Leveraging solver preferences to tame your package managers

Roberto Di Cosmo (I+i+D), Pietro Abate (D), Stefano Zacchiroli (i+D), Fabrice Le Fessant (i+I), Louis Gesbert (O)
and also Jérôme Vouillon (i+D), Ralf Treinen (i+D)

Université Paris (D)iderot, (I)NRIA, (i)rill, and (O)CamlPro

September 5th, 2014
OCaml 2014

OCaml PRO

Today: *an aspect* of our work on OPAM at Irill

Ten years of research on package management ...

- two european projects: EDOS and MANCOOSI (www.mancoosi.org)
- bridging research communities, dependency solver competition
- beautiful results, stable and efficient OCaml tools and libraries

... used as foundation of the ~~ocp-get~~ OPAM package manager

libcudf CUDF manipulation library: `opam show cudf`

dose Mancoosi toolbox: `opam show dose`

Relevant literature for this talk


Di Cosmo, Leroy, Treinen, Vouillon et al *Managing the complexity of large FOSS package-based software distributions.* ASE 2006


Abate, Di Cosmo, Treinen, Zacchiroli *Dependency solving: a separate concern in component evolution management.* Journal of Systems and Software, 2012.


Abate, Di Cosmo, Treinen, Zacchiroli *A modular package manager architecture.* Information and Software Technology, 2013.

Package managers are all around us

Definition of Package Manager (Wikipedia)

... tools to automate ... *installing, upgrading, configuring, and removing* software packages ... *in a consistent manner*. It typically maintains a database of software *dependencies* and *version* information ...

Some package managers

Binary distributions `apt`, `aptitude`, `yum`,...

Source distributions `portage`, `*BSD ports`, `homebrew`, **opam**...

Language specific `PyPI`, `Eclipse P2`, `(opam)`, ...

Application specific `steam`, ...

Decentralised `0install`, ...

Functional approach `nixOS`, `disnixOS`,...

... you name it

Architecture of a package management system

Many distinct tasks are performed in a package management system:

Server side

Maintain a *coherent* set of packages when we add, build, remove, update packages. This includes spotting packages that are no longer installable (or co-installable), due to *dependency issues*.

Client side

fetch and autenticate metadata and packages

dependency solver to find a solution to dependency constraints

user preferences to pick the right solution

deploy the chosen solution

Focus on two aspects that are really common to all

Dependency solving: how hard is it?

Theorem

The following problems are NP-complete:

- *installability of a single package*
- *co-installability of a set of packages*

Proof idea

Equivalence of dependency resolution and boolean satisfiability.

Di Cosmo, Leroy, Treinen, Vouillon et al. *Managing the complexity of large FOSS package-based software distributions*. ASE 2006.

Alternative proof of NP-hardness (Daniel Burrows, 2008)

Encode Sudokus as a package installation problem, left as an exercise

Application: find uninstallable packages *in a repository*

Basic idea

- package installations can be encoded as Boolean Satisfiability problems
- just call a SAT solver on each package in the repository (may be tens of thousands! is this a bad idea?)

Good news: it's feasible in practice

- modern SAT solvers perform well on practical instances
- Jérôme Vouillon's specialised solver, now part of the `dose` library, is engineered to check installability of tens thousands of packages in a few seconds

Debian: this technology is in use since 2006

qa.debian.org/dose (wrapper around dose-distcheck)

QA

About Debian Getting Debian Support Developers' Corner

debian / debian quality assurance

Packages not installable in scenario unstable_main

In a pair *n*/*m*, *n* is the number of packages that are build for that architecture, *m* is the number of packages with Architecture=all.

Summary for the last 7 days

Date	amd64	arm64	armel	armhf	hurd-i386	i386	kfreebsd-amd64	kfreebsd-i386
Today's Weather:								
2014-09-01 05:00:03	123/76	239/4516	133/230	123/146	891/2270	125/78	130/582	132/576
Diff	+4/1 -6/0	+45/1 -9/108	+4/1 -4/0	+4/1 -4/0	+9/1 -4/0	+4/1 -4/0	+6/20 -6/0	+6/20 -4/0
2014-08-31 05:00:05	125/75	203/4623	133/229	123/145	886/2269	125/77	130/562	130/556
Diff	+7/0 -1/0	+0/1 -13/91	+7/0 -1/0	+7/0 -1/0	+5/1 -6/2	+7/0 -1/0	+7/0 -1/0	+7/0 -1/0
2014-08-30 05:00:03	119/75	216/4713	127/229	117/145	887/2270	119/77	124/562	124/556

mancoosi
managing software complexity

Opam Weather Service: online since mid 2014

ows.irill.org (variant of dose-distcheck)

