From Integrated to Object-Oriented

Yannick Trémolet and Mike Fisher

ECMWF

3 October 2012

Thanks to many people who have contributed to the project:
Tomas Wilhelmsson, Deborah Salmond, John Hague, George Mozdzynski, Alan Geer, Anne Fouilloux, Mats Hamrud, code reviewers...

4D-Var and Scalability

- The current implementation of 4D-Var is not scalable enough for the future.
- The code can be optimized routine by routine to increase scalability only up to a certain point.
- Significant leaps in the level of available parallelism can only be achieved through scientific progress in the formulation of the problem and minimization algorithms.
- The IFS is not flexible enough to test such ideas.

Code Design

- Since the IFS was designed, in the late 1980's, the software industry has progessed tremendously.
- We are not the only ones who want a code that is flexible, efficient and reliable.
- The technique that has emerged to answer these needs is called object-oriented programming.
- We are using the latest hardware technology, we should also be looking at recent mature software development technology.
- We have started to re-design our system using this technology in the

  Object-Oriented Prediction System (OOPS).

Object-Oriented Programming

- Organize programs around the data, not the algorithms.
- An object is:
  - Data,
  - Methods that act on the data.
- There is no access to the data other than through the defined methods.

Object-Oriented Programming

- Organize programs around the data, not the algorithms.
- An object is:
  - Data,
  - Methods that act on the data.
- There is no access to the data other than through the defined methods.
- Example: Atmospheric state
  - Data
    - Spectral,
    - Grid-point (on various types of grid).
  - Methods:
    - Read and write,
    - Interpolate (to points or change resolution),
    - Move forward in time (forecast).

Object-Oriented Programming

- Organize programs around the data, not the algorithms.
- An object is:
  - Data,
  - Methods that act on the data.
- There is no access to the data other than through the defined methods.
- Example: Atmospheric state
  - Methods:
    - Read and write,
    - Interpolate (to points or change resolution),
    - Move forward in time (forecast).

Object-Oriented Programming

- Organize programs around the data, not the algorithms.
- An object is:
  - Data,
  - Methods that act on the data.
- There is no access to the data other than through the defined methods.
- Example: Atmospheric state
  - Methods:
    - Read and write,
    - Interpolate (to points or change resolution),
    - Move forward in time (forecast).
- Looking at the methods, notice that "Atmospheric" can be replaced by "Oceanic" or any other system of interest.
- One object can be replaced by another: OO encourages abstraction.

Object-Oriented Programming

- Organize programs around the data, not the algorithms.
- An object is:
  - Data,
  - Methods that act on the data.
- There is no access to the data other than through the defined methods.
- Example: Atmospheric state
  - Methods:
    - Read and write,
    - Interpolate (to points or change resolution),
    - Move forward in time (forecast).
- Looking at the methods, notice that "Atmospheric" can be replaced by "Oceanic" or any other system of interest.
- One object can be replaced by another: OO encourages abstraction.
- There is much more to OO programming...

OOPS Design Principles

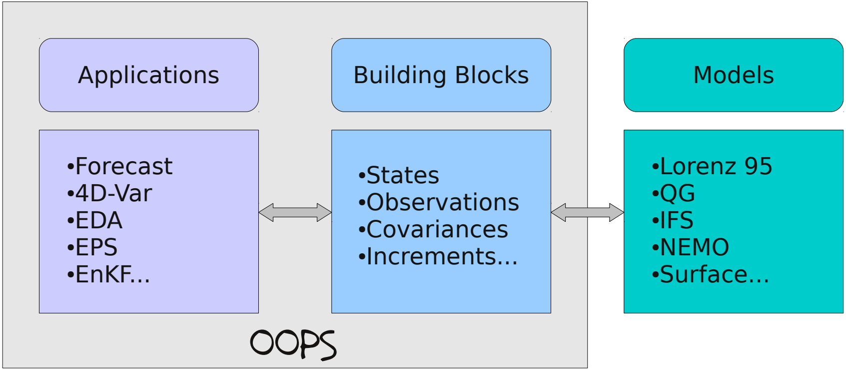
- The problem can be broken into manageable pieces:
  - Data assimilation (or ensemble prediction) can be described without knowing the specifics of a model or observations.
  - Minimisation algorithms can be written without knowing the details of the matrices and vectors involved.

