Bits for the Mancoosi project

yeah, including “visualizing package clusters” : -)

Stefano Zacchiroli
zack@{pps.jussieu.fr,debian.org}

Laboratoire PPS, Université Paris Diderot / The Debian Project

28 July 2009
DebConf9 — Cáceres, Spain
Outline

1. Past
   - The EDOS project
   - Package dependencies: the formal way

2. Present
   - QA tools
   - The Mancoosi project
   - Fun with the Debian dependency graph

3. Future
Outline

1 Past
   - The EDOS project
   - Package dependencies: the formal way

2 Present
   - QA tools
   - The Mancoosi project
   - Fun with the Debian dependency graph

3 Future
Outline

1 Past
   • The EDOS project
   • Package dependencies: the formal way

2 Present
   • QA tools
   • The Mancoosi project
   • Fun with the Debian dependency graph

3 Future
The EDOS project [http://www.edos-project.org]

name  Environment for the development and Distribution of Open Source software

funding  European Commission, IST activities 6th framework programme

timeframe  October 2004 – June 2007

consortium  universities (Paris 7, Tel Aviv, Zurich, Geneva), research institutions (INRIA), companies (Caixa Magica, Nexedi, Edge-IT (i.e. Mandriva), CSP Torino)

objective  study and solve problems associated with the production, management and distribution of open source software packages

Debian: not officially involved, but 1 DD (Ralf Treinen) was involved. A lot of code has been integrated into Debian and is being used daily for QA purposes.
EDOS Workpackages

EDOS was relatively broad in scope, split into several workpackages:

1. formal management of software dependencies
2. flexible testing framework
3. peer-to-peer content dissemination system
4. metrics and evaluation

Focus: distribution coherence from release manager’s point of view

Main question

Is it possible, for a given user selection of packages, to install them when only the packages from a given repository are available?
EDOS Workpackages

EDOS was relatively broad in scope, split into several workpackages:

1. formal management of software dependencies
2. flexible testing framework
3. peer-to-peer content dissemination system
4. metrics and evaluation

Focus: distribution coherence from release manager’s point of view

Main question

Is it possible, for a given user selection of packages, to install them when only the packages from a given repository are available?
Outline

1 Past
   • The EDOS project
   • Package dependencies: the formal way

2 Present
   • QA tools
   • The Mancoosi project
   • Fun with the Debian dependency graph

3 Future
What is an inter-package relationships?

First EDOS objective: establish a simple mathematical model of a distribution. Design decision: do so by looking at inter-package relationships.

Package: aterm
Depends: libc6 (>= 2.3.2.ds1-4), libice6 | xlibs (» 4.1.0), ...

to be interpreted as a propositional logic formula in CNF having (versioned) package names as literals, i.e.

\[ \text{libc6} \land (\text{libice6} \lor \text{xlibs}) \land \ldots \]

... some care is required though:

- **multiple versions**: foo becomes foo\(_{1.0}\) | foo\(_{1.1}\) | ...
- **virtual packages**: m-t-a becomes postfix | exim | sendmail | ...
What is a repository then?

Putting it all together, a distribution repository is modeled as:

1. a set of (versioned) packages $P$
2. a function $D$ associating packages to dependencies (formulae)
3. a set of conflicts $C$, i.e. pairs of non co-installable packages

Example (modeling of the previously shown Packages)

\[
P = \{(a, 1), (b, 2), (b, 3), (c, 3), (d, 1), (d, 2), (d, 3)\}
\]
\[
D(a, 1) = \{\{(b, 2), (b, 3)\}, \{(c, 3), (d, 2), (d, 3)\}\}
\]
\[
D(b, 2) = \emptyset
\]
\[
\ldots
\]
\[
C = \{((b, 2), (b, 3)), ((b, 3), (b, 2)), ((c, 3), (b, 2))\}, \ldots\}
Package installability as SAT

The problem of whether a package is installable in a given repository is equivalent to SAT:\(^1\)

- each package \( p \) with version \( v \) is a boolean variable \( p_v \)
  - if \( p_v \) then the package should be installed else it should not
- each dependency is a logical implication, e.g.:
  \[ \text{aterm} \rightarrow \text{libc6} \land (\text{libice6} \lor \text{xlibs}) \land \ldots \]
- each conflict between \( a \) and \( b \) is a formula \( \neg (a \land b) \)

Theorem

A package \( p \) (with version \( v \)) is installable iff there exist a boolean assignment that makes \( p_v \) true, and satisfies the encoding of the repository.

