17-708 SOFTWARE PRODUCT LINES: CONCEPTS AND IMPLEMENTATION

DEBIAN PACKAGES – CASE STUDY

CHRISTIAN KAESTNER
CARNEGIE MELLON UNIVERSITY
INSTITUTE FOR SOFTWARE RESEARCH

READING ASSIGNMENT OCT 19

Apel, S., Batory, D., Kästner, C., & Saake, G. (2013). Feature-Oriented Software Product Lines. Berlin: Springer. -- Chapter 5

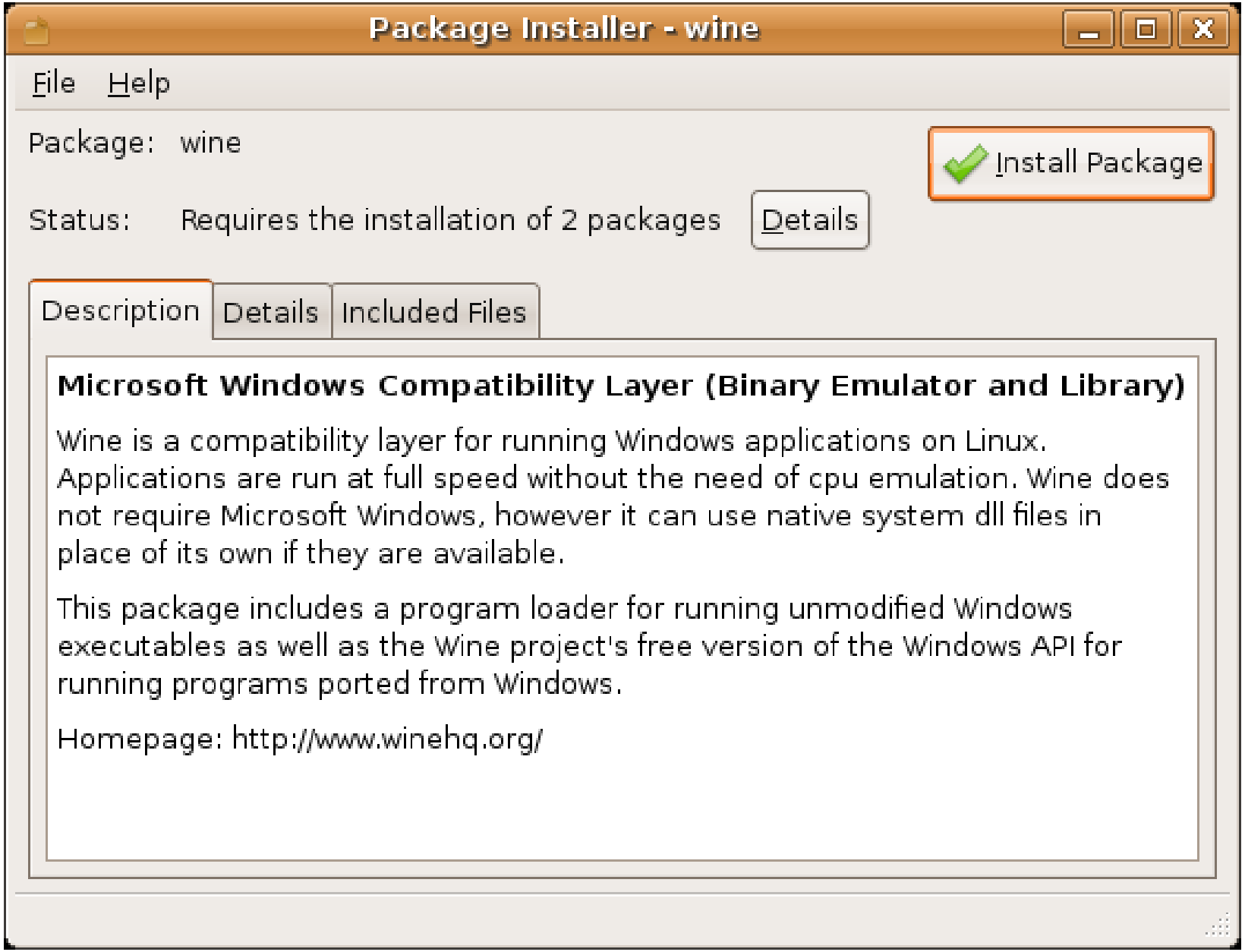
Favre, Jean-Marie. "Understanding-in-the-large." ***Program Comprehension, 1997. IWPC'97. Proceedings., Fifth Iternational Workshop on*****. IEEE, 1997.**

