Mini-OOPS information half-day during Anne Fouilloux's visit at MF (May 11, 2011)
about the very first version of OOPS/IFS type of code;
toward the 3D-VAR demonstration prototype

participants (MF) : Florence Rabier, Claude Fischer, Thibaut Montmerle, Patrick Moll, Pascal Lamboley, Eric Sevault
participants (EC) : Anne Fouilloux
Hirlam visitor: Sami Saarinen

Anne recalled the organization of the IFS-cleaning and the OOPS work at ECMWF: split between Fortran and C++ activities and 4 major topics at Fortran level: definition of fieldset/State, interpolation and geometry/grid, obs interpolators, Jb

The principle of mini-OOPS is to adjust an OO/C++ layer to the minimum required to plug IFS code into it, via C_BINDING for pointer interoperability and specific Fortran interfaces (= interfaces of the methods in OO/C++ vocabulary). C++ should be as much as possible decoupled from Fortran, so that a large number of scientists would not need to care about C++ and OO, and rather focus on the Fortran computational developments and related science. Fortran modules should become more and more autonomous.

Only data assimilation and algorithms people would need to learn about OO/C++. Claude stresses that EPS systems should be considered at the same level as assimilation.

Would specific analysis techniques and vocabulary be helpful to proceed with OOPS ? (UML ...)
Probably no, because these techniques really are helpful only when you start a project from scratch, while here we already have the IFS. However, documentation about goals, expected objects/classes (what they are and what they should be able to do) and their inter-dependence (collaboration diagrams) would be helpful. Those would for instance help people from outside the NWP community to more easily understand the aims of OOPS, while offering NWP scientists some easy track to enter OO thinking.

Anne presented the code of the mini-OOPS on PC:
* thin and rather simple C++ layer (compared with the implementation in the toy model)
* care was taken to well define the several interfaces and methods toward Fortran: obsvalue, obsequiv, obsdep, ...
* for the time being , the mini-OOPS calls HOP (based on CY37R1) and reads AMSUA data from an ODB/ECMA database. The model value is taken from a specific input file (not a full IFS file).
* the code computes y-H(x), so it's equivalent to a Fortran IFS sequence calling obs setup, then HOP and HDEPART (sort of TASKOB)

ECMWF will make a new version of the mini-OOPS available, based on CY37R3. This version would then be compilable and linkable with the latest version of the IFS interim cycle R3.
The 3D-VAR prototype is scheduled for the autumn 2011.

A discussion about various specific aspects around OO and its implementation in mini-OOPS took place. Concerns about the treatment of MPI for several State-objects of different geometries were addressed: it should be possible to instantiate several different geometries/grids in a same executable run. Data structures would have a certain level of hierarchy (in the way of Russian dolls)

so that the highest level eventually would contain (pointers to) about all the data-related aspects of the IFS. At any level below, ideally, one only should use pointers to the required level of the structure, and at some level in Fortran, one should stop using pointers to structures in modules, and only pass pointers as dummy arguments.
Indeed, a change in the modules containing the data structure probably will force to re-compile all the hierarchy of code making USE of these embedded types.

MF staff recalled the importance to assess the benefit of the OOPS work for alternative algorithms, like ETKF or EPS. Be able for instance to show that one can run in a same executable several different resolutions of a forecast model, or a global / coupling / LAM coupled system.

The present mini-OOPS OO layer does not offer new possibilities for plugging in new obs operators. This latter facility would have forced a much bigger recoding in Fortran, and has been postponed. So there will be no strong benefit from OOPS in terms of implementing new obs operators in the short term. But it will allow to start working on various new assimilation algorithms using real IFS operators and code, like for long window 4D-VAR.

Note: Sami Saarinen was taking part as Hirlam participant. He will report back to Hirlam .