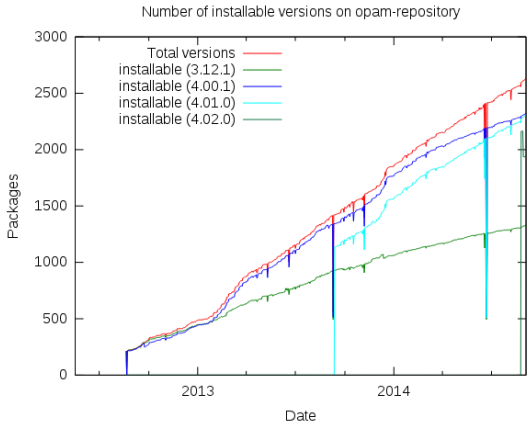
ows.irill.org/index.html

OPAM Weather Service

This service runs reguat regular intervals an evaluation of the solvability of dependencies in the official OPAM repository, for 4 stable releases of OCaml. For each version, a report is generated, showing the packages that cannot be installed because of problems in their dependencies (either a missing package, or a conflict between packages).

Latest report generated on 2014/09/04 at 14:02 (UTC)

- For version 3.12.1
- For version 4.00.1
- For version 4.01.0
- For version 4.02.0
- Summary table



ows.irill.org/table.html

async_parallel	109.23.00		ok	bad	bad
	109.27.00		ok	bad	bad
	109.30.00		ok	ok	bad
	109.34.00		ok	bad	bad
	109.35.00		ok	ok	bad
	109.40.00		ok	ok	bad
	109.41.00		ok	ok	bad
	109.47.00		ok	ok	bad
	109.53.00		ok	ok	bad
	109.53.02		ok	ok	bad
	109.58.00		ok	ok	bad
	109.58.01		ok	ok	bad
	111.17.00		bad	ok	ok
	111.25.00		bad	ok	ok
	111.28.00		bad	ok	ok
async_shell	109.14.00	bad	ok	bad	bad
	109.15.00	bad	ok	bad	bad
	109.17.00	bad	ok	bad	bad
	109.28.00	bad	ok	ok	bad
	109.28.02		ok	ok	bad
	109.28.03		ok	ok	ok
async_smtp	109.38.alpha1	bad	ok	ok	ok
async_ssl	111.06.00		ok	ok	bad
	111.08.00		ok	ok	ok
	111.21.00		ok	ok	ok
async_unix	108.00.01	bad	bad	bad	bad
	108.00.02	ok	bad	bad	bad
	108.07.00	ok	bad	bad	bad
	108.07.01	ok	bad	bad	bad
	108.08.00	ok	bad	bad	bad
	109.07.00	bad	ok	bad	bad
	109.08.00	bad	ok	bad	bad
	109.09.00	bad	ok	bad	bad
	109.10.00	bad	ok	bad	bad

A word of warning

Priceless if we have a proper *QA process* in place, which *we have not*.

Finding a needle in a haystack

Finding *a* solution is NP-complete, but installing and upgrading is more demanding... *how many* ways are there to install a package?

Too many *upgrade* candidates

Suppose we have components q_i, for $1 \leq i \leq n$, available in versions 1 and 2, all of which are installed in version 1 on the system.
We want to install a component p in version 1 that depends on all of $q_1, \ldots, q_n$, any version.
Any of the 2^n configurations $\{(p,1)\} \cup \{(q_i,i) | i \in 1 \ldots n, 1 \leq i \leq 2\}$ is a solution.

Which one do we choose?

paranoid only install p

trendy install p and step up all q_i's to version 2

... and exponentially many in between!

An exponential number of solutions???

The *sidestep* approach

centralize and group : *patch tuesday*, *service pack*, ...

isolate on a monolith : AppStore(s), ...

coexist : NixOS, *backward compatibility policy*, *no conflicts policy*, ...

