td>Δ</td>
<td>+7/−0</td>
<td>+3/−0</td>
<td>+4/−3</td>
<td>+3/−27</td>
<td>+4/−0</td>
<td>+3/−0</td>
<td>+3/−0</td>
<td>+5/−11</td>
<td>+5/−0</td>
<td>+5/−0</td>
</tr>
<tr>
<td>19/06</td>
<td>360(5)</td>
<td>14(2)</td>
<td>220(6)</td>
<td>381(31)</td>
<td>365(8)</td>
<td>9(3)</td>
<td>45(2)</td>
<td>276(2)</td>
<td>267(2)</td>
<td>19(2)</td>
</tr>
<tr>
<td>Δ</td>
<td>+0/−0</td>
<td>+0/−0</td>
<td>+0/−0</td>
<td>+0/−0</td>
<td>+0/−0</td>
<td>+0/−0</td>
<td>+0/−0</td>
<td>+0/−0</td>
<td>+0/−0</td>
<td>+0/−0</td>
</tr>
<tr>
<td>18/06</td>
<td>360(5)</td>
<td>14(2)</td>
<td>220(6)</td>
<td>381(31)</td>
<td>365(8)</td>
<td>9(3)</td>
<td>45(2)</td>
<td>276(2)</td>
<td>267(2)</td>
<td>19(2)</td>
</tr>
<tr>
<td>Δ</td>
<td>+0/−0</td>
<td>+0/−0</td>
<td>+0/−0</td>
<td>+0/−0</td>
<td>+0/−0</td>
<td>+0/−0</td>
<td>+0/−0</td>
<td>+0/−0</td>
<td>+0/−0</td>
<td>+0/−0</td>
</tr>
<tr>
<td>17/06</td>
<td>360(5)</td>
<td>14(2)</td>
<td>220(6)</td>
<td>381(31)</td>
<td>365(8)</td>
<td>9(3)</td>
<td>45(2)</td>
<td>276(2)</td>
<td>267(2)</td>
<td>19(2)</td>
</tr>
</tbody>
</table>
Use in Debian (done by us)

- Release Team is using this information when finalising an official release.
- Detect *file conflicts* between packages: find pairs of packages that contain the same file and that can be installed together.
Use in Debian (not done by us)

- Verify installability of packages before uploading them to the archive, currently done by the *embedian* sub-project (debian for embedded systems).
- Used by debian autobuilders to avoid useless attempts to create build environments.
Putting results into practice

- The FOSS world is to a large extend influenced by a culture of volunteer communities (despite the fact that there are also players with important commercial interests).
- *Do-ocracy*: if you want to change the way things are done you have to implement it yourself and demonstrate that it works.
- Putting results into practice:
  - Identify a real problem in your community
  - Solve the problem with your technology
  - Integrate your solution into your community’s infrastructure and workflow
  - Convince people that it is useful.
The problem

distcheck

Given a repository $R$ and a package $(p, n) \in R$, is $(p, n)$ uninstallable w.r.t $R$?

Our question

Given a repository $R$ and a package $(p, n) \in R$, is $(p, n)$ uninstallable w.r.t all possible futures of $R$?

To be made more precise

Define “possible futures of $R$”
Example 1: Is \((foo,1)\) installable?

**Package**: foo  
**Version**: 1  
**Depends**: baz (\(= 2.5\)) | bar (\(= 2.3\)),  
bar (\(> 2.6\)) | baz (\(< 2.3\))

**Package**: bar  
**Version**: 2

**Package**: baz  
**Version**: 2  
**Conflicts**: bar (\(< 3\))
Example 2: Will \((foo, 1)\) ever be installable?

**Package:** foo
**Version:** 1
**Depends:** baz \((= 2.5)\) \| bar \((= 2.3)\),
bar \((> 2.6)\) \| baz \((< 2.3)\)

**Package:** bar
**Version:** 2.6

**Package:** baz
**Version:** 2.5
**Conflicts:** bar \((> 2.6)\)
One asks for installability of the *current version* of package *p* in all futures of *R*.

If a future *F* of *R* contains (*p*, *m*) with *m* > *n* then (*p*, *n*) is (vacuously) not installable in *F*.

The question is in reality about all futures of *R* that contain the *original version* of *p*.

Interesting for QA: such a package definitely needs action, since noone else can fix it!
Is the problem difficult?

- Not-installability of a package w.r.t. a current repository: co-NP complete.
  - For installability one guesses an installation (coherence is trivial to verify)
  - Allows to encode 3-SAT

- Not-installability of a package w.r.t. all possible futures:
  - co-NP hard, since it allows to encode the original non-installability problem.
  - however, there are infinitely many possible futures of a repository!
First approximation:

- Packages can move to newer versions (there is a total and dense ordering on version numbers)
- Newer versions of packages may change their relations in any way (quite pessimistic approximation)
- Packages may be removed.
- New packages may pop up.
- There are infinitely many possible futures.
What are possible futures of $R$?

