# Cross-Distro Dependency Resolution

## Reusing Solvers among Distributions

**Stefano Zacchiroli** Ralf Treinen
zack@{pps.jussieu.fr,debian.org}
http://upsilon.cc/zack

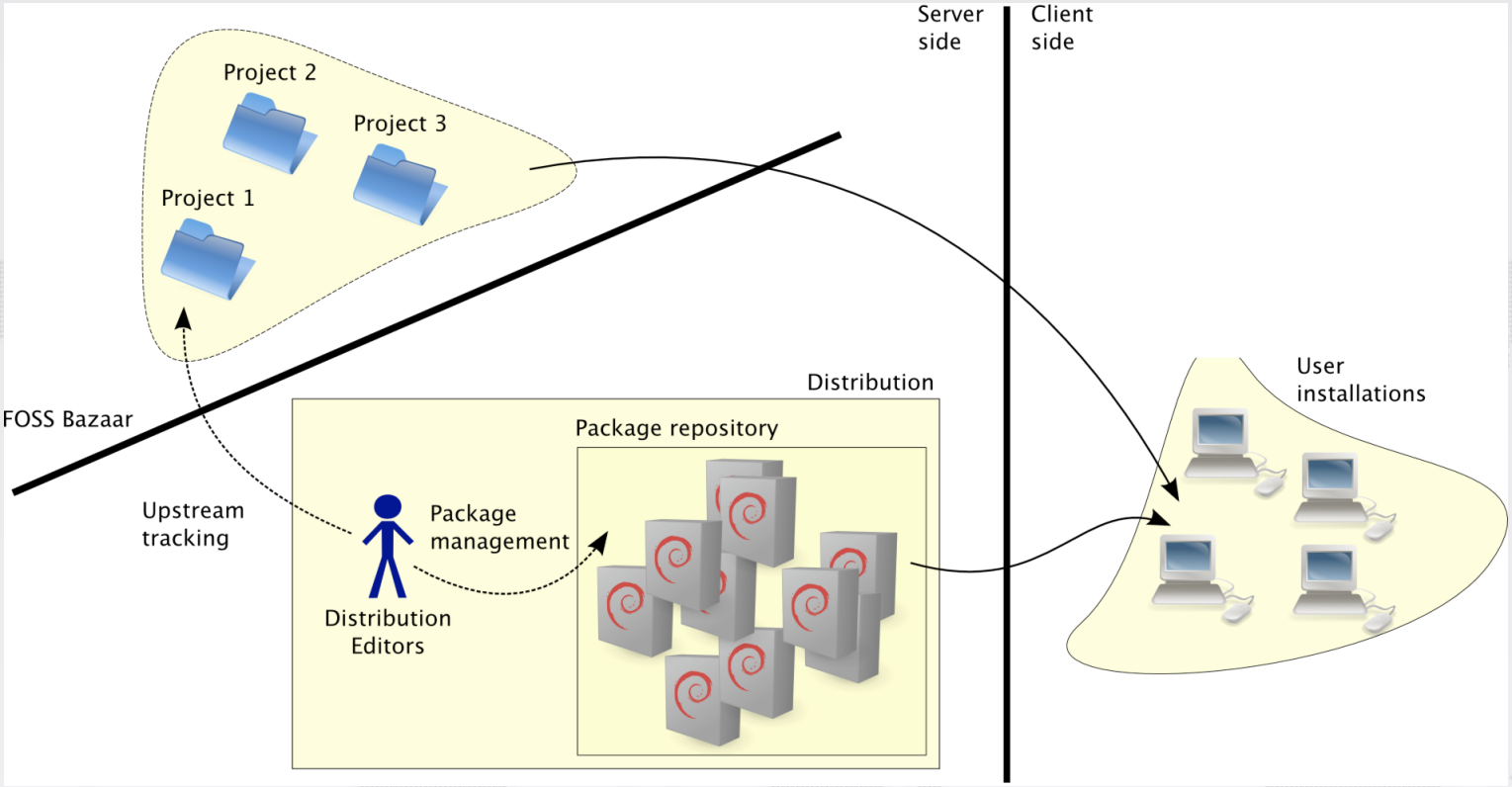
PPS, University Paris Diderot
The Debian Project

6 August 2010
DebConf10, New York City

mancoosi
managing software complexity

# Outline

# The notion of “distribution”: why we do what we do




- distributions are meant to ease software management
- offer coherent software collections (e.g. policies)
- key idea: the abstraction of package
- killer apps: **package managers** (deb/rpm war anyone?)

