Building for different distributions with the openSUSE build service

Dr. Michael Schröder
Novell, Inc.

Novell

Outline

Fine points of project management in the build service.

- building for multiple distributions
- building on top of multiple projects

Building packages: how to control the build environment.

- automatic package expansion
- dealing with ambiguities and excess packages
- automatic package name rewriting

Working around installation script differences.

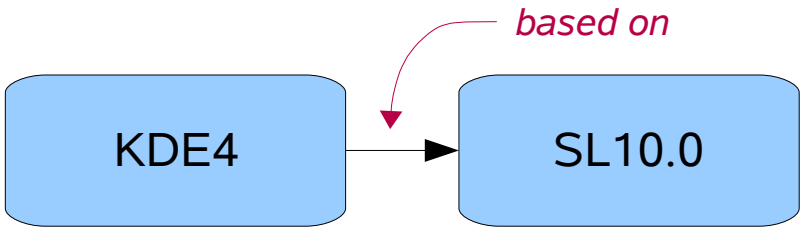
- build service offers standard set of macros

Distributions as Projects

Build service contains multiple complete distributions:

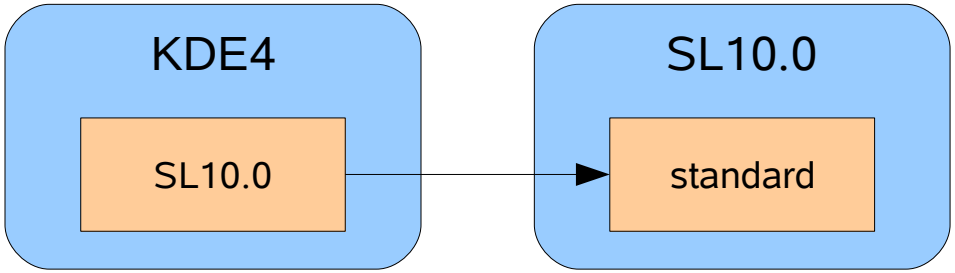
- SUSE Factory
- SUSE 10.0
- Mandriva 2006
- Fedora Core 4
- Debian Etch

These distributions can be used as “base” for other projects.



Project Repositories

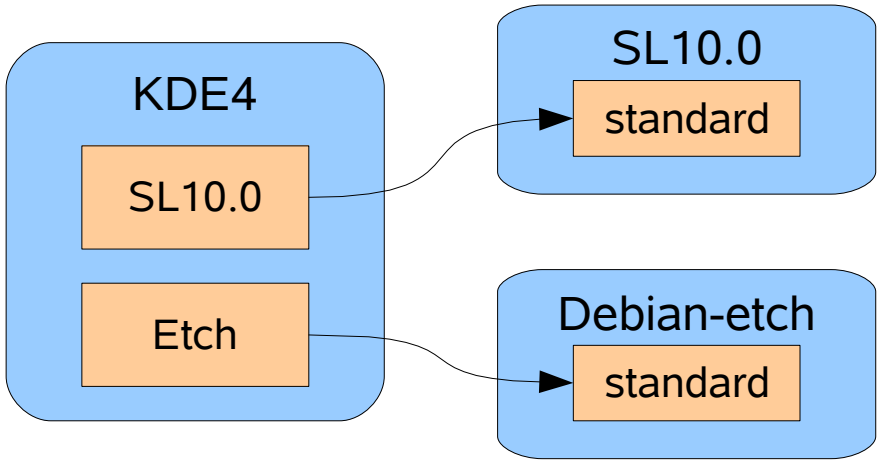
Binary packages are not stored directly with the project, but in a subpart, called “repository”.



Example: KDE4's repository “SL10.0” is based on SL10.0's repository “standard”.

Advantage of this scheme: a project can have multiple repositories.

Building with multiple repositories



Packages will be built for both SL10.0 and Debian-etch.

Each repository can also support multiple architectures.

