Use of Configuration Management tool in LHCb software

J. Harvey, P. Mato, F. Ranjard
CERN (Switzerland)
Outline

- Configuration Management Requirements
- Physical Design
- How we use CMT
- Roles and Procedures
- Experience with CMT
- Conclusions
Configuration Management

- Code repository
  - Storage and identification of versions of the code
- Release mechanism
  - Make available a coherent set of libraries and programs
- Librarian
  - Essential role
- Tools to automate the work and check correctness
- Set of guidelines defining roles and procedures

- Activity spanning the complete software life cycle
- Basic entity is the *software package*
LHCb Software

- Many software developers (~500 physicists)
- Distributed around the world ( ~50 institutes, 15 countries)
- Many different software components to manage
  - Legacy Fortran code: 35 packages - still developed and in use to produce MonteCarlo data for detector studies.
  - New C++ framework (GAUDI) code : 10 packages.
  - New C++ reconstruction code: many sub-detector packages.
  - External libraries not maintained by us: CERNLIB, CLHEP, NAGC…

- Supported platforms
  - Linux, NT and HP
Requirements for Tools

- Have control on the configuration and dependencies of packages.
- Allow the release of a new version of a package without disturbing the main development line.
- Relieve developers from the burden of writing makefiles (or MSdev projects) taking account of various platforms and environments.
- Help librarians with installation of new packages or/and specific version of programs on different platforms.
- Allow customization of the configuration of a package.
- Allow to query the configuration used to build an application.
- Must be easy to use by casual users, developers and librarians.
Selected Tools

- CVS for managing the code repository
  - Linux and NT share the same code repository
- CMT for managing the software building and release process
  - Developed by C. Arnault (LAL/IN2P3)
  - Presented in this conference
Physical design

- **Physical design** (packaging) is an architectural issue.
- Big consequences on:
  - compilation time
  - link dependencies
  - configuration management
  - executable size
  - ...
- Package interdependencies require approval of architect.
- Avoid cyclic dependencies
Package layout

- All packages are available in the public release area
- Users may checkout from repository in their private working area
- Layout is identical in both areas.

Version number

$PACKAROOT

manager directory contains the requirements file

public include files
#include "packA/xxx.h"

binaries

mgr  src  packA  doc  win32  hp-ux102  ...  i386-linux22
How we use CMT

- What to build
- How to build
- Package dependencies

CVS repository

requirements

CMT

makefile
DevStudio files

Building tools (compilers, linkers, IDEs)

Libraries & Executables

code
package packA
version v1

branches doc src mgr packA
include_dirs $(PACKAROOT)

use packB v1
use packR v2r1
use CERNLIB v2000
use CLHEP v1r4

library packA ../src/*.cpp

macro packA_linkopts "$(PACKAROOT)/$(packA_tag)/libpackA.a"
    VisualC "$(PACKAROOT)/Win32/packA.lib"
macro packA_stamps "$(PACKAROOT)/$(packA_tag)/packA.stamp"
Package categories

- **A program**: is a package that contains a main routine and a list of dependent packages needed to link it.
  - The requirements file contains the name and version of all the packages used by the application.

- **A library**: contains a list of routines (classes) and the list of dependent packages needed to compile it.
  - The requirement file indicates how the result library is linked in programs.

- **A package group**: contains a list of other packages with their version number valid for a specific version of the framework.
  - To install the current version of the framework in a new environment it is sufficient to install the framework package and all dependent packages.

- **An external package**: CERNLIB, CLHEP, ROOT,... are maintained by external groups.
Package categories(2)

- Their requirements file contains references to their include files and binary locations.
- The use of the CMTSITE environment variable allows the various locations to be defined in a single place

```plaintext
package CLHEP
version v1r4

set LHCXX_DIR ""
    CERN  "/afs/cern.ch/sw/lhcxx/specific/@sys/egcs_1.1.1"
    NIKHEF  "/project/lhcxx/specific/lnx"

include_dirs "${LHCXX_DIR}/1.4/include"
macro CLHEP_linkopts "${LHCXX_DIR}/11.4/lib/libclhep.a"
```
Roles and Procedures

- **The casual user:** develops an algorithm in his working area and builds an application by linking it with other selected packages from the public release area.
  
  ```
  > cd mywork
  > cmt checkout LHCbprog
  > cd LHCbprog/v1/src
  add user code
  > cd ../mgr
  modify the requirements file if necessary
  > gmake
  the executable is stored in ../$(LHCbprog_tag)
  ```
Roles and procedures (2)

- The package developer: develops and maintains code for general public use. He is expected to supply code, test routines and documentation.

  set the CMTPATH to use the private version of the package:
  
  > cd mywork
  > setenv CMTPATH $PWD

  check out the package he is working on:
  
  > cmt checkout packA
  > cd packA/v2/src

  modify code
  
  > gmake

  check out the program package:
  
  > cmt checkout LHCbprog v1
  > cd LHCbprog/v1/mgr
  > gmake
Roles and Procedures (3)

- **The librarian**: installs new versions of packages, programs, package groups in the public release area using the CMT recursive mode to **checkout** all packages dependent on the package being checked out and the **broadcast** facility to build the corresponding libraries:

  ```
  remove the CMTPATH definition to not use private code:
  > unsetenv CMTPATH
  go to the release area and checkout the program package
  > cd $LHCBSOFT
  > cmt checkout -R LHCbprog
  > cmt broadcast cmt config
  > cd LHCbprog/v2/mgr
  > cmt broadcast gmake
  ```
Our experience with CMT

- Easy to use (physicist, developer, librarian)
- Very helpful (no need to write makefiles or MSdev projects)
- Full monitor of versions and options used during build.
- Build options are inherited from the package hierarchy (single place).
- Partial release.
- Runs on UNIX and NT (essential for us).

- Scalability (foreseen problems when large number of packages)
- The “use” statement needs to be qualified (for compilation, linking, running)
Conclusions

- We have taken Configuration Management very seriously from the beginning.
- It is essential for managing contributions from many developers working on various platforms.
- The Fortran simulation program is in production in 5 institutes in very different environments.
- Our Configuration Management relies on the CVS & CMT tools.
- We have been using CMT for about a year and we are very happy with it.