OOPS: Object-Oriented Prediction System
From IFS to OOPS

Yannick Trémolet

ECMWF

November 18, 2009

Outline

Toy System Summary

- The basic design seems appropriate for our purpose.
- Data assimilation algorithm can be made independent from the model.
- The same basic design can be implemented in Fortran 2003, C++ or a mixture of C++ and Fortran 90.
- Tools (debugger, tracebacks, profilers, MPI, etc...) work for all languages.
- Performance should not be an issue since we only re-code the control level where almost no computing time is spent.

Programming Languages

- C++:
  - Syntax takes getting used to (for Fortran programmers),
  - More flexible (OO aspects and memory management),
  - Compilers are available, widely used and debugged.

- Fortran 2003:
  - The OO aspects are more limited but enough for a scientific code.
  - `SELECT TYPE` construct is a cumbersome equivalent for (dynamic) cast.
  - User defined constructors are missing!
  - Some compilers are available but we are debugging them.
  - Fortran 2008 looks promising but when will we have it? (Co-Arrays exist at least since 1996.)

- Python:
  - Should be used for scripts.

Other questions to be investigated in the toy system

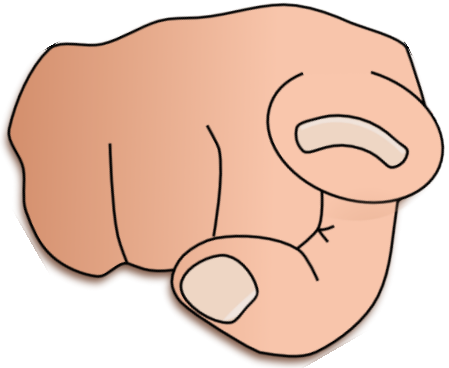
- Interfacing with ODB and use of classes for major observation types.
- Should the outer loop be done at script level or in the code?
- Handling of trajectory.
- Write cycling scripts. Can they be made abstract?
- Testing and validation:
  - Inside the code: adjoint tests for example,
  - Outside the code: test suite for all major configurations.
- Configuration of the system:
  - Should choices be made in prepOOPS, the scripts or the code?
  - Use of XML (or other modern format) instead of `namelists`.

Outline

Transition from IFS to OOPS

- The main idea is to keep the computational parts of the existing code and reuse them in a re-designed structure.
- This can be acheived by a top-down and bottom-up approach.
- From the top: Develop a new modern, flexible structure.
  - Expand the existing toy system.
- From the bottom: Move setup, namelists, data and code together.
  - Propose new coding guidelines to that effect,
  - Everybody participates by applying it to the part of the code they know.
  - Create self-contained units of code.
- Put the two together: Extract self-contained parts of the IFS and plug them into OOPS.
  - This step should be quick enough for versions not to diverge.

How will it affect you?

