# Solving Package Dependencies:

## *from EDOS to Mancoosi*

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

Laboratoire PPS, Université Paris Diderot / The Debian Project

10 August 2008
DebConf8 — Mar Del Plata, Argentina






# The Challenge of Distributions

Debian, as other vendors, is meant to carry the burden of maintaining a free software distribution. It is a challenging task! (smooth upgrades, automatic dependency solving, up to date software . . . )

Help from: better infrastructure for *package maintainers* and better package managers for final users.

Two projects to the rescue:

EDOS [2004–2007] *aim*: provide FOSS distribution editors with better QA tools

Mancoosi [2008–2011] *aim*: provide better package managers to improve "upgrade" experiences

This talk gives *an overview of the EDOS and Mancoosi projects* and their relationships with Debian.

# Outline

# Intermezzo: EDOS/Mancoosi terminology

**installer** file-level package manager, e.g. `dpkg`

**meta-installer** repository-level package manager, e.g. `apt-get`, `aptitude`, ...

**package metadata** static information about a package, e.g. inter-package relationships declared in `debian/control`

# Outline

# 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 was not officially involved, even though 1 DD was enrolled as a researcher among the ranks of Paris 7. 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 and was split in several workpackages about the following subjects:

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

We will focus on (1): we were mostly involved in it, and it was the origin of most Debian-related results.

*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 and was split in several workpackages about the following subjects:

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

We will focus on (1): we were mostly involved in it, and it was the origin of most Debian-related results.

*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

# Which inter-package relationships?

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

Inter-package relationships (policy) and which concern EDOS:

- Depends
- ~~Recommends~~ — user-overridable
- ~~Suggests~~ — ignored by default
- Pre-Depends — $\approx$ Depends, different only for installer
- ~~Enhances~~ — ignored by default
- Conflicts
- ~~Breaks~~ — not available back then, installer-specific
- ~~Replaces~~ — installer-specific

# What is *an* inter-package relationship?

Each relationship among packages is something like:

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

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

$$\texttt{libc6} \wedge (\texttt{libice6} \vee \texttt{xlibs}) \wedge \ldots$$

What about version constraints?

*Given a package repository*:

- substitute each non-versioned package name for a disjunction of its available versions
- substitute each versioned package name for a disjunctions of all of its available versions *which satisfy the version constraint*

# What is *an* inter-package relationship?

Each relationship among packages is something like:

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

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

$$\texttt{libc6} \wedge (\texttt{libice6} \vee \texttt{xlibs}) \wedge \ldots$$

What about version constraints?
*Given a package repository*:

- substitute each non-versioned package name for a disjunction of its available versions
- substitute each versioned package name for a disjunctions of all of its available versions *which satisfy the version constraint*

# Repository expansion

Before reasoning about a repository, an expansion is performed

## Example (Version number expansion)

```
Package:      a
Version:      1
Depends:      b, c|d(>=2)

Package:      b
Version:      2

Package:      b
Version:      3

Package:      c
Version:      3
Conflicts:    b

Package:      d
Version:      1

Package:      d
Version:      2

Package:      d
Version:      3
```

```
Package:      a
Version:      1
Depends:      b(=2)|b(=3),
              c(=3)|d(=2)|d(=3)

Package:      b
Version:      2

Package:      b
Version:      3

Package:      c
Version:      3
Conflicts:    b(=2),b(=3)

Package:      d
Version:      1

Package:      d
Version:      2

Package:      d
Version:      3
```

# Repository expansion (cont.)

. . . the same can be done to handle virtual packages

- substitute a virtual package name for a disjunction of the (versioned) packages providing it

## Example (Virtual package expansion)

```
Package:      a
Provides:     v

Package:      b
Provides:     v
Depends:      w

Package:      c
Provides:     w
Conflicts:    w

Package:      d
Provides:     w
Conflicts:    w
```

```
Package:      a

Package:      b
Depends:      w

Package:      v
Depends:      a|b

Package:      c
Conflicts:    d

Package:      d
Conflicts:    c

Package:      w
Depends:      c|d
```

[ versions omitted for the sake of clarity ]

# What is a repository then?

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

1. a set of *packages* $P$
2. a function $D$ determining package *dependencies*
3. a set of *conflicts* $C$, i.e. pairs of non co-installable packages

## Example (modeling of the previously shown `Packages`)

$$
\begin{aligned}
P &= \{(a,1),(b,2),(b,3),(c,3),(d,1),(d,2),(d,3)\} \\
D(a,1) &= \{\{(b,2),(b,3)\},\{(c,3),(d,2),(d,3)\}\} \\
D(b,2) &= \emptyset \\
&\dots \\
C &= \{((b,2),(b,3)),((b,3),(b,2)),((c,3),(b,2)),\dots\}
\end{aligned}
$$

# Package installability as SAT

Based on the given formalization it is easy to show that the problem of whether a package is installable in a given repository is equivalent to SAT[1]