# What is the *role* of package managers, then?

making easy and flexible software “upgrades”
(i.e. install/remove/upgrade packages)

1. abstract over package retrieval
   - avoid manual downloads, enforce trust paths, . . .
2. low-level deployment on disk (dpkg-/rpm-level)
   all fancy features:
   - triggers, transactions
   - conffile management
   - diversions / alternatives
   - . . .
3. dependency solving
   - install recursively missing dependencies
   - spot conflicts
   - compute upgrade paths (e.g. among distro releases)

# What is the *role* of package managers, then?

making easy and flexible software “upgrades”
(i.e. install/remove/upgrade packages)

1. abstract over package retrieval
   - avoid manual downloads, enforce trust paths, . . .
2. low-level deployment on disk (dpkg-/rpm-level)
   all fancy features:
   - triggers, transactions
   - conffile management
   - diversions / alternatives
   - . . .
3. dependency solving ⟵ **Is it as good as we want?**
   - install recursively missing dependencies
   - spot conflicts
   - compute upgrade paths (e.g. among distro releases)

# Dependency solving issues

bold statement:

**dependency solving is not (yet) good enough!**

main issues:

1. incompleteness
2. poor expressivity
3. not so easy to implement

# Dependency solving issues

bold statement:

**dependency solving is not (yet) good enough!**

main issues:

1. incompleteness

## Example

```
package:  a            package:  b
version:  2            version:  2
depends:  b (= 1)      depends:  a (= 1)

package:  a            package:  b
version:  1            version:  1
depends:  b            depends:  a
```

“# apt-get install a b” fails, how about yours?

# Dependency solving issues

bold statement:

**dependency solving is not (yet) good enough!**

main issues:

1. incompleteness
2. poor expressivity *(AKA policies)*

does your package manager enable you (sysadm) to:

1. minimize installed size *cool for embedded*
2. minimize download size *28.8 Kbps connections*
3. blacklist package maintained by *Joe Random Developer*
4. *name yours worst desire that pinning does not fulfill . . .*

# Dependency solving issues

bold statement:

**dependency solving is not (yet) good enough!**

main issues:

1. incompleteness
2. poor expressivity
3. not so easy to implement *(engineering problem)*
   - NP-complete problem, after all
   - naive implementations have been shown to be either too naive or explosive (looping)
   - “new generation” package manager developers *welcome* reusing dependency solving logics

# Let's do what we do best: share efforts

how about reusing dependency solvers?

1. within distributions (different package managers have different solvers, some times even more than one per package manager, which one a buildd will use?)
2. across distributions (both among "same world" distros and "different worlds" distros)
3. with the scientific community and companies which are already doing solving (they are interested in our data)

## end goal

- ~~"standard" package format~~
- 1. find "the" solver / algorithm / technique / paradigm that works best for our *common goals*
  2. deploy it in each package manager (or in a reusable lib)

# CUDF: a common interchange format

CUDF *Common Upgradeability Description Format*: a file format to describe upgrade scenarios

package universe all known packages

package status the set of currently installed packages

user request desired change to the package status

## abstraction challenges (deb vs rpm worlds)

- version numbers
- package relationships (depends/recommends/obsoletes/...)
- lexical conventions (e.g. package names)
- virtual packages and their dependencies
- multiple version installation vs singleton

# CUDF: file format

- plain text file format
- inspired by RFC 2822 (easy on the eyes and to parse)
- list of empty-line-separated stanzas
- each stanza: typed key-value pairs

```
package: m4
version: 3
depends: libc6 >= 8

# this is a comment ...
package: openssl
version: 11
depends: libc6 >= 18, libssl0.9.8 >= 8,
 zlib1g >= 1        # this is a line continuation
conflicts: ssleay < 1
```

# CUDF: types

integers . . . -2, -1, 0, 1, 2, . . .

positive integers 1, 2, 3, . . .

booleans true, false

package names regexp ^[a-zA-Z0-9+./@()%-]+$
libc6, libdb4.6, libc-dev, /bin/bash

package formulae over versioned package predicates

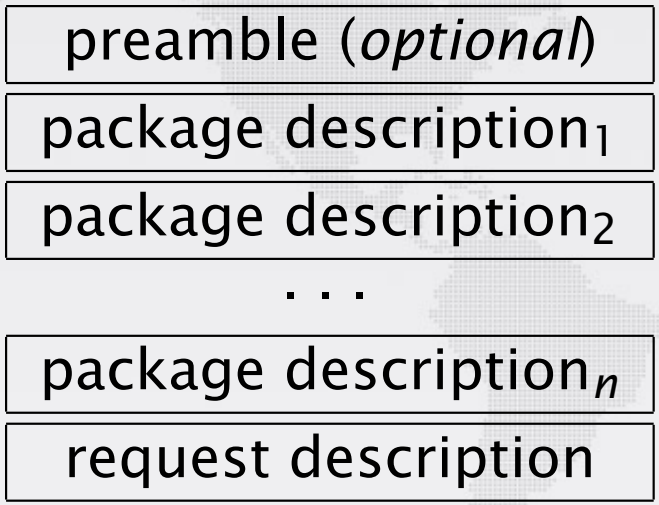
- python-minimal
- libedac1 = 1
- haskell-doc <= 2
- libz-dev != 3
- postfix > 2 | exim4-base
- ocaml-nox, libc6 >= 6
- php4, apache | httpd

