OOPS Meeting - 18 November 2009

9:30	-	**Re-structuring the IFS** Yannick Trémolet
10:10	-	Coffee Break
10:40	-	**Do we need full object-oriented features?** Mike Fisher
11:25	-	**Practical lessons learned from implementing a toy system** Deborah Salmond, Anne Fouilloux
12:00	-	**From IFS to OOPS** Yannick Trémolet
12:15	-	Lunch

OOPS Meeting - 18 November 2009

Afternoon Discussions

Council Room

13:30	-	Discussions
15:00	-	Break
15:30	-	Discussions
17:00	-	End

What language should we use?

- In principle, Fortran 2003, C++ or C++/F90 can be used.
- Re-writing the entire code in C++ would require too much ressources?
- Are Fortran 2003 compilers available on all platforms (and reasonably stable)?
- Is the combination of C++ and Fortran 90 the right approach?

User interface

- There are many cases where the IFS tries to be clever and resets input parameters to *sensible* values, for example depending on the resolution. prepIFS does the same, unless run in expert mode.
- It is up to the user to decide what parameters are sensible. If a user sets a nonsensical parameter then prepIFS should issue a warning, and the IFS should abort if it cannot run with this parameter.

- Namelists provide a crude, one-dimensional method for configuring the model.
- In 4D-Var, we run multiple instances of the model within the same executable. Things will get even more complicated if we decide to incorporate the outer loop into the executable. With namelists, it will be difficult to pass different configuration parameters to different instances of the model (inner/outer loop model, different resolution minimisations, etc.)

- An XML (or similar: YAML, JSON, etc.) configuration file would allow a more structured approach, and would allow us to take advantage of the wide availability of XML viewers and editors.

Scripts

- 4D-Var spends 25% of its time executing script code.
- The scripts are long, complicated and difficult to maintain.
- Korn shell is not very suitable for such complicated tasks:
  - Global namespace.
  - Limited control structures.
  - No notion of `subroutines`.
  - No data structures.
- Should we use a modern scripting language, such a Python?
- Should we share scripts (RD/OD/M-F/Aladin/Hirlam)?

Source code management

- At ECMWF, version control (Perforce, ClearCase, etc.) is used in a very sub-optimal way resulting in a massive code database, containing hundreds of versions of many files, and many debug branches.
- The size of the database brought ClearCase to its knees, and is stretching the capabilities of Perforce.
- Having such a massive code database makes it difficult to trace changes. Many of the tools provided by the version control system are rendered useless.
- We should develop a better way of working with the version control system.
- This has implications for the compilation system.

Understanding the code

- Documentation:
  - The documentation for IFS CY33R1 (April 2008) came out this week.
  - Should we integrate the documentation with the code more?
  - Do we need more formal code specifications? UML?
- How much training is needed?
  - OOP in general or language specific?
  - For core team?
  - For general IFS users/developpers?

What should be done inside the model?

- Modularising will be make coding and debugging easier.
- More can be done in addition to the work being proposed in data assimilation.
- Interface dynamics/physics?
- Limited area?

Next steps

- Organisation, decisions, timeframe?
- Propose guidelines for "bottom-up" work (approved before Easter).
- First version of toy system available before Christmas.
  - Fortran 2003 and C++/F90
- Second version of toy system available before Easter.
  - Fortran 2003 or C++/F90
- Next steps?
- Ressources?