Using Preferences to Tame your Package Manager

Pietro Abate
IRILL & Univ. Paris Diderot
Pietro.Abate@pps.univ-paris-diderot.fr

Roberto Di Cosmo
IRILL & Univ. Paris Diderot
roberto@dicosmo.org

Louis Gesbert
OCamlPro
louis.gesbert@ocamlpro.com

Fabrice Le Fessant
INRIA & OCamlPro
fabrice.le_fessant@inria.fr

Stefano Zacchiroli
IRILL & Univ. Paris Diderot
zack@pps.univ-paris-diderot.fr

Abstract

Determining whether some components can be installed on a system is a complex problem: not only it is NP-complete in the worst case, but there can also be exponentially many solutions to it. Ordinary package managers use ad-hoc heuristics to solve this *installation problem* and choose a particular solution, making extremely difficult to change or sidestep these heuristics when the result is not the one we expect.

When software repositories become complex enough, one gets vastly superior results by delegating dependency handling to a specialised solver, and use optimisation functions (or *preferences*) to control the class of solutions that are found.

The `opam` package manager relies on the `CUDF` pivot format, which allows OCaml users that have a CUDF-compliant solver on their machine to reap the benefits of preferences-based dependency resolution. Thanks to the solver farm provided by Irill, these benefits are now extended to the OCaml community at large.

In this talk we will present the preferences language and explain how to use it.

1 Introduction

When installing a new software component or upgrading a previously installed one, users often make implicit assumptions regarding the outcome of such operations. To name one, we generally want to install the latest and greatest available version of any software component. Unfortunately, it is not always possible to do so. For example, installing a recent version of a package p may force to *downgrade* or remove another package q, which fall short of the general desire of only having the latest and greatest components.

What should a package manager do in the above situation? Remove q to get the latest p, or keep q and get the most recent available version of p which is compatible with q The answer really depends on fine-grained user expectations, and different users may have very different points of view and needs. Somebody might want to install the latest version of p because of a new feature that she needs, while someone else might want to just install any version of p and to keep a recent version of q.

Most package managers use ad-hoc heuristics to handle these cases, sometimes leading to surprising results. Moreover, since these heuristics are often hard coded in the dependency solving algorithms, it is usually quite difficult to adapt them to specific needs.

Following the guidelines of [2], this problem is handled in an original way in the design of the `opam` package manager, which is specifically tailored to install, upgrade and remove OCaml

libraries and tools. Leveraging 10 years of research in the realm of dependency solving, `opam` uses the de facto standard format CUDF[1] to encode upgrade problems, which is recognised by several specialised dependency solvers, and allows users to express complex scenarios in a well defined and compact way by means of *preferences* that are passed over to the external solver. To make external solvers easily available to all users, independently of their platform or processing power, we have made available at Irill a solver farm that can be used in the Cloud, thus extending the benefits of preferences based dependency solver to all the OCaml community.

2 Using preferences to guide component installation

A flexible and expressive means of tailoring the behaviour of a dependency solver is offered by the preference language developed by the Mancoosi project, which combines *package selectors* and *measurement functions* to give back to the user a fine control on how dependency solving is performed by the package manager. Here we describe the latest version of the preference language, stabilised during the MISC competition in 2012 [1].

Package selectors

Package selectors are used to identify specific sets of packages, and denoted by the constant expressions: `solution`, `new`, `removed`, `changed`, `up` and `down` .

Given an initial installation I and a proposed solution S, each of these constant expressions denotes a set of packages that can be made formally precise. For example, `solution` is the set of all packages installed in S, `removed` is the set of packages that were in I are no longer in S, and so on.

Measurements

A measurement is an integer valued function that can be applied to any of the above package selectors, plus possibly additional arguments. The measurements available in the preference language are the following:

$$count(X), sum(X, f), unsat\_recommends(X), notuptodate(X)$$

where X is a set of packages, and f a property of type `int` or one of its subtypes. These properties may be core properties defined in the CUDF specification, or properties that are declared in the preamble of the CUDF document.

Measurements are evaluated as follows. Note that some of the definitions make use of the set S (the solution) or U (the universe, that is the set of all known packages):

count(X) is $\#(X)$ that is, the number of packages in X.

sum(X,f) is $\Sigma_{x \in X} x.f$ that is, the sum over all packages in the set X of the value of their field f.

notuptodate(X) $\#\{p \in X | \exists q \in U : \ p.name = q.name \wedge p.version < q.version\}$ that is, notuptodate(X) is the number of packages in X that are not in the latest known version.

unsat_recommends(X) counts the number of disjunctions appearing in the `Recommends` (a.k.a. "soft/optional dependencies") field of packages in X that are not satisfied by S.

Different versions of this preference language are supported by several CUDF-compatible solvers, and can be directly exploited by `opam` users to tailor the behaviour of `opam` to their personalised needs.

[1] see http://www.mancoosi.org/cudf/ and the CUDF format specification [3]

Preferences A *user preference* is a sequence of measurements of package selectors, each prefixed by either the plus or minus sign, to indicate to the solver whether we desire to see the corresponding measure maximised or minimised in the solution.

For example, an `opam` user interested in installing a new package at its latest version, but touching as little as possible all other installed packages, may write the following preference expression:

-notuptodate(new),-count(changed)

Such expression can be passed to `opam` directly on the command line, using `--criteria`.

3 Dependency solving in the cloud

To be able to leverage the best dependency solvers, it is important to keep the solver as a modular component, separate from the rest of the package manager, as explained in [1, 2]. This may lead to some added complexity in setting up a multi-platform package manager like `opam`, as it is necessary to make sure that the existing dependency solvers (among which `aspcud`[2] is the most advanced) are available, with an aligned version, on all platforms.

For this reasons, a task force at Irill has been organised, with help from OCamlPro, and in particular by Pierre Chambart, Benjamin Canou and Gregoire Henry, to set up a solver farm which is online at http://cudf-solvers.irill.org.

The solver farm can be easily accessed over the network by `opam` to perform dependency solving *in the Cloud*. This can be particularly useful for developers working on devices with limited resources, like the Raspberry Pi. The testing phase has shown that the communication and solving time amount to less than two seconds, even on legacy internet connections. Hence the benefits of personalised dependency solving are now available to all the OCaml community.

4 Conclusion

In this talk we will present the basic underpinnings of external dependency solving for a package manager, with a special focus on the preferences language and its expressive power; we will also show how to use the CUDF solver farm in conjunction with the `opam` package manager.

Software availability The dependency solving toolchain covered by this talk is made of several building blocks, all of which are entirely written in OCaml and available as Free Software:

- `opam` — the new OCaml package manager, can be found at http://opam.ocamlpro.com/
- DOSE — the comprehensive library for dependency solving and analysis, collecting the works developed during the EDOS and MANCOOSI project, can be found at https://gforge.inria.fr/projects/dose/
- libCUDF — the reference implementation for manipulating and checking CUDF files, is available at http://www.mancoosi.org/cudf/

References

[1] P. Abate, R. Di Cosmo, R. Treinen, and S. Zacchiroli. Dependency solving: a separate concern in component evolution management. *Journal of Systems and Software*, 85(10):2228–2240, October 2012.

[2] P. Abate, R. Di Cosmo, R. Treinen, and S. Zacchiroli. A modular package manager architecture. *Information and Software Technology*, 55(2):459–474, February 2013.

[3] R. Treinen and S. Zacchiroli. Description of the cudf format. Technical report, November 2008. http://www.mancoosi.org/deliverables/d5.1.pdf.

[2] http://www.cs.uni-potsdam.de/wv/aspcud/