Various ad-hoc algorithms

implement a *fixed* strategy for selecting a solution, e.g.
identify "suites" of components (e.g. *stable*, *testing*, *unstable*), let the user order them, and then try to stick to this order

Advantage tries to align the system with a chosen suite

Disadvantage
- depends on the quality of the most recent state: not assured for development versions, or when mixing repositories
- difficult and cumbersome to obtain a different behaviour
- when things go wrong, we may *really* get lost

Installation woes: debatable solution

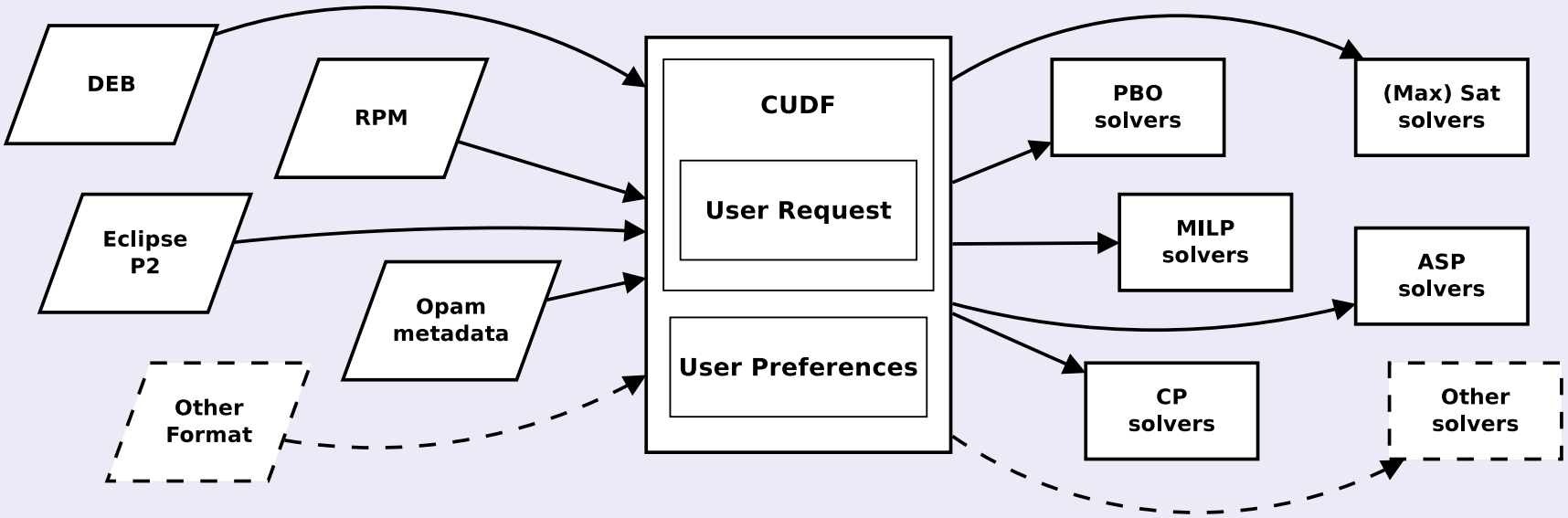
```
# sudo apt-get install  debhelper
Reading Package Lists... Done
Building Dependency Tree... Done
The following extra packages will be installed:
 armagetron armagetron-common autoconf bonobo-activation codebreaker debconf
 debconf-i18n debconf-utils dialog esound-common fb-music-high fontconfig
 frozen-bubble-data grepmail gv intltool-debian libaiksaurus-data
 libaiksaurus0c102 libatk1.0-0 libatk1.0-dev libbonobo-activation4 libbonobo2-0
 libbonobo2-common libdb3 libdbd-mysql-perl libdbi-perl libeel2-data libesd0
 ...
The following packages will be REMOVED:
  autoconf2.13 frozen-bubble frozen-bubble-lib gconf2 gnomemeeting itk3.1-dev
  libbonoboui2-0 libbonoboui2-common libdigest-md5-perl libforms0.89 libgconf2-4
  libgnome2-0 libgnome2-common libgnomeui-0 libgnomevfs2-0 libgnomevfs2-common
  libgtk1.2-dev libgtk2.0-0png3 libgtk2.0-dev libmime-base64-perl
  libpango1.0-dev libsdl-mixer1.2-dev libsdl-perl libsdl-ttf1.2-dev
  libsdl1.2-dev libsmpeg-dev libstorable-perl nautilus tk8.3-dev tktable-dev
  x-window-system x-window-system-core xaw3dg-dev xlib6g xlib6g-dev xlibmesa-dev
  xlibmesa3 xlibosmesa3 xlibs-dev xlibs-pic xpdf xpdf-reader
The following NEW packages will be installed:
  armagetron-common debconf-i18n fb-music-high fontconfig intltool-debian
  libaiksaurus-data libaiksaurus0c102 libeel2-data libfilehandle-unget-perl
  libfontconfig1 libforms1 libgdbm3 libgnutls7 libgsf-1 libice-dev libice6
  libidl0 liblzo1 libmagick5.5.7 libmail-mbox-messageparser-perl
  libmysqlclient12 libncursesw5 libnet-daemon-perl libnewt0.51 libpaper1
  libplrpc-perl libsdl-console ...

75 packages upgraded, 80 newly installed, 42 to remove and 858  not upgraded.
Need to get 67.1MB of archives. After unpacking 26.9MB will be used.
Do you want to continue? [Y/n] Abort.
```