LEARNING GOALS

Understand how packages on Debian are bundled and composed

Identify the dependency management mechanisms and challenges when both revisions and variants are involved

Apply reasoning about configuration spaces outside of feature models


Package Installer - wine
File
Help
Package: wine
Install Package
Status: Requires the installation of 2 packages
Details
Description
Details
Included Files
Microsoft Windows Compatibility Layer (Binary Emulator and Library)
Wine is a compatibility layer for running Windows applications on Linux. Applications are run at full speed without the need of cpu emulation. Wine does not require Microsoft Windows, however it can use native system dll files in place of its own if they are available.
This package includes a program loader for running unmodified Windows executables as well as the Wine project's free version of the Windows API for running programs ported from Windows.
Homepage: http://www.winehq.org/

DEBIAN PACKAGES

Independent and interdependent programs to be installed on Unix systems

- **Component and context analogy**

Management through repository and package manager software

Dependency management

Open world, but global name space

Package:

- **Compressed archive with sources/binaries and meta-information**

Mancinelli, Fabio, et al. "Managing the complexity of large free and open source package-based software distributions." *Automated Software Engineering, 2006. ASE'06. 21st IEEE/ACM International Conference on*. IEEE, 2006.

CONTROL FILE

```
Package: firefox
Version: 1.5.dfsg+1.5.0.1-2 ...
Depends: fontconfig, psmisc,
 libatk1.0-0 (>= 1.9.0), libc6 (>= 2.3.5-1) ...
Suggests: xprint, firefox-gnome-support
 (= 1.5.dfsg+1.5.0.1-2), latex-xft-fonts
Conflicts: mozilla-firefox (<< 1.5.dfsg-1)
Replaces: mozilla-firefox
Provides: www-browser...
```

```
Depends: libc6 (>= 2.0),
 xlibs (>= 4.0) | xlib6g (>= 3.3.3.1), ...
```

INSTALLATION AND SCRIPTS

CONFFILE: files that are not overwritten during installation

preinst: execute before unpacking (e.g. stop service)

postinst: execute after unpacking (e.g., configure, often interactive, (re)start service)

prerm: execute before removal (e.g., stop service)

postrm: execute after removal (e.g., remove user files)

VIRTUAL PACKAGE

"A virtual package is a generic name that applies to any one of a group of packages, all of which provide similar basic functionality."

Mechanism to describe dependencies on alternatives

DEPENDENCIES

Package A *depends* on Package B if B absolutely must be installed in order to run A. In some cases, A depends not only on B, but on a version of B. In this case, the version dependency is usually a lower limit, in the sense that A depends on any version of B more recent than some specified version.

Package A *recommends* Package B, if the package maintainer judges that most users would not want A without also having the functionality provided by B.

Package A *suggests* Package B if B contains files that are related to (and usually enhance) the functionality of A.

Package A *conflicts* with Package B when A will not operate if B is installed on the system. Most often, conflicts are cases where A contains files which are an improvement over those in B. "Conflicts" are often combined with "replaces".