Appendix: list of methods that need to be implemented as Fortran routines in the mini-OOPS (the code corresponds to the C++ wrapper)

```
#ifndef IFS_FORTRAN_H
#define IFS_FORTRAN_H

// Forward declarations

namespace utils {
  class DateTime;
  class XmlDom;
}

using utils::XmlDom;
using utils::DateTime;

namespace IFS {

//! Empty classes to give types to fortran pointers

class f90geom {};
class f90conf {};
class f90vars {};
class f90locs {};
class f90flds {};
class f90traj {};
class f90bmat {};
class f90goms {};
class f90ovec {};
class f90help {};
class f90obso {};
class f90rmat {};

//! Interface to Fortran IFS model
```

```
extern "C" {

// ------------------------------------------------------------------------
//  Geometry
// ------------------------------------------------------------------------
//    Tomas, Deborah, Mike, Yannick, George, John, Mats
// ------------------------------------------------------------------------

  void geometry_setup(f90geom **, const XmlDom **);
  void geometry_clone(const f90geom * const *, f90geom **);
  void geometry_delete(f90geom **);

// ------------------------------------------------------------------------
//  Model
// ------------------------------------------------------------------------
//    Tomas (not needed for 3DVar)
// ------------------------------------------------------------------------

* void ifs_setup(const XmlDom **, const f90geom **,
                 f90conf **, f90vars **, f90vars **);
* void ifs_delete(f90conf **);

* void ifs_init   (const f90conf * const *, f90flds **);
* void ifs_inittra(const f90conf * const *, f90flds **, f90traj **);
* void ifs_init_tl(const f90conf * const *, f90flds **, const f90traj **);
* void ifs_init_ad(const f90conf * const *, f90flds **, const f90traj **);

* void ifs_step   (const f90conf * const *, f90flds **);
* void ifs_steptra(const f90conf * const *, f90flds **, f90traj **);
* void ifs_step_tl(const f90conf * const *, f90flds **, const f90traj **);
* void ifs_step_ad(const f90conf * const *, f90flds **, const f90traj **);

* void wipe_traj(f90traj **);

// ------------------------------------------------------------------------
//  Fields
// ------------------------------------------------------------------------
//    Tomas, John, George, Deborah, Yannick, Mike
// ------------------------------------------------------------------------

  void field_setup(f90flds **, const f90geom **, const f90vars **);
  void field_clone(const f90flds **, f90flds **);
  void field_delete(f90flds **);

  void field_assign(f90flds **, const f90flds **);
  void field_assign_scal(f90flds **, const double &);
  void field_add(f90flds **, const f90flds **);
  void field_sub(f90flds **, const f90flds **);
  void field_mul(f90flds **, const double &);
  void field_axpy(f90flds **, const double &, const f90flds **);
  void field_prod(const f90flds **, const f90flds **, double &);

  void field_add_incr(f90flds **, const f90flds **);
  void field_diff_incr(f90flds **, const f90flds **, const f90flds **);

* void field_change_resol(f90flds **, const f90flds **);

  void field_read(f90flds **, const XmlDom **, DateTime **);
  void field_write(const f90flds **, const XmlDom **, const DateTime **);
```

```
// IFS might need a non tl version of interp?
  void field_interp_tl(const f90flds **, const f90locs **, f90goms **);
  void field_interp_ad(f90flds **, const f90locs **, const f90goms **);

  void field_gpnorm(const f90flds **, const int &, const char *);
  void field_random(f90flds **);

// ------------------------------------------------------------------------
//  Background error
// ------------------------------------------------------------------------
//    Mike
// ------------------------------------------------------------------------

  void b_setup(f90bmat **, const XmlDom **, const f90geom **);
  void b_delete(f90bmat **);

  void b_linearize(f90bmat **, const XmlDom **);

  void b_mult(const f90bmat * const *, const f90flds **, f90flds **);
  void b_inv_mult(const f90bmat * const *, const f90flds **, f90flds **);

* void b_sqrt_mult(const f90bmat * const *, f90flds **, const double *);
* void b_sqrt_mult_ad(const f90bmat * const *, const f90flds **, double *);
* void b_sqrt_inv_mult(const f90bmat * const *, double *, const f90flds **);
* void b_sqrt_inv_mult_ad(const f90bmat * const *, const double *, f90flds **);

// ------------------------------------------------------------------------
//  Variables
// ------------------------------------------------------------------------
//    Tomas, Deborah, Mike, Yannick, George, John, Mats
// ------------------------------------------------------------------------

  void var_clone(const f90vars * const *, f90vars **);
  void var_delete(f90vars **);