Deb versus rpm packages

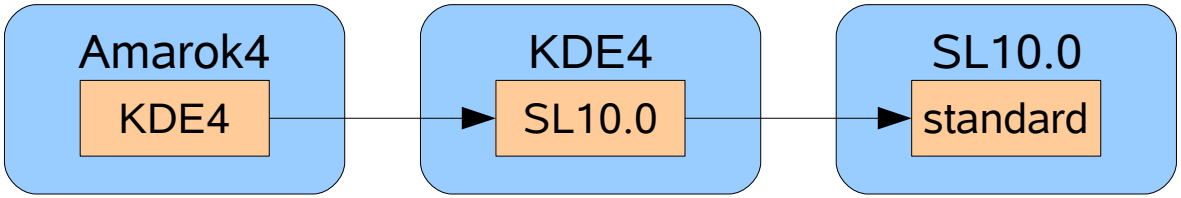
contents of spec file and debian build files are very different:

- rpm uses many macros
- deb uses many debian helper (`dh_xxx`) scripts
- different installation script semantics (order, arguments)

➜ Build service does not try to build a deb package from a spec file or vice versa.

- uses dsc file if base distribution uses dpkg
- uses spec file if base distribution uses rpm

Advanced repository linking

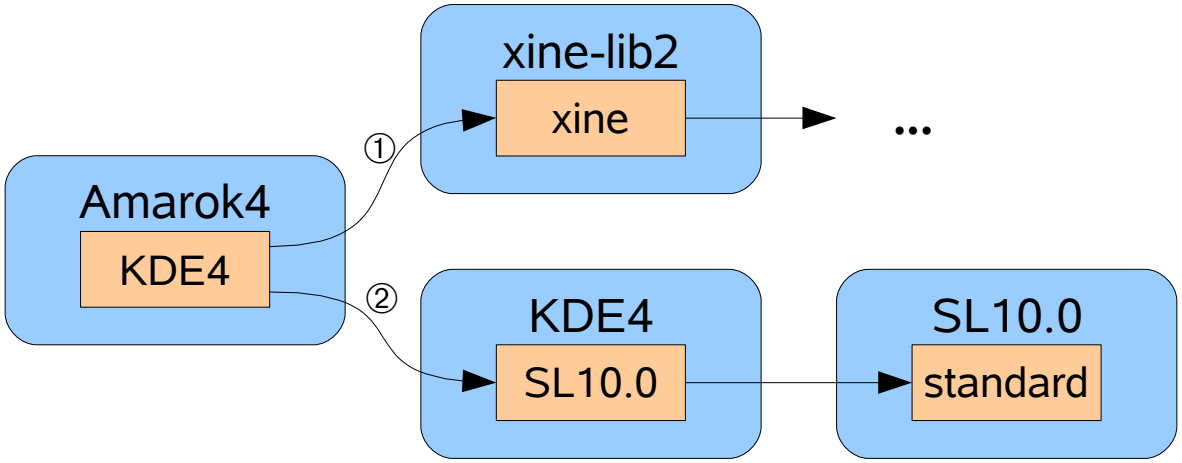


When building for the “Amarok4” project, packages will be first searched in the following order:

- the project's own repository
- the “SL10.0” repository of the KDE4 project
- the “standard” repository of the SL10.0 project

The system automatically adds the KDE4 → SL10.0 link to the search path.

Advanced repository linking (cont.)



Search order is:

- own repository
- xine-lib2's “xine” repository
- KDE4's “SL10.0” repository
- SL10.0's “standard” repository (automatically added)

The Repository Search Path

Summary:

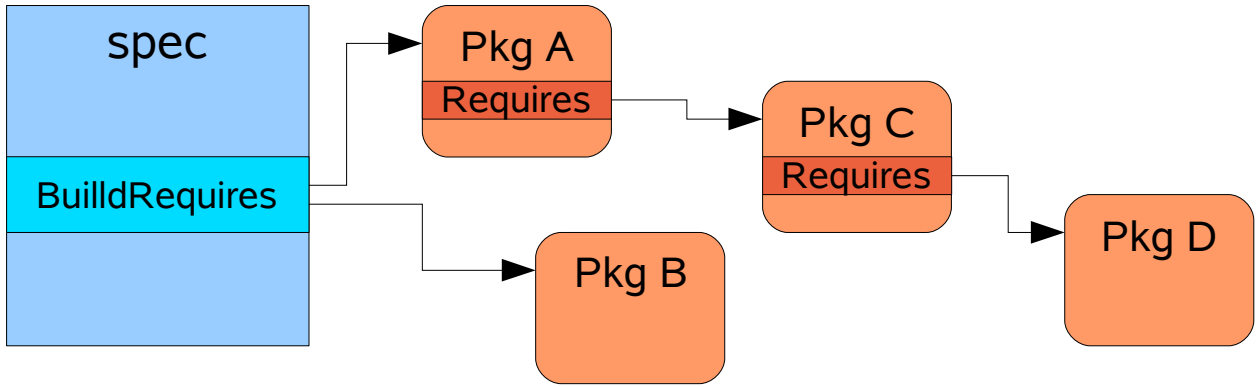
The Repository search path defines from which projects' repositories the packages are taken when setting up the build environment.