Towards modular package managers

Dependency solving is NP-hard: stop coding a petty solver for every new component based system, and adopt a *modular* approach!

1 - Use a Common Upgrade Description Format



2 - Provide means for expressing our choice

A full fledged *user preferences* language to guide the solver towards our preferred solution.

1 - An excerpt from a CUDF file

```
preamble:
property: opam-version: string, opam-name: string

package: herelib
version: 3
depends: ocamlfind
conflicts: herelib
opam-name: herelib
opam-version: 109.12.00

...

package: lwt
version: 6
depends: base-threads , base-unix , camlp4 , ocamlfind
conflicts: react >= 3 , lwt
opam-name: lwt
opam-version: 2.4.4

...

request: opam
install: tyxml
```

2 - User preferences

A *preference expression* is built from four basic ingredients:

package selectors denote in a proposed solution certain classes of packages (the ones that changed, the ones that got removed, etc.)

measurements can be appliead to a package selector to obtain an integer value (the number of package selected, the number of packages selected that are up-to-date, etc.)

maximisation/minimisation directives to ask the solver to find a solution that maximises or minimises the value of a measurement

aggregation combinations can be used to ask the solver to combine criteria in lexicographical order

For full details, see

`www.dicosmo.org/Articles/usercriteria.pdf`
`www.mancoosi.org/misc-2012/criteria/`
`opam.ocaml.org/doc/Specifying_Solver_Preferences.html`

User preferences: package selectors

solution	packages installed in the solution proposed by the solver
down, up	packages downgraded or upgraded
removed, new	packages no longer there (removed) or not present before (new)
changed	packages changed (an aggregation of the above two lines)
request	packages *explicitly mentioned* in the *user request* ...
installrequest, upgraderequest	... for installation or upgrade

User preferences: measurements

count(X) : number of packages in X

sum(X,f) : sum of values of key f over packages in X

notuptodate(X) : number of packages in X not current

unsat_recommends(X) : number of unsatisfied clauses in the `recommends` field of packages from X

aligned(X,g1,g2) : number of packages aligned according to given criteria

(see the full documentation for examples and explanations)

Optimising and combining preferences

Optimisation

We can ask for a solution that maximizes (+) or minimizes (−) each of these criteria, e.g.:

```
-count(removed)
```

specifies that we want a solution where the number of removed packages is minimised.

Aggregation

We can combine criteria in lexicographic order, e.g.

```
-count(removed),-count(changed)
```

specifies that *among all solutions where the number of removed packages is minimised*, we look for one that has the *smallest number of changes*.

Examples preferences when installing your packages

Global focus

paranoid `-count(removed),-count(changed)`

trendy `-count(removed),-notuptodate(solution),-count(new)`

Drawback

May upgrade *all* your packages, when you only wanted to change *a few*

Local focus

paranoid

```
-count(removed),-notuptodate(request),-count(down),-count(changed)
```

trendy

```
-notuptodate(request),-count(removed),-count(down),-count(changed)
```

Drawback

may *use* not uptodate versions of dependecies of the request

Examples preferences when installing your packages, cont'd

Local focus, alternative

paranoid `-count(removed),-notuptodate(request),-notuptodate(changed),...`

trendy `-notuptodate(request),-count(removed),-notuptodate(changed),...`

Drawback

may leave untouched *existing* not uptodate versions of dependecies of the request