- each *package* $p$ (with version $v$) is interpreted as a *boolean variable* $p_v$ (`if` $p_v$ `then` the package should be installed `else` it should not)
- each *dependency* is interpreted as an *implication*, e.g.: $\texttt{aterm} \rightarrow \texttt{libc6} \wedge (\texttt{libice6} \vee \texttt{xlibs}) \wedge \ldots$
- each *conflict* between packages $a$ and $b$ is interpreted as the formula $\neg(a \wedge b)$

**Theorem**

*A particular package p, version v is installable iff there exist a boolean assignment that makes* $p_v$ `true`*, and satisfies the encoding of the repository.*

[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: 2.1.3-8
Depends: libc6 (>= 2.3.5-1)

Package: libdb1-compat
Version: 2.1.3-7
Depends: libc6 (>= 2.2.5-13)
```

becomes

$$
\begin{aligned}
& I^{2.3.2.ds1-22}_{\texttt{libc6}} \\
& \wedge \\
& \neg(I^{2.3.2.ds1-22}_{\texttt{libc6}} \wedge I^{2.2.5-11.8}_{\texttt{libc6}}) \\
& \wedge \\
& \neg(I^{2.3.2.ds1-22}_{\texttt{libc6}} \wedge I^{2.3.5-3}_{\texttt{libc6}}) \\
& \wedge \\
& \neg(I^{2.3.5-3}_{\texttt{libc6}} \wedge I^{2.2.5-11.8}_{\texttt{libc6}}) \\
& \wedge \\
& \neg(I^{2.1.3-7}_{\texttt{libdb1-compat}} \wedge I^{2.1.3-8}_{\texttt{libdb1-compat}}) \\
& \wedge \\
& I^{2.3.2.ds1-22}_{\texttt{libc6}} \longrightarrow \\
& (I^{2.1.3-7}_{\texttt{libdb1-compat}} \vee I^{2.1.3-8}_{\texttt{libdb1-compat}}) \\
& \wedge \\
& I^{2.1.3-7}_{\texttt{libdb1-compat}} \longrightarrow \\
& (I^{2.3.2.ds1-22}_{\texttt{libc6}} \vee I^{2.3.5-3}_{\texttt{libc6}}) \\
& \wedge \\
& I^{2.1.3-8}_{\texttt{libdb1-compat}} \longrightarrow I^{2.3.5-3}_{\texttt{libc6}}
\end{aligned}
$$

. . . average formula has 400 literals, KDE installation 32'000

# Good qualities for a repository

An installation is a subset of a repository packages; a healthy installation is one in which all packages have their dependencies installed (*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

# The EDOS toolchain

Several tools have been developed during EDOS ($\approx$ 110'000 OCaml LOCs), some examples:

edos-debcheck command line checker for package installability

pkglab interactive, console-based environment for repository inspection

ceve parser/converter between package list formats

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

Let's see some of them in more detail ...

# edos-debcheck

- `edos-debcheck` takes as input an APT package list (e.g. `/var/lib/apt/lists/*`) and checks whether one, several, or all packages in it are installable wrt that repository.

It is based on a customized SAT solver and it is quite 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
```

# edos-debcheck (cont.)

## Noteworthy success story

emdebian is using `edos-debcheck` *before* package uploads to ensure that the upload won't introduce package brokenness in the archive.
The path between upload and the archive in Debian can be significantly longer (e.g. NEW-processing), but a `dput` patch implementing pre-upload hooks is pending; using it `edos-debcheck` can be optionally used as a pre-upload sanity check.

Debian packages: `edos-debcheck`, `edos-rpmcheck`

# 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 (not possible with plain edos-debcheck)

Debian packages: dose2 (underlying library), ceve (package list parser/converter), pkglab (interactive environment).
All available in experimental/NEW.

# pkglab — examples

```
(* interactive equivalent of edos-debcheck *)

> $diag <- check($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>
```

# pkglab — examples (cont.)

```
(* 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>
```

# Finding uninstallable packages in Debian

`edos-debcheck` is used daily to monitor uninstallable packages in Debian, Skolelinux, and Debian GNU/kFreeBSD:

`http://edos.debian.net/edos-debcheck`

Most common cases of uninstallable packages:

1. autobuilders catching-up (e.g.: arch:all package uploaded together with arch:any packages + autobuilder delays): normal *transient* uninstallabilities
2. *a* depends on *b*, with *b* not available on all archs: either build problem with *b*, or *too liberal architecture specification* in *a* (should be stricter)
   - special case of the above: *a* is arch:all. A recent proposal of adding an Install-To field was meant to address this (#436733)
3. serious packaging bugs, you should really fix them :-)

