Mancoosi tools for the analysis and quality assurance of FOSS distributions

Ralf Treinen

UFR Informatique
Université Paris Diderot
treinen@pps.jussieu.fr

February 5, 2011
Joint work with the Mancoosi team at Paris-Diderot

Roberto Di Cosmo
Pietro Abate
Jaap Boender
Yacine Boufkhad
Jérôme Vouillon
Zack
Our (Paris-Diderot) focus in EDOS and Mancoosi

**Meta-data of packages**
- Core inter-package relationships:
  - Dependencies
  - Conflicts
  - Provides
- Optionally, less central relationships (recommends, etc.)

**Global analysis**
- Looking at a *complete distribution*
- E.g.: take into account dependency *chains*
- In contrast to looking checks (existence of mentioned packages)
Why is this interesting for FOSS distributions?

Dependency solving
- Which packages do I have to install/deinstall/upgrade in order to satisfy an installation request?
- Important task of package managers
- One focus of the Mancoosi project (2008 – 2011)

Quality assurance
- Which packages in a distribution need care?
- Only judging from meta-data, but analysis all over the distribution
- One focus of the EDOS project (2004 – 2007)
Why is this interesting for scientists?

**Software Engineering**
- Component-Based Software Engineering
- FOSS distributions are (afaik) the largest existing component-based systems
- FOSS systems are available for everybody to scrutinize

**Combinatorial Problem Solving**
- These are very challenging problems:
- Even basic problems are NP-complete (in theory computationally unfeasible)
- Due to logical relations between packages: conjunctions, disjunctions, and conflicts (explicit or implicit).
Just one word about dependency solving

- We need dependency solvers for many different component models.
- There are many different promising techniques for combinatorial problem solving.
- We advocate a modular approach, using CUDF as an interface language.
- MISC international competition for dependency solvers: talk to us when you are interested!!
At the beginning: a quite basic problem

- Given a repository $R$ of packages and a package $p \in R$, is $p$ installable w.r.t. $R$?
- That is: Does there exist $I \subseteq R$ such that
  - does the job: $p \in I$;
  - is \textit{in peace}: no conflicts inside $R$;
  - is \textit{abundant}: all dependencies in $R$ satisfied.
- That means: installable in a completely empty environment.
- 2005: Tools \texttt{edos-debcheck} and \texttt{edos-rpmcheck}
- Very efficient, using SAT-solver technology, and caching of results obtained for various packages in the distribution.
- Time for a demonstration . . .
Debian weather

- Running on edos.debian.net (today hosted by Mancoosi)
- Daily summary of uninstallable packages
- Differences between successive days
- Distinction between arch=all and arch-specific
- Date since when package uninstallable
- Explanation of uninstallability
- Demo …
Since Debconf 2010 (August 2010): bugs are filed against packages that are not installable, however only when
- package reported as non-installable everywhere
- uninstallable since some time (∼ 1 month)

List of reported bugs: See edos.debian.net
More uses of distcheck in debian

- **emdebian**: check installability of package before uploading new (versions of) packages to the archive

- **Build-dependencies**:
  - turn a build-dependency (conflict) into a normal dependency (conflict) of a dummy package
  - `edos-builddepcheck`: (currently) a wrapper that generates a new repository, then runs `edos-debcheck` on it
  - Used by debian autobuilders to avoid useless attempts to create build environments.
Detecting file conflicts

- Goal: detect cases where two packages can be installed at the same time, but doing so causes an error since one package tries to highjack a file owned by another package.
- Algorithm:
  - Look at the debian Contents file, compute all pairs of packages that contain a common file (debian sid: ~1000 pairs)
  - Use edos-debcheck to select pairs that are installable together (debian sid: ~170 pairs)
  - Test in a chroot
Statistics about detecting file conflicts

- Analysis done since april 2008, several times per week, on sid main+contrib+nonfree
- Bugs are reported with severity serious, against both packages (the debian BTS allows to file one bug against two packages)
- 290 Bugs found and reported (or simply reproduced)
- 286 of them are resolved. In most cases these bugs are closed very rapidly, however there are some hairy cases.
- See the list of bugs on edos.debian.net
From dose2 to dose3

- **dose2**: library that contains the basic building blocks of edos-debcheck
- **Problems with dose2:**
  - Debian and rpm package formats hard-wired: not very easy to add new package formats
  - Preprocessing (like for build-dependencies) requires ugly and inefficient wrappers
  - Output format not easily parsable
New in dose3