(Not so) nice consequence: package installability is a hard problem.

\(^1\)deciding whether a formula in propositional logic is satisfiable or not
Package installability as SAT — example

apt-get install
libc6=2.3.2.ds1-22

in

Package: libc6
Version: 2.2.5-11.8

Package: libc6
Version: 2.3.5-3

Package: libc6
Version: 2.3.2.ds1-22

Depends: libdb1-compat

Package: libdb1-compat
Version: 2.1.3-7
Depends: libc6 (>= 2.3.5-1)

Package: libdb1-compat
Version: 2.1.3-8
Depends: libc6 (>= 2.2.5-13)

becomes

\[
\begin{align*}
&I_{\text{libc6}}^{2.3.2.ds1-22} \\
&\land \\
&\neg (I_{\text{libc6}}^{2.3.2.ds1-22} \land I_{\text{libc6}}^{2.2.5-11.8}) \\
&\land \\
&\neg (I_{\text{libc6}}^{2.3.2.ds1-22} \land I_{\text{libc6}}^{2.3.5-3}) \\
&\land \\
&\neg (I_{\text{libc6}}^{2.3.5-3} \land I_{\text{libc6}}^{2.2.5-11.8}) \\
&\land \\
&\neg (I_{\text{libdb1-compat}}^{2.1.3-7} \land I_{\text{libdb1-compat}}^{2.1.3-8}) \\
&\land \\
&I_{\text{libc6}}^{2.3.2.ds1-22} \rightarrow \ (I_{\text{libdb1-compat}}^{2.1.3-7} \lor I_{\text{libdb1-compat}}^{2.1.3-8}) \\
&\land \\
&I_{\text{libdb1-compat}}^{2.1.3-7} \rightarrow \ (I_{\text{libdb1-compat}}^{2.3.2.ds1-22} \lor I_{\text{libc6}}^{2.3.5-3}) \\
&\land \\
&I_{\text{libdb1-compat}}^{2.1.3-8} \rightarrow I_{\text{libc6}}^{2.3.5-3}
\end{align*}
\]

...average formula has 400 literals, KDE installation 32’000
Good qualities for a repository

An installation is a repository subset. In a healthy installation: all dependencies are satisfied (*abundance*) and no pairs of conflicting packages are co-installed (*peace*)

    i.e. what our package managers are meant to enforce!

A package in a repository is **installable** if there exists at least one healthy installation which contains it

    i.e. there is at least *one way* for our users to install it

A package repository is **trimmed** if every package it contains is installable wrt the repository itself

    i.e. there are no “broken” packages

Shipping non-trimmed repositories = shipping packages that users will not be able to install
Outline

1. Past
   - The EDOS project
   - Package dependencies: the formal way

2. Present
   - QA tools
   - The Mancoosi project
   - Fun with the Debian dependency graph

3. Future
Outline

1. Past
   - The EDOS project
   - Package dependencies: the formal way

2. Present
   - QA tools
   - The Mancoosi project
   - Fun with the Debian dependency graph

3. Future
Quality Assurance

On the basis of the presented formalization, several QA tools for distro have been developed:

**edos-debcheck** command line checker for package installability

**pkglab** interactive, console-based environment for repository inspection

**ceve** parser/converter between package list formats

**tart** slice a repository (e.g. media), so that packages available on the $i$-th slice are installable using only slices up to $i$
edos-debcheck

- edos-debcheck takes as input **APT package list(s)** and checks whether one, several, or all packages in it are installable

Customized SAT solver, **very fast**: checking installability of all package in main testing/amd64 takes 5 seconds on an entry-level machine.

**Example**

edos-debcheck </var/lib/apt/lists/..._main_binary-amd64_Packages
Parsing package file... 1.2 seconds 21617 packages
Generating constraints... 2.3 seconds
Checking packages... 1.5 seconds
acx100-source (= 20070101-3): FAILED
alien-arena (= 7.0-1): FAILED
alien-arena-browser (= 7.0-1): FAILED
alien-arena-server (= 7.0-1): FAILED
alsa-firmware-loaders (= 1.0.16-1): FAILED
amoeba (= 1.1-19): FAILED
...
# explanation can be required as well

**Debian package**: edos-debcheck

**main author**: Jérôme Vouillon
edos-debcheck noteworthy applications

- **EmDebian**: upload time check to avoid uninstallability
  - harder in Debian: long path between upload and archive
  - how about an advisory dput hook?