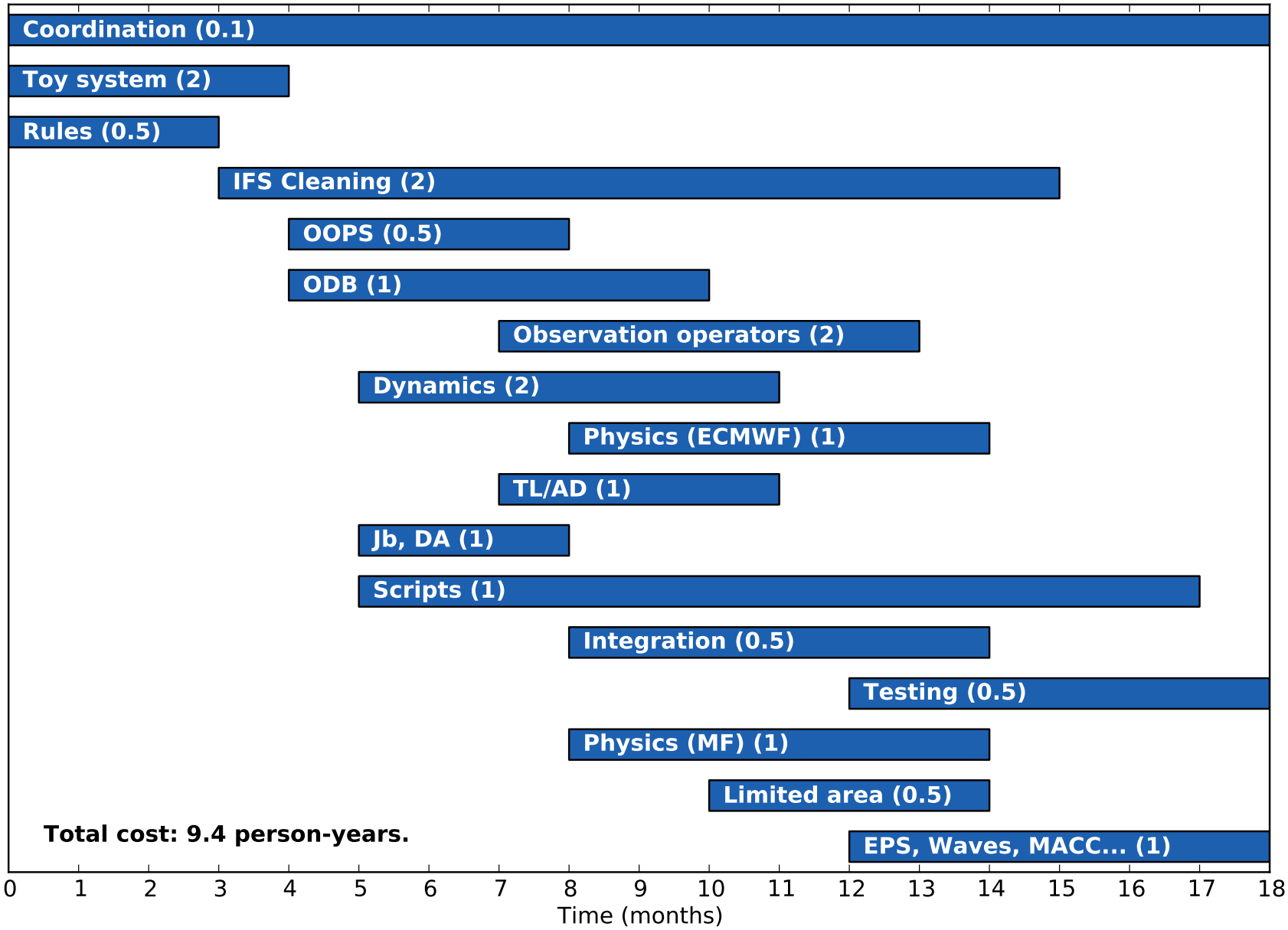


- In the model: very little below `STEPO`/`STEPOTL`/`STEPOAD`.
  - The required functionality is to move the state forward in time.
  - The fundamental unit is to move forwad by one time step.
  - `STEPO` is the natural interface.
- In the observation operators: very little below `COBS*`/`HOP*`.
  - `COBS*` and `HOP*` might be merged into one,
  - The routines they call should not change much.
- This assumes the initial cleaning phase has taken place!
- The boundary between object and non-object code could be moved to lower levels as necessary/desirable.

Around the IFS/OOPS

- Coding guidelines (avoiding overly restrictive norms),
- Better use of a version control system,
  - Compiling environment.
- Configuration files to replace `namelists`,
  - prepOOPS,
  - Scripts.
- Testing suite.

From IFS to OOPS



Some additional benefits

- Changes in the model should not affect data assimilation... as long as it keeps the ability to move the state forward in time!
- Toy systems are very useful to understand concepts and validate ideas.
- It is possible to move to the full system without re-writing code (for algorithms that can be written at the abstract level).
  - No fear that testing ideas in a simple system first will slow down the *real work*.
  - The cost of developing the system is compensated by increased productivity.

Summary

- The quality of a scientific code is not measured only by CPU time.
- Reliability, maintenability and flexibility are important.
- A small overhead is acceptable if it does not break scalability.
  - The approach envisaged here only affects the highest levels of the code structure which accounts for a negligible fraction of the run time.
  - Each entity can be optimised fully and independently.
- Applies to any model: Lorenz, Lorenz 95, QG, surface, NEMO, IFS.

Summary

- The quality of a scientific code is not measured only by CPU time.
- Reliability, maintenability and flexibility are important.
- A small overhead is acceptable if it does not break scalability.
  - The approach envisaged here only affects the highest levels of the code structure which accounts for a negligible fraction of the run time.
  - Each entity can be optimised fully and independently.
- Applies to any model: Lorenz, Lorenz 95, QG, surface, NEMO, IFS.

- We should take advantage of advances in programming technology.
- Improve speed of development and productivity of everyone involved in developing the IFS.
- A better, more flexible, more user friendly IFS is possible!
- The code should not limit the science!