- All aspects exist but scientists focus on one aspect at a time.

- The code should reflect this: problems that are orthogonal to each other should be in independent parts of the code.
  - The alternatives are code duplication or ever more complex `IF` statements.

- OOPS starts by applying these principles at the highest level.

What is OOPS?



- The high levels Applications use abstract building blocks.
- The Models implement the building blocks.
- OOPS is independent of the Model being driven.

OOPS Design

- OOPS is independent of the model being driven.

- Flexibility (including yet unknown future development) requires that this goes both ways.

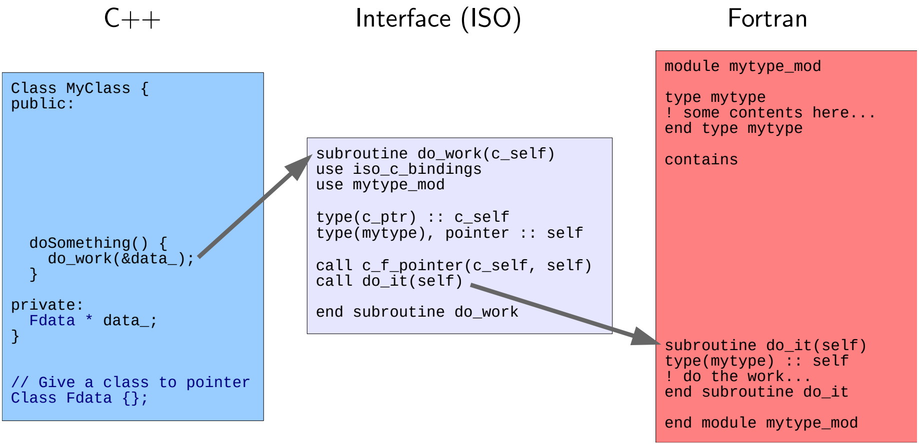
- The Models do not know about the high level algorithm currently being run:
    - All actions are driven by OOPS,
    - All data, input and output, is passed by arguments.

- Models interfaces must be general enough to cater for all cases, and detailed enough to be able to perform the required actions.
    - Abstraction.

From IFS to OOPS

- The high level (object-oriented) code is implemented in C++.
- A model can be written in any language as long as:
  - It provides an interface to the abstract C++ building blocks.
  - It does what it says on the tin (and nothing else).
- The IFS “model” will be re-used and remain in Fortran.
  - Capitalize on many years of investment in the code.
  - Fortran is a good language for numerical code.
- All data is progressively moved from `modules` (global variables) to argument lists:
  - Encapsulation in derived types.
- The result is self-contained parts of the IFS that can be used by OOPS.
  - The Fortran code called by OOPS can still be called from within the IFS.
  - No divergence of code or blocking points.

Encapsulating Fortran Code in C++ Classes



C++

```
Class MyClass {
public:







  doSomething() {
    do_work(&data_);
  }

private:
  Fdata * data_;
}


// Give a class to pointer
Class Fdata {};
```

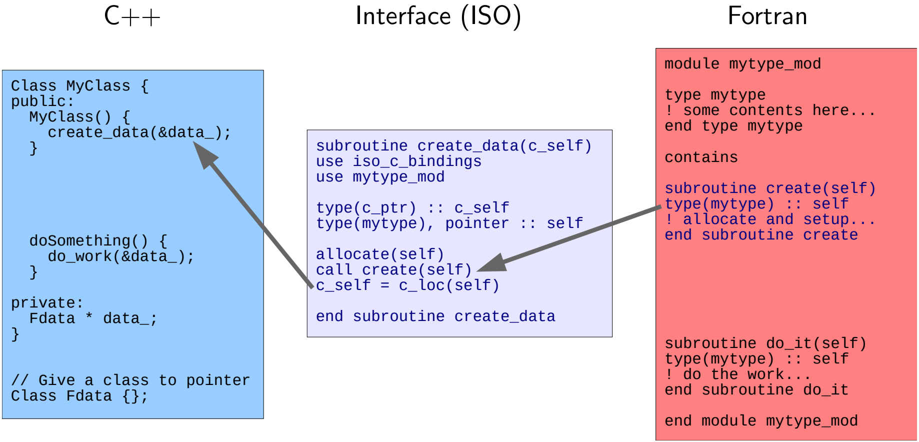
Interface (ISO)

```
subroutine do_work(c_self)
use iso_c_bindings
use mytype_mod

type(c_ptr) :: c_self
type(mytype), pointer :: self

call c_f_pointer(c_self, self)
call do_it(self)

end subroutine do_work
```