- **edos-builddebcheck**: wrapper around edos-buildcheck to check satisfiability of *build-dependencies* (by zack and treinen)
  - used pre-release to check buildability in the new release
  - soon(?) in wanna-build to avoid spurious errors (by nomeata)

- **uninstallable packages**, daily monitor
  [http://edos.debian.net/edos-debcheck](http://edos.debian.net/edos-debcheck)

- **undeclared Conflicts**, periodic monitor (by treinen)
  [http://edos.debian.net/missing-conflicts/](http://edos.debian.net/missing-conflicts/)

dpkg: error processing
/var/cache/apt/archives/gcc-avr_1%3a4.3.0-1_amd64.deb (--unpack):
trying to overwrite ‘/usr/lib64/libiberty.a’, which is also in package binutils
Debian weather!

Just for fun, Debian weather (http://edos.debian.net/weather/) gives a weather-like representation of uninstallable packages statistics

The “Debian weather” for today: mostly sunny in stable and testing, at places overcast and rainy in unstable.

<table>
<thead>
<tr>
<th>Condition</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>clear</td>
<td>&lt; 1%</td>
</tr>
<tr>
<td>few clouds</td>
<td>1%...2%</td>
</tr>
<tr>
<td>clouds</td>
<td>2%...3%</td>
</tr>
<tr>
<td>showers</td>
<td>3%...4%</td>
</tr>
<tr>
<td>storm</td>
<td>&gt; 4%</td>
</tr>
</tbody>
</table>

Stable:

- clear
- few clouds
- clouds
- showers
- storm

Testing:

- clear
- few clouds
- clouds
- showers
- storm

Unstable:

- clear
- few clouds
- clouds
- showers
- storm

alpha amd64 arm hppa i386 ia64 mips mipsel powerpc
pkglab

- pkglab is an interactive, console-based environment to explore package repositories of package-based software distributions.

Features:
- load current and past package lists
- package installability checks (a-la edos-debcheck)
- functional query language (map, filter, fold, . . .)
- inspect historical evolution of repositories

Debian package: pkglab
> $diag <- check($unstable,$unstable)
Solver: Computing closure
Solver: Done, 22156 packages in closure
Solver: Numbering
Solver: Converting to boolean problem
Solver: Done, formula of size 200184
<diagnosis:closure size 22156, 141 failures>
> #show $diag
Diagnosis:
  Conflicts: 13997
  Disjunctions: 155280
  Dependencies: 164279
  Failures (total 141):
  Package acidlab'0.9.6b20-22@all
cannot be installed:
  acidlab'0.9.6b20-22@all depends on one of:
    - libphp-phplot'4.4.6+5.0rc1.dfsg-0.1@all
  libphp-phplot'4.4.6+5.0rc1.dfsg-0.1@all
depends on missing:
    - php3
    - php3-cgi
    - php4
    - php4-cli

(* same check in stable of a few months ago *)
check(acidlab'0.9.6b20-22@all,
    contents(%debian/stable/main/i386, 2008-03-20))
(...)
<diagnosis:closure size 557, 0 failures>
(* check co-installability of php{4,5} *)

> $d<-check_together(
  php4'6:4.4.4-8+etch4@all,
  php5'5.2.5-3@all, $a)
Solver: Not successful, 1 failures
> #show $d
Diagnosis:
(...)
Failures (total 1):
  Packages php5'5.2.5-3@all
  and php4'6:4.4.4-8+etch4@all
cannot be installed together:
  php4'6:4.4.4-8+etch4@all
depends on missing
  - libapache-mod-php4(>='6:4.4.4-8+etch4)
  - libapache2-mod-php4(>='6:4.4.4-8+etch4)
  - php4-cgi(>='6:4.4.4-8+etch4)

(* works in the union of stable and unstable *)

> check_together(php4'6:4.4.4-8+etch4@all,
  php5'5.2.5-3@all,
  $a|contents(%debian/stable/main/i386,
    2008-03-20))
(...)
<diagnosis_list:closure size 857,
  0 failures>
Outline

1 Past
   - The EDOS project
   - Package dependencies: the formal way

2 Present
   - QA tools
   - The Mancoosi project
     - Fun with the Debian dependency graph

3 Future
Mancoosi picks up the baton from where EDOS left: the focus is now the sysadm (our user and her interaction with package management).