- order is important, first match wins
- the current repository is always added to the front of the path
- when the last repository entry of the path is reached, its search path is automatically added for further searching.

Setting up the build environment

The build service parses BuildRequires / Build-Depends from spec file / dsc file.

- these packages get added to a “base system”
- packages get automatically added so that all of the run-time dependencies are met



Problems caused by expansion

Expansion can pull in unneeded packages.

- Example: SL tetex requires xorg-x11-libs

Excess packages are bad:

- installation costs time
- your packages will only get built after the excess packages are finished
- your packages will get rebuilt if one of the excess packages is modified

➜ keep the number of involved packages as small as possible!

Breaking dependencies

To get rid of excess packages one can break the unwanted dependencies.

Dependencies can be broken on the project level (affects every package of the project) or on the package level:

- project level: by adding “Ignore:” lines to the project configuration

```
Ignore: tetex:xorg-x11-libs
```

- package level: by adding “#!BuildIgnore” lines to the specfile

```
#!BuildIgnore: xorg-x11-libs
```

Dealing with ambiguities

Ambiguities can happen if two packages provide the same functionality.

The system treats ambiguities as errors:

Specfile:

```
BuildRequires: apache2
```

```
expansion errors:
   have choice for apache2-MPM needed by
   apache2: apache2-prefork apache2-worker
```

Dealing with ambiguities

To solve ambiguities, either select one of the choices or deselect all unwanted ones:

- project level: “Prefer” lines

```
Prefer: apache2-prefork
Prefer: -apache2-worker
```

- package level: “BuildRequires” / “#!BuildIgnores”

```
BuildRequires: apache2-prefork
#!BuildIgnore: apache2-worker
```

Automatic dependency rewriting

Problem: packages get renamed or are named different for different distributions.

- Example: package containing shared libraries for canna

SUSE:	canna-libs
Fedora:	Canna-libs
Mandriva:	libcanna1
Debian:	libcanna1g

Project can specify per repository dependency rewrite rules:

`Substitute:` <*package*> <*replacement packages*>

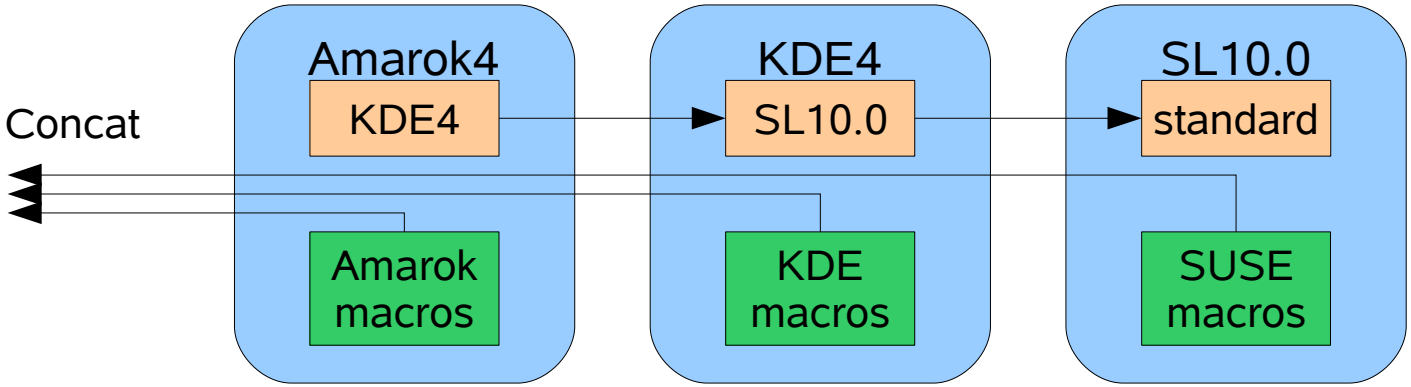
Project specific build data

A project consists of:

- a number of packages and repositories
- macros for the project
- information for setting up the build environment

The build process concatenates the configuration of every involved project.