A further complication (ignored for most of the rest of this talk):

- In a distribution, packages are upgraded by clusters of source packages. ⇒ all packages with the same source are synchronized in their version.
- This ignores abnormal situations due to autobuilder failure.
- It also ignores the fact that packages may change their source (this happens!)
- Problem: a source package may generate binary packages with different versions ⇒ it is not clear how future versions of binary packages relate.
### Futures

A repository $F$ is a *future* of a repository $R$, written $R \rightsquigarrow F$, if

**monotonicity** For all $p \in R$ and $q \in F$: if $p.n = q.n$ then $p.v \leq q.v$.

### Upgrades

If $R \rightsquigarrow F$, we say that a package $p \in R$ is *upgraded* when there is a $q \in F$ with $p.n = q.n$ and $p.v < q.v$. 
names(R): names of packages defined in R.

Focus of package sets
Let $R, P$ be two sets of packages. The $R$-focus of $P$ is

$$\pi_R(P) := \{(p.n, p.v) \mid p \in P, p.n \in names(R)\}$$

Focused properties
A property $\phi$ of installations is called $R$-focused if for all installations $I_1$ and $I_2$ (not necessarily subsets of $R$)

$$\pi_R(I_1) = \pi_R(I_2) \implies \phi(I_1) = \phi(I_2)$$
Admissible properties of futures

Let $R$ be a repository. A property $\psi$ of futures of $R$ is called \textit{admissible} if there is an $R$-focused property $\phi$ of installations such that for all futures $F$ of $R$:

$$\psi(F) \iff \text{for all } F\text{-installations } I: \phi(I)$$

Outdated packages

Let $R$ be a repository. A package $p \in R$ is \textit{outdated} in $R$ if $p$ is not installable in any future $F$ of $R$.

Outdated is admissible

$p$ is outdated in $R$ iff $\forall F. \forall I \in \text{Inst}(F), \phi_{out}(I)$ where

$$\phi_{out}(I) = (p.n, p.v) \notin \pi_R(I)$$
Definition

A repository $F$ is an optimistic future of a repository $R$ if any package in $F - R$ has empty dependency and conflicts.

Lemma

Let $R$ be a repository, and $\psi$ an admissible property of repositories. The following two assertions are equivalent:

- All futures $F$ of $R$ satisfy $\psi$.
- All optimistic futures $F$ of $R$ satisfy $\psi$. 
**Definition**

Let $R \rightsquigarrow F$. $F$ is a *conservative* future of $R$ if

$$\text{names}(F) = \text{names}(R) \cup \text{depnames}(R)$$

**Lemma**

Let $R$ be a repository, and $\psi$ an *admissible* property of repositories. The following two assertions are equivalent:

- All futures $F$ of $R$ satisfy $\psi$.
- All optimistic and conservative futures $F$ of $R$ satisfy $\psi$.
What remains to solve

- We have only a finite set of new package names.
- We may ignore package removals.
- New versions of packages have no relations (but conflict implicitly with different versions of packages with the same name, due to Debian semantics).
- Remaining problem: infinitely many future versions of packages, hence infinitely many future repositories.
Finitely many versions

- It is sufficient to consider, for package name $p$, version numbers that are explicitly mentioned, plus one intermediate, plus one that is beyond.

- Example: $(p, 5) \in R$
  Dependencies and conflicts in $R$ on $(p, \diamond 9)$, $(p, \diamond 12)$, where $\diamond$ is any comparison.

- Representatives of future versions of $p$:
  
  $5, 6(\in [5, 9]), 9, 10(\in [9, 12]), 12, 13(> 12)$
Further reduction: observational equivalence

- Consider all unary predicates on versions of $p$ occurring in $R$
- Build quotient under observational equivalence: identifying versions that behave the same on all these unary predicates.
Still, that’s a huge number of repositories

- So far we have a finite set (but huge) set $F$ of repositories.
- Packages $(p, n)$ in any repository in $F$ are unique (same metadata).
- We can build a new repository: $\bigcup F$, containing representatives of the complete future of all relevant packages.
- Any $R \in F$-installation is a $\bigcup F$-installation.
- 😞There are $\bigcup F$ installations that aren’t in any future repository because . . .
The problem when lumping together all futures

- Binary packages coming from the same source are synchronized!
- When considering $\bigcup F$: we have to exclude installations that mix binary packages coming from the same source but different version.
- Solution: add (versioned!) provides and conflicts:
  - If $(p, n)$ has source $s$: Add
    - Provides: $\text{src:s} (\equiv n)$
    - Conflicts: $\text{src:s} (\not\equiv n)$
- Finally: One single distcheck run on a large repository.
Class of admissible properties of futures.

Finite set of futures to consider.

Another instance of this class: “In any future in which $p$ is upgraded (now matter how) and without touching any other packages, it is no longer possible to install $q$”.

To do: define a logic for package repositories!