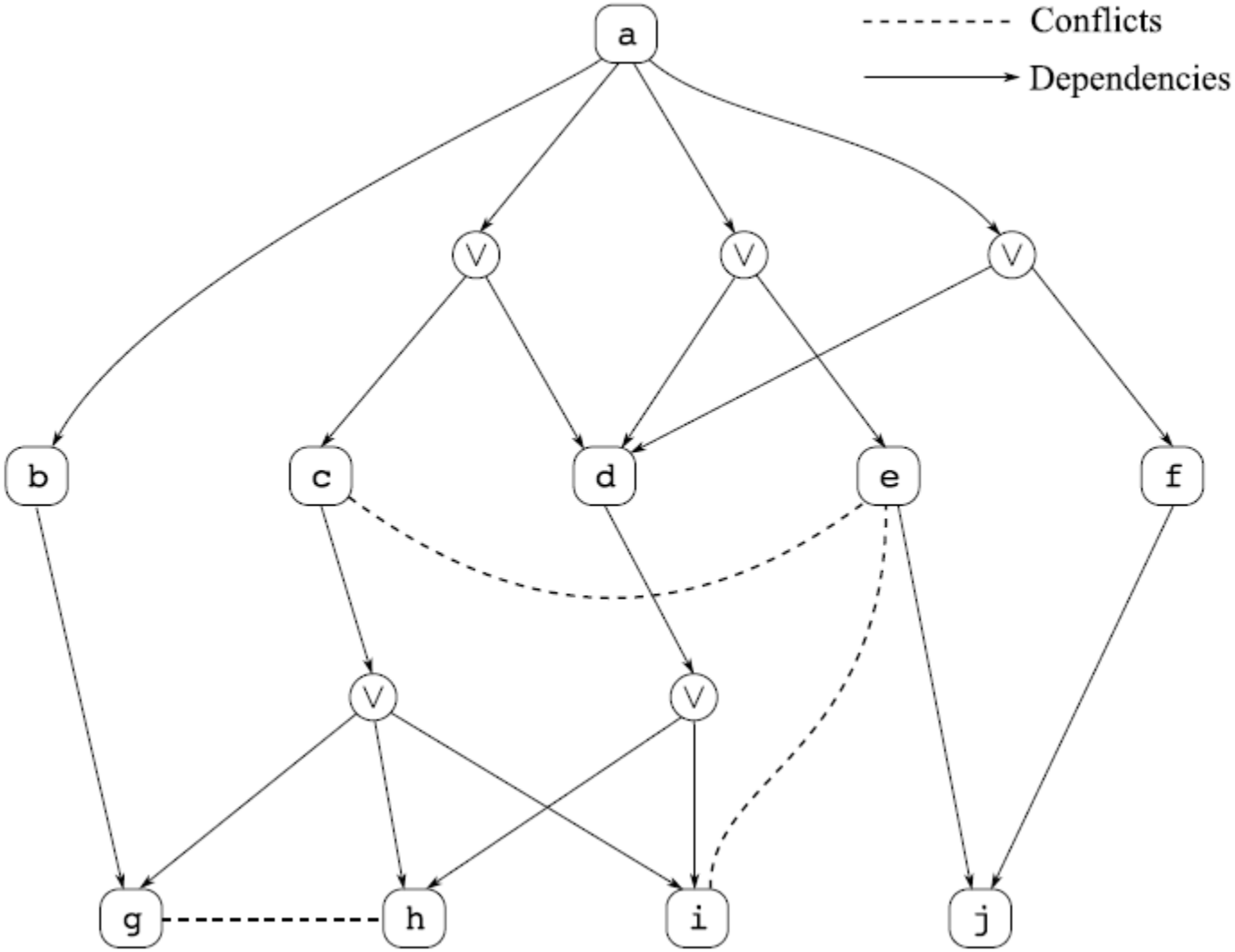
Package A *replaces* Package B when files installed by B are removed and (in some cases) over-written by files in A.

Package A *breaks* Package B when both cannot packages cannot be simultaneously configured in a system. The package management system will refuse to install one if the other one is already installed and configured in the system.

Package A *provides* Package B when all of the files and functionality of B are incorporated into A. This mechanism provides a way for users with constrained disk space to get only that part of package A which they really need.

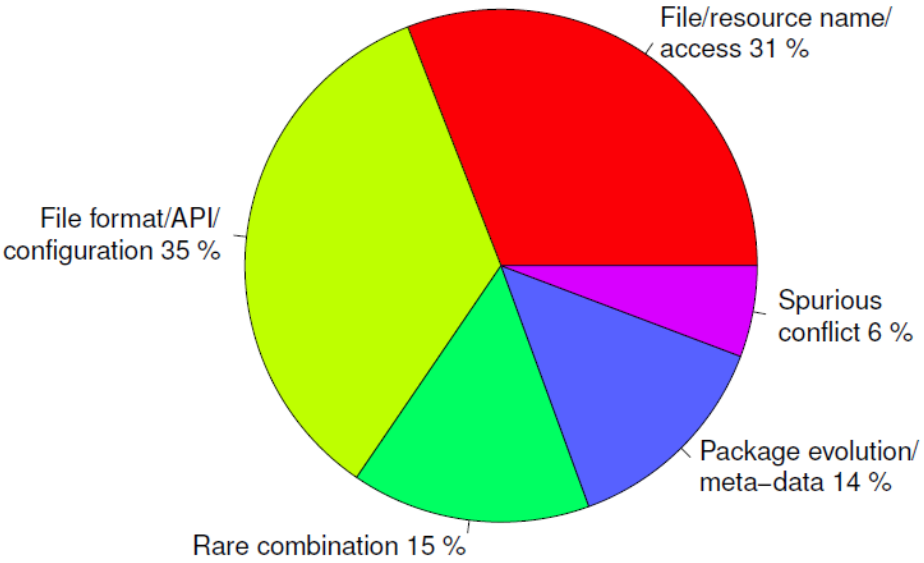
Mancinelli, Fabio, et al. "Managing the complexity of large free and open source package-based software distributions." *Automated Software Engineering, 2006. ASE'06. 21st IEEE/ACM International Conference on*. IEEE, 2006.