**name**  MANaging the COmplexity of the Open Source Infrastructure

**funding**  European Commission, IST activities 7th framework programme

**timeframe**  February 2008 – January 2011

**consortium**  universities (Paris 7, L’Aquila, Sophia Antipolis, Tel Aviv, Louvain), research institutions (INESC-ID), companies (Caixa Magica, Pixart, Edge-IT (i.e. Mandriva), ILOG)

**objective**  develop *rollback mechanisms for package upgrades* and *better algorithms to plan package upgrade paths*

Debian is not officially involved, but 2 DDs (treinen and zack) are enrolled as researchers among the ranks of Paris 7
The upgrade problem

**Upgrade problem** = the “problem” posed by a user request to change the *local status* of installed packages

Solving an upgrade problem can *fail* for several reasons:

- invocation error, dependency solving, package retrieval, package unpacking, maintainer script execution, . . .

Mancoosi will try to attack the upgrade problem from two sides:

**rollback support** there are unpredictable failures (e.g. maintscripts), a posteriori recovery techniques are the only way out

**dependency solving** not satisfying meta-installer state of the art (e.g. *incompleteness*: the inability to find a solution when there is one): we should to better!

while studying this ... we’ve met the Debian dependency graph
The upgrade problem

Upgrade problem = the “problem” posed by a user request to change the *local status* of installed packages
Solving an upgrade problem can *fail* for several reasons:

- invocation error, dependency solving, package retrieval, package unpacking, maintainer script execution, . . .

Mancoosi will try to attack the upgrade problem from two sides:

- **rollback support** there are unpredictable failures (e.g. maintscripts), a posteriori recovery techniques are the only way out
- **dependency solving** not satisfying meta-installer state of the art (e.g. *incompleteness*: the inability to find a solution when there is one): we should to better!

while studying this ... we’ve met the Debian dependency graph
Outline

1 Past
- The EDOS project
- Package dependencies: the formal way

2 Present
- QA tools
- The Mancoosi project
- Fun with the Debian dependency graph

3 Future
Debian dependency graph

- a node for each (binary) package
- an edge from \( p \) to \( q \) each time \( q \) appears somewhere in the (Pre)-Depends field of \( p \)

Debian is huge, its dependency graph is huge as well: about 25’000 nodes, 400’000 edges.
It used to grow exponentially, it is stabilizing.
All dependencies are equal but . . .

The explicit, syntactic dependency relation $p \rightarrow q$ is too coarse grained to answer natural questions like:

\emph{can I remove package p without affecting package q?}

Answer may not be dependent on packages $p$ and $q$ only! 

e.g.: alternative (OR-ed) dependencies, virtual packages

let’s try again

**Strong dependencies**

$p$ strongly depends on $q$ with respect to repository $R$ ( $p \Rightarrow_R q$) if it is not possible to install $p$ without also installing $q$
Strong vs “normal” dependencies

Example

Package: p
Depends: q, r

Package: a
Depends: b | c

Strong deps: \( p \Rightarrow q, p \Rightarrow r \)

Example

...but in general things get more complicated:

Package: p
Depends: q | r

Package: r
Conflicts: p

Package: q

the conflict can come from a galaxy far, far away ...

Strong deps: \( p \Rightarrow q \)
Correlation between strong and normal dependencies

(data from Lenny)
Impact Set and Package Sensitivity

*Impact set*: the set of packages potentially affected by changes in a given package.

Definition (Impact set of a component)

Given a repository \( R \) and a package \( p \) in \( R \), the *impact set* of \( p \) in \( R \) is the set \( Is(p, R) = \{ q \in R \mid q \Rightarrow p \} \).

Similarly, the *direct impact set* of \( p \) is the set \( Dirls(p, R) = \{ q \in R \mid q \rightarrow p \} \).

Definition (Sensitivity)

The strong sensitivity, or simply *sensitivity*, of a package \( p \in R \) is \( |Is(p, R)| - 1 \), i.e., the cardinality of the impact set minus 1. Similarly, the *direct sensitivity* is the cardinality of the direct impact set.