# Uninstallable packages — examples

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

| Date | alpha | amd64 | arm | armel | hppa | i386 | ia64 | mips | mipsel | powerpc |
|---|---|---|---|---|---|---|---|---|---|---|
| 23/06 | 367(7) | 14(2) | 217(4) | 348(21) | 369(9) | 12(4) | 48(3) | 267(3) | 269(3) | 21(3) |
| Δ | +0/−0 | +0/−0 | +0/−1 | +0/−0 | +0/−0 | +0/−0 | +0/−0 | +0/−0 | +0/−0 | +0/−3 |
| 22/06 | 367(7) | 14(2) | 218(4) | 348(21) | 369(9) | 12(4) | 48(3) | 267(3) | 269(3) | 24(4) |
| Δ | +0/−0 | +0/−0 | +0/−0 | +0/−0 | +0/−0 | +0/−0 | +0/−0 | +0/−3 | +0/−3 | +0/−0 |
| 21/06 | 367(7) | 14(2) | 218(4) | 348(21) | 369(9) | 12(4) | 48(3) | 270(4) | 272(4) | 24(4) |
| Δ | +0/−0 | +0/−3 | +0/−3 | +0/−9 | +0/−0 | +0/−0 | +0/−0 | +0/−0 | +0/−0 | +0/−0 |
| 20/06 | 367(7) | 17(3) | 221(5) | 357(24) | 369(9) | 12(4) | 48(3) | 270(4) | 272(4) | 24(4) |
| Δ | +7/−0 | +3/−0 | +4/−3 | +3/−27 | +4/−0 | +3/−0 | +3/−0 | +5/−11 | +5/−0 | +5/−0 |
| 19/06 | 360(5) | 14(2) | 220(6) | 381(31) | 365(8) | 9(3) | 45(2) | 276(2) | 267(2) | 19(2) |
| Δ | +0/−0 | +0/−0 | +0/−0 | +0/−0 | +0/−0 | +0/−0 | +0/−0 | +0/−0 | +0/−0 | +0/−0 |
| 18/06 | 360(5) | 14(2) | 220(6) | 381(31) | 365(8) | 9(3) | 45(2) | 276(2) | 267(2) | 19(2) |
| Δ | +0/−0 | +0/−0 | +0/−0 | +0/−0 | +0/−0 | +0/−0 | +0/−0 | +0/−0 | +0/−0 | +0/−0 |
| 17/06 | 360(5) | 14(2) | 220(6) | 381(31) | 365(8) | 9(3) | 45(2) | 276(2) | 267(2) | 19(2) |

The "Debian weather" for 27 June 2008: mostly sunny in stable and testing, at places overcast and rainy in unstable.

| | |
|---|---|
| clear | $< 1\%$ |
| few clouds | $1\% \ldots 2\%$ |
| clouds | $2\% \ldots 3\%$ |
| showers | $3\% \ldots 4\%$ |
| storm | $> 4\%$ |

Stable:

Testing:

Unstable:

alpha amd64 arm hppa i386 ia64 mips mipsel powerpc

# Finding undeclared `Conflicts` in Debian

```
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
```

. . . get rid of these *before they reach our users*.

0. naively: try co-installing together all package pairs (200'000'000) . . . no way!
1. only consider pairs sharing at least one file (easy using `Contents`): 867 pairs (16 April 2008, amd64/sid)
2. restrict to pairs co-installable according to dependencies (easy using `pkglab`): 102 pairs
3. still diversion can account for false positives: test pair installations in chroot: 27 buggy package pairs detected

Reports: http://edos.debian.net/missing-conflicts/
BTS: user `treinen@debian.org`, tag `edos-file-overwrite`

# Outline

# The Mancoosi project [ http://www.mancoosi.org ]

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 DDs are enrolled as researchers among the ranks of Paris 7

# The upgrade problem

Upgrade problem = the problem posed by a meta-installer request of changing 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 impredictable 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!

we will focus on dependency solving, as we will mostly be involved in it

# The upgrade problem

Upgrade problem = the problem posed by a meta-installer request of changing 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 impredictable 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!

we will focus on dependency solving, as we will mostly be involved in it

# Desiderata for dependency solving

**completeness** each time a solution to an upgrade problem does exists, 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
- 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 (opt-in, data collector plug-ins in meta-installers)
- 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

Similar competitions have proven fruitful to push state of the art in related fields, such as SAT solving itself, why this one shouldn't?

# Submitting problems for the competition

We are standardizing submission formats to contribute upgrade problems for the competition. Each participating distribution will have its own submission format (DUDF), to be converted in a common format later on (CUDF).

## sample submission for Debian's apt

1. `/var/lib/dpkg/status` (excerpt of)
2. `/var/lib/apt/lists/*` (checksums of)
3. the given APT command
4. current APT conf (repositories, pinning, ...)
5. "debian", "apt-get", v*x.y.z*, "dpkg" (tool identifiers)
6. "broken packages, the following packages can not ..." (outcome)

Additionally, the submission format can be useful for bugreports against package managers.

# Debian and Mancoosi

Foreseeable contact points between Debian and Mancoosi:

- common meta-installer ABI for pluggable solvers
- the competition: Debian meta-installers participate as legacy tools, and smart optimization ideas can travel a long way . . . submit yours!
- state logging in meta-installers: we will develop state logging for at least one meta-installer, integration and additional extra implementations will be needed

## Contact us!

debian@mancoosi.org

http://mancoosi.debian.net

(or ping/query/mail/. . . directly *Ralf* and *Zack*)

looking for more info about EDOS/Mancoosi?
a good starting point is the paper accompanying this talk: look it up on Penta!

# Questions?

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

# Cheers!

*the Mancoosi team, Feb 2008*