- the repository search path defines which projects to use



Standard installation tasks

Three kinds of installation tasks make up major part of scriptlets:

- adding a service to the system
- adding a new user and new group
- adding an “info format” manual

Current distributions handle those tasks very different:

- SUSE: macros and code
- Fedora: no macros, lots of verbatim code
- Mandriva: macros and helper programs

➜ provide standard set of macros.

Adding a service

SUSE:

postinstall:

%{fillup_and_insserv -f <srv>}

preuninstall:

%stop_on_removal <srv>

postuninstall:

%restart_on_update <srv>

%insserv_cleanup

Mandriva:

postinstall:

%_post_service <srv>

preuninstall:

%_preun_service <srv>

Novell

Adding a service (cont.)

Fedora:

postinstall:

```
/sbin/chkconfig --add <srv>
```

preuninstall:

```
if [ “$1” = 0 ] ; then
    service <srv> stop >/dev/null 2>&1
    /sbin/chkconfig --del <srv>
fi
```

postuninstall:

```
if [ “$1” -ge 1 ] ; then
    service <srv> condrestart >/dev/null 2>&1
fi
```

Adding a service (cont.)

Proposed macros:

postinstall:

```
%service_add <srv>
```

preuninstall:

```
%service_del_preun <srv>
```

postuninstall:

```
%service_del_postun <srv>
```

Adding a new user/group

SUSE:

/usr/sbin/groupadd -g <gid> -o -r <group>
/usr/sbin/useradd -r -o -g <group> -u <uid> ... <user>

Fedora:

/usr/sbin/groupadd -g <gid> <group>
/usr/sbin/useradd -c <com> -u <uid> -g <group>...

Mandrake:

%_pre_useradd <user> <dir> <shell>

%_postun_userdel <user>

Adding a new user/group (cont.)

Proposed macro:

%user_group_add <name> <dir> <shell> <comment>

Only supports “Mandrake” semantics:

- add user and group in restricted area
- uid/gid is chosen by the useradd/groupadd program

Adding an “info” file

SUSE:

postinstall:

```
%install_info --info-dir=%{_infodir} \
              %{_infodir}/<file>.gz
```

postuninstall:

```
%install_info_delete --info-dir=%{_infodir} \
              %{_infodir}/<file>.gz
```

Mandrake:

postinstall:

```
%_install_info <file>
```

preuninstall/postuninstall:

```
%_remove_install_info <file>
```

Adding an “info” file (cont.)

Fedora:

postinstall:

```
/sbin/install-info %{_infodir}/<file>.gz %{_infodir}
```

preuninstall:

```
if [ “$1” = 0 ] ; then
    /sbin/install-info –delete %{_infodir}/<file>.gz \
        %{_infodir}
fi
```

Adding an “info” file (cont.)

Proposed macro:

postinstall:

%info_add <file> [infodir]

postuninstall:

%info_del <file> [infodir]

Mandriva compresses man-pages and info-pages with bzip2 instead of gzip, breaks file lists.

- Two new macros: %ext_man %ext_info

Conclusion

Multiple Repositories can be used to automatically build for different distributions.

- The built packages are stored in the different repositories
- Repository search path is used to merge the project configurations and thus the rpm macros.

The build service can build deb and rpm binary packages.

- spec files build rpms, dsc files build debs

Dependency expansion can lead to unwanted packages and ambiguities.

- both can be dealt with on project and package level

Standard installation tasks need standard macros.

Conclusion

Now start to populate the build service with lots of packages building for lots of different distributions!

General Disclaimer

This document is not to be construed as a promise by any participating company to develop, deliver, or market a product. Novell, Inc., makes no representations or warranties with respect to the contents of this document, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. Further, Novell, Inc., reserves the right to revise this document and to make changes to its content, at any time, without obligation to notify any person or entity of such revisions or changes. All Novell marks referenced in this presentation are trademarks or registered trademarks of Novell, Inc. in the United States and other countries. All third-party trademarks are the property of their respective owners.

Novell