Idea: sensitivity assess how “delicate” is a package.

How many packages can I break uploading/installing \( p \)?
Top 15 of sensitive packages in Lenny

What’s the most sensitive package in Lenny?
# Top 15 of sensitive packages in Lenny

| #  | Package         | | p | || p || | || p || − | p || |
|----|----------------|---|---|-----|-----|-----|-----|-----|
| 1  | gcc-4.3-base   | 43 | 20128 | 20085 |
| 2  | libgcc1        | 3011 | 20126 | 17115 |
| 3  | libselinux1    | 50  | 14121 | 14071 |
| 4  | lzma           | 4   | 13534 | 13530 |
| 5  | coreutils      | 17  | 13454 | 13437 |
| 6  | dpkg           | 55  | 13450 | 13395 |
| 7  | libattr1       | 110 | 13489 | 13379 |
| 8  | libace1        | 113 | 13467 | 13354 |
| 9  | perl-base      | 299 | 13310 | 13011 |
| 10 | libstdc++6     | 2786 | 14964 | 12178 |
| 11 | libncurses5    | 572 | 11017 | 10445 |
| 12 | debconf        | 1512 | 11387 | 9875  |
| 13 | libc6          | 10442 | 20126 | 9684  |
| 14 | libdb4.6       | 103  | 9640  | 9537  |
| 15 | zlib1g         | 1640 | 10945 | 9305  |

...
Dominators

Intuition

$p$ dominates $q$ if the strong dependency of $p$ on $q$ “explains” the impact set of $q$, i.e., $q$ is “important” due to a lot of other packages which requires $p$ (it is the case for gcc-4.3-base).

Definition

Strong dominance Given two packages $p$ and $q$ in a repository $R$, we say that $p$ strongly dominates $q$ ($p \trianglerighteq_{Is} q$) iff

- $Is(p, R) \supseteq (Is(q, R) \setminus Scons(p))$, and
- $p$ strongly depends on $q$

The dominance relation gives a good device to highlight complex structure in the Debian dependency graph.
Strong dominance graphs in Debian

let’s showcase some examples ...

Live data (all Debian releases + daily snapshots) available at http://www.mancoosi.org/measures/
Outline

1. Past
   - The EDOS project
   - Package dependencies: the formal way

2. Present
   - QA tools
   - The Mancoosi project
   - Fun with the Debian dependency graph

3. Future
Strong conflicts

- Like strong dependencies, **but with conflicts!**
- $a$ and $b$ conflict strongly iff they cannot be installed together

1591 ppmtofb-0.32:
1591 (python-2.4.4-2 <-> ppmtofb-0.32)
* python-osd-0.2.12-1.2 (conjunctive)
  - dependency: python-osd-0.2.12-1.2 -> python-2.4.4-2
  - conflict: python-2.4.4-2 - ppmtofb-0.32
* python-oss-0.0.0.20010624-3.3 (conjunctive)
  - dependency: python-oss-0.0.0.20010624-3.3 -> python-2.4.4-2
  - conflict: python-2.4.4-2 - ppmtofb-0.32
...

ppmtofb-0.32 has had **1591** strong conflicts, why?
- All caused by **one** explicit conflict
- In the metadata: conflict with python > 2.4
Better dependency solving

**completeness** each time a solution to an upgrade problem does exist, a meta-installer should be able to find it

**optimality** it should be possible to specify *optimization criteria* to discriminate among otherwise equivalent solutions, e.g.:

- minimize download size
- minimize used disk space
- minimize the number of sensitive package touched
- blacklist packages maintained by J. Random DD
- ...

**efficiency** dependency resolution should be as fast as possible
A dependency solver competition

We surely do not hope to find magically the silver bullet algorithm for dependency solving, but we can help the fate organizing a dependency solving competition

- real-life upgrade problem collected a-la popcon
- various *tracks*: plain resolution (speed), optimizing resolution (better solution), . . .
- developers and researchers can submit their implementations of their algorithms
- the winner gains fortune and glory

A distro-independent format to describe upgrade scenario has been developed: CUDF (Common Upgradeability Description Format)

- it can also be used to share dependency solver between package managers
- currently implemented in CUPT
Questions?

looking for something else than Q & A time?
...ok, here is some SPAM a friendly reminder: http://www.mancoosi.org