From source code management to binaries production for Arpege/Aladin

- Overview of the situation / solutions
- Code organization at GMAP
- Tools to operate on the code at GMAP

R. El Khatib Météo-France – CNRM/GMAP October, 2002

Why to use a source code management software ?

- Huge software (≈ 400 000 lines)
  => Organization
- Numerous developers (≈ 75 to 100 persons)
  => Phasing
- Perpetual changes (innovations, debug, …)
  => Historization
- Multi-purposes (operations, research …)
  => Versioning

Aspects of source code management at Météo-France

- Continung principle :
  - Code partitioning in a database
  - Stamped modifications
  - More or less automatic merging operations
- Continuing problem :
  - The fear of developers !
  - Necessity to learn how to use the tool

An investment for the code maintenance

Arising problems with source code handling

- Externalizations, modularizations :
  - ☺ : More flexibility and longer life
  - ☹ : Manual links between object libraries
- Programming languages :
  - ☺ : more structured and more robust
  - ☹ : Dependencies of modules => hierarchical compilation

Paradoxally …
It is more and more difficult to make binaries !!!

From compilation to binaries

- Before the use of Fortran 90 modules :
  - …There were nothing ! (too easy ?)
- With F90 and the dependencies problems :
  - Elaboration of a script based on a Makefile
  - Quite a failure but we have drawn the lessons !
- After the Makefile failure :
  - Home-made user-friendly sophisticated unified procedures : now stable ☺

Arpege/Aladin code : organization at GMAP

- Original source codes inside a database (ClearCase) : for viewing or modifying
- Copy of the code for a html browser
- ***GCO*** : Copy of source codes, object codes, libraries and binaries on the supercomputer for compilations/executions : « packs »
- ***Developers*** : virtual « packs » (mainly : links)

Arpege/Aladin « packs » (1)

- Arborescent structures containing :
  - bin/ : binaries
  - lib/ libraries
  - src/ : source and object code
- Below src/ :
  - A « ClearCase-view-like » structure of branches (ex : main/ bf/)
  - Below each branch : all the projects (ex : arp/ ald/ tal/ tfl/ xrd/ odb/ etc …)
  - Below each project : a « ClearCase-project-like » structure of code (ex : setup/ control/ etc …)

Arpege/Aladin « packs » (2)

- Strict naming conventions for the background packs

  Example :

  cy24t1_export-bf.01.L9912.x.pack/

  Leading arpege release Compilation options

  Branch release Compilation

- Strict naming conventions for the files in packs
- Strict definitions for object libraries :
  - Ordering (ex : *ald* before *arp*)
  - Content (ex : bf_[n] always included in bf_[n+1]

Current operations on source code

- Local code modification
  - ✓Use ClearCase to make a « branch »
- Local pack elaboration
  - ✓Procedure « gmkpack » to create a user pack
  - ✓Procedure « cc_pack » to populate the pack from the Clearcase branch
- Local pack usage
  - ✓Procedure « gmkpack » to create a script for compilation and binary elaboration

gmkpack

- Purpose :
  - to compile, make object libraries and binaries in a « pack » environment
- Specifications :
  - **Universal** (serving anybody working on arpege/aladin)
  - **Secure** (specially : in libraries/compilation ordering)
  - **Flexible** (choice of compiling options, libraries …)
  - **Users friendly** (simple interface, automatic analysis of what should be recompiled/rebuilt)
  - **Efficient** (fast commands, fully distributed compilation)

… And what about portability ??

Conclusion/perspectives

- Source code manager : seems ok (up to now)
- Source code browser : to be fully re-written soon
- Packs structure : needs further rationalization
- gmkpack :
  - Its portability would warranty its long life
  - Its transversality (GMAP/GCO) would warranty the robustness of the whole code management