package lists degenerate formulae w/o disjunctions (i.e. "|")

# CUDF: stanzas

A CUDF document is made of several stanzas:

1. one (optional) preamble stanza for meta-data
2. one stanza for each known package
3. one stanza for the user request




```
preamble:
# ...

package: foo
# all known packages ...

package: bar
# ...

request:
# what the user has asked for?
# ...
```

# CUDF: package stanzas

Each package stanza describes *a package known to the package manager*

Legacy package properties:

```
package: ...   (mandatory; type: package name;
                must be the 1st in the stanza)
version: ...   (mandatory; type: positive integer)
installed: ... (optional; default: false; type: bool)
depends: ...   (optional; type: package formula)
conflicts: ... (optional; type: package list)
provides: ...  (optional; type: package list)
```

```
package: gasoline-engine
version: 1
depends: turbo
provides: engine
conflicts: engine, gasoline-engine
installed: true
```

# CUDF: package highlights

1. versions are positive integers
   - usual version strings "1.2.3-4" are not accepted (no clear cross-distro semantics)
   - each set of versions in a distro has a total order → can be easily mapped to positive integers

# CUDF: package highlights

1. versions are positive integers
2. **provides:** account for features / virtual packages
   - features are versioned, you can **provides:** `httpd > 2`
     unversioned features provide *all possible versions*

# CUDF: package highlights

1. versions are positive integers
2. **provides**: account for features / virtual packages
3. conflicts are not implicit among different versions of the same package
   - multiple versions of the same package are co-installable
   - ...but self-conflicts are ignored

```
package: bash
version: 5
conflicts: bash
# i.e. conflict with all other versions of foo
```

# CUDF: package highlights

1. versions are positive integers
2. **provides**: account for features / virtual packages
3. conflicts are not implicit among different versions of the same package
4. self-conflicts on virtual packages are ignored, too

```
package: postfix                    # mutual exclusion
version: 2
provides: mail-transport-agent
conflicts: mail-transport-agent

package: exim
version: 3
provides: mail-transport-agent
conflicts: mail-transport-agent
```

# CUDF: extra properties

Extra properties are permitted for packages, e.g.:

- download-size: posint
- installed-size: posint
- maintainer: string
- security-fix: bool
- priority: enum[essential, important]
- suite: enum[stable, testing, unstable]

Extra property must be typed and declared in the preamble

```
preamble:
property: suite: enum[stable,testing,unstable] = [stable]
property: bugs: int = 0
property: pin-priority: int
```

# CUDF: request stanza

```
request: ...    (mandatory; type: string;
                 just a delimiter)
install: ...    (optional; type: package list)
remove: ...     (optional; type: package list)
upgrade: ...    (optional; type: package list)
```

- install/remove/upgrade: which packages must be installed/removed/upgraded.
  - version requirements can be specified, e.g. **install:** bash > 3 request installation of a version of bash greater than 3
- all upgraded packages
  1. must have a single version installed in the solution proposed by the solver
  2. cannot be downgraded to a version strictly smaller to the one that was previously installed

# CUDF: putting all together

```
package: car
version: 1
depends: engine, wheel, door, battery
installed: true

package: bicycle
version: 7

package: gasoline-engine
version: 1
depends: turbo
provides: engine
conflicts: engine, gasoline-engine
installed: true

package: gasoline-engine
version: 2
provides: engine
conflicts: engine, gasoline-engine

package: electric-engine
version: 1
depends: solar-collector | huge-battery
provides: engine
conflicts: engine, electric-engine

# ...

request: source: Debian/CUDF 733963bab9fe1f78fd551ad20485b217
install: bicycle, electric-engine = 1
upgrade: door, wheel > 2
```

# CUDF specifications

CUDF

http://www.mancoosi.org/cudf/

A full-fledged specification of CUDF is available. Extras:

