Introduction to 0install

Thomas Leonard

0install (pronounced "Zero Install") is a decentralised cross-platform package manager. "Decentralised" means that organisations and individuals can host their software in their own package repositories, without naming conflicts. "Cross-platform" means that it works on Linux, OS X, Unix and Windows. "Package management" means support for binary and source packages, dependency resolution, automatic updates and digital signature verification.

Originally created in 2003 to distribute the components of the ROX desktop[1], 0install was converted from Python to OCaml in 2013, improving robustness and performance. 0install is now available from the repositories of most Linux distributions (including Arch, Debian, Fedora, Gentoo, Mint, openSUSE and Ubuntu), as well as from Homebrew on OS X and through OPAM.

0install is used within the OCaml Platform to distribute signed binaries. This talk is both an introduction to 0install and an experience report from a new OCaml user.

1 Introduction

A single package format

Traditionally, every platform has its own unique method of distributing software. Examples include `.rpm` archives on Red Hat, `.deb` packages on Debian, `setup.exe` executables on Windows and `.dmg` images on OS X.

Creating, testing and maintaining packages in so many different formats requires considerable effort. For smaller projects, providing these packages may require more effort than writing the software in the first place.

Even when the packages exist, users will often struggle to install them unless they are part of the user's distribution's repository, as the software may depend on other libraries, which may or may not be part of the distribution.

0install does not define a new packaging format; unmodified `.tar` or `.zip` archives are used. Instead, it defines an XML metadata format to describe packages and the dependencies between them. A single metadata file can be used on multiple platforms (e.g. Linux, OS X and Windows), assuming binary or source archives are available that work on those systems. This greatly reduces the effort projects must spend producing packages.

However, as distributions provide a large number of high quality packages, 0install is also able to detect these and use them as additional candidates. 0install supports the package managers in Arch, Debian (including derivatives such as Ubuntu), Gentoo, MacPorts, FreeBSD (Ports), Red Hat (and other systems using RPM such as SUSE), and Slackware. It also contains hard-coded special cases for important packages on systems without package managers, such as being able to find native Java and Python installations on Windows and OS X.

A web of software

0install packages are identified by URIs rather than by short names. This makes naming conflicts impossible and removes the need for centralised repositories. Given the URI of a program, 0install will fetch the signed XML metadata from the appropriate repository (which is nothing more than a static web site). Any dependencies given in the metadata will themselves be URIs, which 0install will follow recursively to find the full set of candidates.

Isolation

Traditional package managers such as `rpm` and `dpkg` install a single version of each package, placing the files in shared directories such as `/usr/bin` and `/usr/lib`. This means that upgrading one program to a new version may force other libraries to be upgraded too, possibly breaking other software on the machine. Users must choose between a distribution containing only old (but well-tested) software, or one containing only new (less tested) programs.

0install selects package versions independently for each application. Each selected package is installed into its own directory, named after the hash of its contents. In the common case, each application will prefer the latest stable version of each library. The same version of each library will therefore be selected for each application and they will all share that single copy, as in a traditional system. However, if some applications require a different version for some reason, that version will be installed in parallel and used only by them.

This allows a user of an enterprise Linux distribution to

[1] `http://rox.sourceforge.net/`

access the latest versions of specific programs, or even to run multiple versions of the same program at once.

0install's cache, which contains one subdirectory for each package version, named after the secure hash of its contents, can be shared between multiple machines (physical or virtual), even if they have different architectures or operating systems. The machines will then automatically share any platform independent packages, since they will have the same hash, while installing architecture-dependent packages in parallel. No special support is required in the programs or libraries themselves to enable this. Since the only requirement for adding software to the cache is that the new directory's contents match its digest, untrusted VMs can download and submit new packages for inclusion, using a simple helper service.