// ------------------------------------------------------------------------
//  Locations
// ------------------------------------------------------------------------
  void loc_delete(f90locs **);
  void loc_nobs(const f90locs * const *, int &);

// ------------------------------------------------------------------------
//  Local Values (GOM)
// ------------------------------------------------------------------------
  void gom_create(f90goms **);
  void gom_delete(f90goms **);

// ------------------------------------------------------------------------
//  Observations Operators
// ------------------------------------------------------------------------
//    Deborah, Anne
// ------------------------------------------------------------------------

  void obs_setup(f90obso **, const XmlDom **, f90help **);
  void obs_delete(f90obso **);

  void obs_equiv   (const f90obso * const *, const f90goms **, f90ovec **);
  void obs_equivtra(const f90obso * const *, const f90goms **,
                    f90ovec **, f90goms **);
  void obs_equiv_tl(const f90obso * const *, const f90goms **,
```

```
                    f90ovec **, const f90goms **);
  void obs_equiv_ad(const f90obso * const *, f90goms **,
                    const f90ovec **, const f90goms **);

  void obs_locations(const f90obso * const *,
                     const DateTime **, const DateTime **, f90locs **);
  void obs_getgom(const f90obso * const *,
                  const DateTime **, const DateTime **, f90goms **);
  void obs_gettra(const f90obso * const *, f90goms **);
  void obs_checks(const f90obso * const *,
                  const DateTime **, const DateTime **, int &);
* void obs_generate(const f90obso * const *, const XmlDom **, f90ovec **);

// ------------------------------------------------------------------------
//  Observation Errors
// ------------------------------------------------------------------------
//    Alan, SAT?
// ------------------------------------------------------------------------

  void r_setup(f90rmat **, const XmlDom **);
  void r_delete(f90rmat **);

  void r_linearize(f90rmat **, const XmlDom **);

  void r_mult(const f90rmat * const *, const f90ovec **, f90ovec **);
  void r_imul(const f90rmat * const *, const f90ovec **, f90ovec **);

* void r_sqrt   (const f90rmat * const *, f90ovec **, const double *);
* void r_sqrtad (const f90rmat * const *, const f90ovec **, double *);
* void r_sqrti  (const f90rmat * const *, double *, const f90ovec **);
* void r_sqrtiad(const f90rmat * const *, const double *, f90ovec **);

// ------------------------------------------------------------------------
//  Observation Vectors
// ------------------------------------------------------------------------
//    Deborah, Anne
// ------------------------------------------------------------------------

  void obsvec_create(const f90ovec * const *, f90ovec **);
// diff_create is optional: it can be replaced by create followed by sub.
  void obsvec_diff_create(f90ovec **, const f90ovec * const *, const f90ovec *
const *);
  void obsvec_delete(f90ovec **);

  void obsvec_assign(f90ovec **, const f90ovec * const *);
  void obsvec_assign_scal(f90ovec **, const double &);
  void obsvec_mul_scal(f90ovec **, const double &);
  void obsvec_add(f90ovec **, const f90ovec * const *);
  void obsvec_sub(f90ovec **, const f90ovec * const *);
  void obsvec_mul(f90ovec **, const f90ovec * const *);
  void obsvec_div(f90ovec **, const f90ovec * const *);
  void obsvec_axpy(f90ovec **, const double &, const f90ovec * const *);
  void obsvec_random(f90ovec **);
  void obsvec_dotprod(const f90ovec * const *, const f90ovec * const *, double &
);
  void obsvec_minmaxavg(const f90ovec * const *, double &, double &, double &);
  void obsvec_nobs(const f90ovec * const *, int &);

  void obsvec_read(f90ovec **, const f90help * const *,
                   const int &, const char *, const int &, const char *);
```

```
  void obsvec_write(const f90ovec **, f90help **,
                    const int &, const char *, const int &, const char *);

// ------------------------------------------------------------------------
//  Observation Handler
// ------------------------------------------------------------------------
//    Anne
// ------------------------------------------------------------------------

  void obsdb_setup(f90help **, const XmlDom **);
  void obsdb_delete(f90help **);

// ------------------------------------------------------------------------
//  Observation Auxiliary Variables (VarBC, TOVSCV...)
// ------------------------------------------------------------------------
//    Alan? Niels? Dick? SAT?
// ------------------------------------------------------------------------

// ObsAux needs the same interface as ObsVector
// and a covariance matrix with the same interface as R.

}

}
#endif
```