- detailed (formal) semantics: useful to double-check implementations
- DUDF a related format to collect distro-specific upgrade scenario information *a la popcon*
  - offspring usage: collect package manager bug reports
- a CUDF primer is available as well
  http://www.mancoosi.org/cudf/primer/

Forthcoming changes:

- support for multiarch and locally rebuilt packages

# libCUDF

libCUDF is a reference implementation of CUDF

- CUDF parsing
  - of CUDF documents (packages + problem)
  - of solutions
- consistency checking *is the package status healthy?*
- solution checking *does the solution fulfill user request?*

Code:

- OCaml library + command line checker
- C bindings available (fully hiding OCaml)
- LGPL
- `http://www.mancoosi.org/cudf/`
  Debian and RPM packages available

# cudf-check

```
# ./cudf-check -univ examples/universe.cudf
parsing package universe ...
installation status consistent
```

```
# ./cudf-check -univ examples/universe-broken.cudf
parsing package universe ...
installation: broken (reason: Cannot satisfy
  dependencies turbo of package gasoline-engine
  (version 1))
```

```
# ./cudf-check -cudf examples/legacy.cudf
parsing CUDF ...
installation status consistent
```

# cudf-check (cont.)

```
# ./cudf-check -cudf examples/legacy.cudf \
        -sol examples/legacy-sol.cudf
loading CUDF ...
loading solution ...
installation status consistent
is_solution: true
```

```
# ./cudf-check -cudf examples/legacy.cudf \
        -sol examples/legacy-sol-bad.cudf
loading CUDF ...
loading solution ...
installation status consistent
is_solution: false (reason: Unmet installation request,
  missing packages: bicycle)
```

# CUDF: deployment and other implementations

CUDF is being deployed in various places.

In Debian:

- ongoing discussion on `deity@lists.debian.org`
  - 1st goal: share solvers *within* Debian
- CUDF support in: CUPT
- forthcoming in: APT, APT2

Elsewhere:

- CUDF support in: URPMi
- forthcoming in: RPM5

# CUDF: deployment and other implementations

CUDF is being deployed in various places.

In Debian:

- ongoing discussion on `deity@lists.debian.org`
  - 1st goal: share solvers *within* Debian
- CUDF support in: CUPT
- forthcoming in: APT, APT2

Elsewhere:

- CUDF support in: URPMi
- forthcoming in: RPM5

Next:

- . . . your favorite package manager?

# . . . a solving competition?

As part of the Mancoosi project we have run the 1st edition of a solving competition. Goals:

1. collect a (CUDF) corpus of real-life upgrade scenarios
2. make the scientific community aware of our needs
3. find a 1st reusable common-ground dependency solver

held @ `http://lococo2010.mancoosi.org`

## MISC competition

- 2 tracks: “trend” & “paranoid”
- 11 participants (total over the 2 tracks)
- http://www.mancoosi.org/misc-2010/

# MISC: paranoid track

Satisfy my request, but by changing as less as possible my system !

1. minimize the number of removed packages
2. minimize the number of changes in the installation status (to avoid gratuitous installation of new packages)

# MISC: trendy track

Satisfy my request, and if possible update packages to their most recent version and install recommended packages!

1. minimize the number of removed packages
2. minimize the number of packages not in the most recent version
3. minimize the number of not satisfied recommendations of installed packages
4. minimize the number of newly installed packages.

# The problem classes

Five categories of problems have been used for the 2010 competition.

| | |
|---|---|
| cudf_set | Encoding in cudf of 1-in-3-SAT |
| debian-dudf | Real installation problems, Problems collected via dudf-save. |
| easy | Debian unstable / desktop installation from unstable / 10 install - 10 remove |
| difficult | Debian stable + unstable / server installation from stable / 10 install - 10 remove - 1 upgrade all |
| impossible | Debian oldstable, stable, testing, unstable / server installation from oldstable / 10 install - 10 remove - 1 upgrade all |

# The participants of MISC 2010

- aspcud - Answer Set Programming
- INESC - a SAT-based solver, MaxSAT
- apt-pbo - APT + minisat
- UCL
- P2 - SAT, Eclipse
- UNSA

# The participants of MISC 2010

- aspcud - Answer Set Programming
- INESC - a SAT-based solver, MaxSAT
- apt-pbo - APT + minisat
- UCL
- P2 - SAT, Eclipse — 2nd
- UNSA — 1st

# Questions & discussion

## Questions?

zack@debian.org http://www.mancoosi.org/

**PS** we welcome suggestions on exciting *policies* that dependency solving should enable:

- minimize installed size — *cool for embedded*
- minimize download size — *28.8 Kbps connections*
- blacklist package maintained by *Joe Random Developer*
- . . . what else? Share your thoughts!