Bottomline

there is no "one size fits all" solution

help us design a set of *profiles* with an intuitive meaning and a well defined rationale

More exotic examples

Performing upgrades

upgrade `-count(down),-count(removed),-notuptodate(solution),-count(new)`

priority `-count(down),-count(removed),-notuptodate(solution),`
`+sum(solution,priority),-count(new)`

Building systems

Minimal system size `-sum(solution,installedsize),-count(solution)`

Noah's ark `+count(solution)`

Noah's ark, fresh `-notuptodate(solution),+count(solution)`

Fast bootstrap `-sum(solution,compiletime)`

More exotic examples, cont'd

Repairing a broken system configuration

Use an empty request with

fixup simple `-count(changed)`

fixup trendy `-count(changed),-count(down),-notuptodate(solution)`

Did you notice?

All of this requires zero changes to the package manager code!

Available external solvers

Three external CUDF solvers packaged in Debian

```
$apt-cache search cudf

aspcud - CUDF solver based on Answer Set Programming
mccs - multi-critera CUDF solver
packup - CUDF solver based on pseudo-Boolean constraints
```

There is also a nice solver for Java addicts

p2cudf, based on the Eclipse P2 plugin dependency resolver, available from http://wiki.eclipse.org/Equinox/p2/CUDFResolver

They do not all support the full language of preferences: aspcud version 1.9 or later is recommended

You can use all this in `opam` for OCaml packages!

External solvers in opam

The solver aspcud is supported out of the box in opam since 1.0,and in 1.2 typing `opam --help` shows

```
...
OPTIONS
  --criteria=CRITERIA
      Specify user preferences for dependency solving for this run.
      Overrides both $OPAMCRITERIA and $OPAMUPGRADECRITERIA.
      For details on the supported language, see
      http://opam.ocaml.org/doc/Specifying_Solver_Preferences.html.
      The default value is -count(down),-count(removed),
      -notuptodate(solution),-count(new) for upgrades, and
      -count(removed),-notuptodate(request),-count(down),
      -notuptodate(changed),-count(changed),-notuptodate(solution)
      otherwise.

  --cudf=FILENAME
      Debug option: Save the CUDF requests sent to the solver to
      FILENAME-<n>.cudf.

  --solver=CMD
      Specify the name of the external dependency solver. The default
      value is aspcud
...
```

Getting your external solver

Debian/Ubuntu user?

Lucky guy! Just `apt-get install aspcud`, and you are done

No aspcud ¿= 1.9 for you?

- go to `http://cudf-solvers.irill.org`
- follow the instructions
- access the *Irill CUDF solver farm*
- get your solving done *in the cloud*

Big thanks to OCamlPro folks (Benjamin,Fabrice,Gregoire,Pierre), for help setting this up.
Only real solution for low-power arch (arduino, raspberri-pi)

Absolutely need to work offline?

Try `0install` ... and if it is not enough, get the `Dockerised version` here: `https://github.com/rdicosmo/docked-aspcud`

You can help

Building profiles

try out different preferences, find those that appear more useful, and propose them as profiles

Test expressivity of the preferences language

play with the preferences, check whether we need extensions (see `http://www.dicosmo.org/Articles/usercriteria.pdf` for existing proposals)

Help debug `opam`

if you are unhappy with a proposed install/upgrade, remember to dump the CUDF file (`--cudf` option)

Conclusions

Package managers are complex: a very hard part is dependency solving!
Modern package managers *must* share common components, in particular

dependency solvers *user preference language*

You ~~can~~ should use external solvers and preferences in `opam` *today*!

You might tell people *in other communities* they are welcome to adopt the same approach.

Learn more at `www.mancoosi.org`, `cudf-solvers.irill.org` and `ows.irill.org`

User preferences primer at `http://www.dicosmo.org/Articles/usercriteria.pdf`

Questions?

Old preference combinators

Notice that some solvers (mccs, packup) only support an older preference language (still recognised by `aspcud` 1.9 and later): here is the correspondence table

Old language	New language
removed	count(removed)
new	count(new)
changed	count(changed)
notuptodate	notuptodate(solution)
unsat_recommends	unsat_recommends(solution)