- CUDF as an intermediate format: new component models can easily be added, by writing a translator to CUDF.
- The new design of the library makes it much easier to integrate various preprocessing tasks.
- New output format based on YAML: human readable and machine parsable.
Graphical representation of dependency problems:

```
compass-susy-plugin1.8 (= 0.8.1-20100924-1)
  ↓
libcompass-ruby1.8
  ↓
libcompass-ruby1.8 (= 0.10.6~dfsg-1)
  ↓
libfssm-ruby1.8
  ↓
libfssm-ruby1.8 (= 0.1.4-2)
  ↓
librb-inotify-ruby1.8
  ↓
librb-inotify-ruby1.8 (= 0.7.0-4)
  ↓
libffi-ruby1.8
  ↓
missing
```
Two different questions that we have worked on:

- If we upgrade a particular package $p$, what are the other packages that (in their current version) become uninstallable? These are the packages that will have to upgraded together with $p$.

- If the current version of a package $p$ is found uninstallable w.r.t. the current repository: can this be solved by upgrading other packages in the distribution? If not, that means that $p$ has to upgraded!

And this is done with *distcheck* too!
What’s the future of a distribution?

- New packages may be created
- Packages may be removed
- Infinitely many possible future versions of packages
- Future versions of packages may change their dependencies/conflicts in an arbitrary way
It is sufficient to consider futures where all new versions of packages have **no** dependencies.

Justification: If $I$ is an $R$-installation, and $R'$ is obtained from $R$ by removing dependency or conflict relations, then $I$ is also an $R'$-installation.
A further problem: clustering

- In a distribution, binary packages do not evolve in isolation.
- They are updated in clusters that are identified by source packages.
- Consider only futures where all binary packages from the same source have moved together.
- That deliberately ignores: auto-build failures, packages that change source.
- Version numbers of packages with the same source may still have different version numbers: can be handled in restricted cases (binNMU, difference in epoch only).
Predicting the future

- Computing the consequences of updating on package: we have, for every relevant future of the package, run distcheck on that repository (takes around 10 hours)

- Analyzing which packages *must* be upgraded in order to become installable: we can just fold all relevant future versions of all package into one repository (sid: $\sim 70.000$ packages), plus conflicts between packages of different version and the same source (takes about 1 minute)
Results: upgrading by cluster

<table>
<thead>
<tr>
<th>Source</th>
<th>Version</th>
<th>Target Version</th>
<th>#(BP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>python-defaults</td>
<td>2.5.2-3</td>
<td>≥ 3</td>
<td>1079</td>
</tr>
<tr>
<td>python-defaults</td>
<td>2.5.2-3</td>
<td>2.6 ≤ . &lt; 3</td>
<td>1075</td>
</tr>
<tr>
<td>e2fsprogs</td>
<td>1.41.3-1</td>
<td>any</td>
<td>139</td>
</tr>
<tr>
<td>ghc6</td>
<td>6.8.2dfsg1-1</td>
<td>≥ 6.8.2+</td>
<td>136</td>
</tr>
<tr>
<td>libio-compress-base-perl</td>
<td>2.012-1</td>
<td>≥ 2.012.</td>
<td>80</td>
</tr>
<tr>
<td>libcompress-raw-zlib-perl</td>
<td>2.012-1</td>
<td>≥ 2.012.</td>
<td>80</td>
</tr>
<tr>
<td>libio-compress-zlib-perl</td>
<td>2.012-1</td>
<td>≥ 2.012.</td>
<td>79</td>
</tr>
<tr>
<td>icedove</td>
<td>2.0.0.19-1</td>
<td>&gt; 2.1-0</td>
<td>78</td>
</tr>
<tr>
<td>iceweasel</td>
<td>3.0.6-1</td>
<td>&gt; 3.1</td>
<td>70</td>
</tr>
<tr>
<td>haskell-mtl</td>
<td>1.1.0.0-2</td>
<td>≥ 1.1.0.0+</td>
<td>48</td>
</tr>
<tr>
<td>sip4-qt3</td>
<td>4.7.6-1</td>
<td>&gt; 4.8</td>
<td>47</td>
</tr>
<tr>
<td>ghc6</td>
<td>6.8.2dfsg1-1</td>
<td>6.8.2dfsg1+ ≤ . &lt; 6.8.2+</td>
<td>36</td>
</tr>
</tbody>
</table>
The Mancoosi Project

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