Isolation also solves performance problems seen in some other systems. 0install's solver follows the OPIUM approach [2] of encoding all constraints as a SAT problem and then using a standard SAT solver algorithm [1]. However, where a traditional installer must consider simultaneously the needs of perhaps 2,000 installed packages, 0install's solver need only consider the 10 or so packages needed by the application being installed. In addition, 0install never needs to uninstall one package in order to install another, since all packages can coexist in the cache.

0install is designed to provide safe installation (but not execution) of untrusted software, and we would be interested to see people integrating it with various sandboxing technologies.

Language-specific package managers

Most programming languages need a way to distribute libraries. Relying on distribution package managers is not satisfactory for a number of reasons: Windows doesn't have a package manager; developers like their packages to be up-to-date; developers don't like installing build dependencies as root; and it should be possible to use a library without getting it added to every distribution's repository.

Examples of language-specific package managers include OCaml's `opam`, Haskell's `cabal`, Python's `pip`, Rust's `rustpkg`, Go's `go get` and Ruby's `gem`. While these work well for projects written entirely in that language, using a generic package manager such as 0install allows dependencies between components written in different languages. For new languages, 0install provides a ready-made solution that allows them to focus on language features rather than package management (however, it's important to note that providing a well-curated repository of library packages is still of vital importance, whatever technology is used).

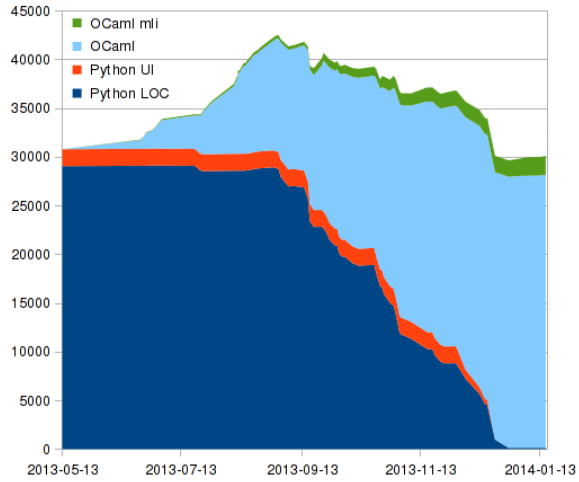

Figure 1: Lines of code during the conversion from Python to OCaml.

Developers

The XML file describing each program's requirements can also be included in a source-code repository, allowing full dependency handling for unreleased developer versions. For example, a user can clone a Git repository and build and test the program, automatically downloading newer versions of libraries where necessary, without interfering with the versions of those libraries installed by their distribution, which continue to be used for other software.

2 Porting to OCaml

In 2013 a number of factors combined to motivate switching 0install to a new language: an interest in using it on mobile platforms where Python's performance would be a problem, on-going stability and performance regressions caused by Python 3, and the desire for a statically-linked binary for bootstrapping.

An initial evaluation of seven languages (ATS, C#, Go, Haskell, OCaml, Python and Rust) eventually led to the decision to migrate the existing Python code-base to OCaml. Functionality was moved from the Python to the OCaml piece by piece, with the old and new code bases communicating using JSON, allowing the software to continue to work at all times during the transition. The code was migrated over the course of seven months, as shown in Figure 1, and the whole process was documented in a series of blog posts[2].

[2] `http://roscidus.com/blog/blog/categories/ocaml/`

With the port complete, we are now taking advantage of OCaml's features to improve the code with more module interfaces and abstract types, more immutable data structures, and fewer classes.

3 Conclusions

This talk will introduce and demonstrate the 0install tool, and report on the experience of converting it from Python to OCaml.

References

[1] Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In *Theory and applications of satisfiability testing*, pages 502–518. Springer, 2004.

[2] Chris Tucker, David Shuffelton, Ranjit Jhala, and Sorin Lerner. Opium: Optimal package install/uninstall manager. In *Software Engineering, 2007. ICSE 2007. 29th International Conference on*, pages 178–188. IEEE, 2007.