DEPENDENCY GRAPH

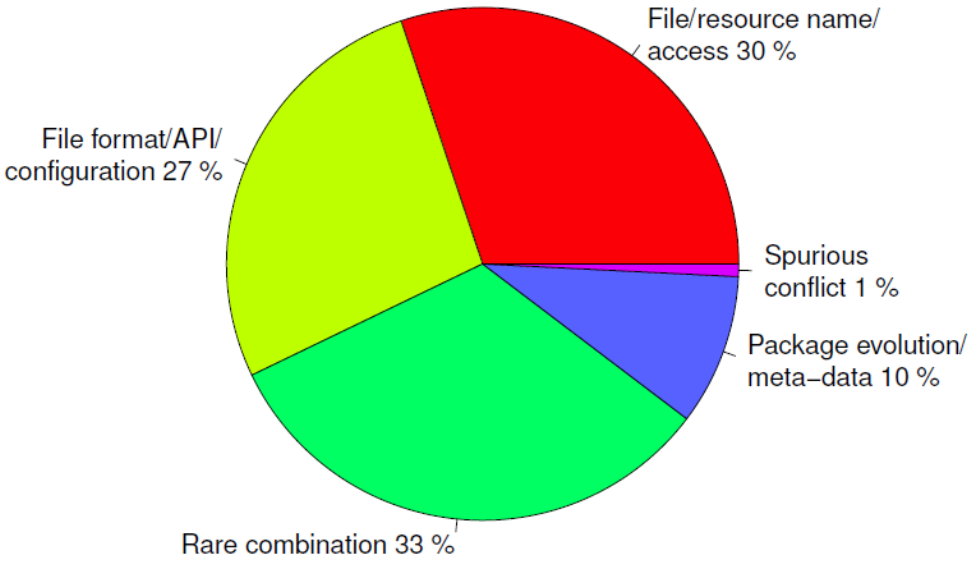


```
Unpacking gcc-avr (from .../gcc-avr_1%3a4.3.0-1_amd64.deb) ...
dpkg: error processing /var/cache/apt/archives/gcc-avr_1%3a4.3.0-1_amd64.deb
 (--unpack):
 trying to overwrite '/usr/lib64/libiberty.a', which is also in package binutils
dpkg-deb: subprocess paste killed by signal (Broken pipe)
Errors were encountered while processing:
 /var/cache/apt/archives/gcc-avr_1%3a4.3.0-1_amd64.deb
E: Sub-process /usr/bin/dpkg returned an error code (1)
```

Artho, C., Suzaki, K., Di Cosmo, R., Treinen, R., & Zacchiroli, S. (2012, June). Why do software packages conflict?. In Proceedings of the 9th IEEE Working Conference on Mining Software Repositories (pp. 141-150). IEEE Press.



(a) Debian



(b) Red Hat

Figure 8. Categorization of conflict defects in our case study.

Artho, C., Suzaki, K., Di Cosmo, R., Treinen, R., & Zacchiroli, S. (2012, June). Why do software packages conflict?. In Proceedings of the 9th IEEE Working Conference on Mining Software Repositories (pp. 141-150). IEEE Press.

ANALYSIS OPPORTUNITIES?

Component interfaces?

Dependency information?

Revisions?

Conflicting packages? Coinstallability?

Automatic reasoning? Conflict resolution?

Usability?