Fortran

```
module mytype_mod

type mytype
! some contents here...
end type mytype

contains











subroutine do_it(self)
type(mytype) :: self
! do the work...
end subroutine do_it

end module mytype_mod
```

Encapsulating Fortran Code in C++ Classes



C++

```
Class MyClass {
public:
  MyClass() {
    create_data(&data_);
  }




  doSomething() {
    do_work(&data_);
  }

private:
  Fdata * data_;
}


// Give a class to pointer
Class Fdata {};
```

Interface (ISO)

```
subroutine create_data(c_self)
use iso_c_bindings
use mytype_mod

type(c_ptr) :: c_self
type(mytype), pointer :: self

allocate(self)
call create(self)
c_self = c_loc(self)

end subroutine create_data
```

Fortran

```
module mytype_mod

type mytype
! some contents here...
end type mytype

contains

subroutine create(self)
type(mytype) :: self
! allocate and setup...
end subroutine create





subroutine do_it(self)
type(mytype) :: self
! do the work...
end subroutine do_it

end module mytype_mod
```

- No static variable of type `mytype` is declared in the module!

Encapsulating Fortran Code in C++ Classes

C++

```
Class MyClass {
public:
  MyClass() {
    create_data(&data_);
  }

  ~Myclass() {
    delete_data(&data_);
  }

  doSomething() {
    do_work(&data_);
  }

private:
  Fdata * data_;
}


// Give a class to pointer
Class Fdata {};
```

Interface (ISO)

```
subroutine do_work(c_self)
use iso_c_bindings
use mytype_mod

type(c_ptr) :: c_self
type(mytype), pointer :: self

call c_f_pointer(c_self, self)
call do_it(self)

end subroutine do_work
```

Fortran

```
module mytype_mod

type mytype
! some contents here...
end type mytype

contains

subroutine create(self)
type(mytype) :: self
! allocate and setup...
end subroutine create

subroutine delete(self)
type(mytype) :: self
! deallocate...
end subroutine delete

subroutine do_it(self)
type(mytype) :: self
! do the work...
end subroutine do_it

end module mytype_mod
```

- No static variable of type `mytype` is declared in the module!
- The Fortran module does not know about C++: it is fully usable in the rest of the Fortran code.

OOPS Granularity

- OOPS requires the definition of a small number of classes:
  - In model space:
    1. State
    2. Increment
    3. ErrorCovariance
    4. Trajectory
  - In observation space:
    5. ObsOperator
    6. ObsTraj
    7. ObsVector
  - To make the link:
    8. LocalModelValues
    9. Locations

- This leads to less than 100 methods (Fortran interfaces) to be implemented.

- Observation and model errors (biases) will be added.

Fortran and C++

- If Fortran modules implement C++ classes, can we implement these modules and stay in Fortran?
- Most of the work for going from IFS to OOPS is in implementing the Fortran modules.
- It brings only a fraction of the benefits:
  - No polymorphism:
    C++ has run-time (inheritance) and compile-time (templates) polymorphism, Fortran derived types cannot be substituted one for another.
  - No unit testing:
    C++ (and OO languages) have support for automated testing of classes.
  - Old (bad) habits would remain...
- A large fraction of the work on the C++ side is already done.

OOPS Status

- OOPS is working and is being used for scientific studies:
  - 3D-Var, 4D-Var, weak-constraint 4D-Var with Lorenz 95 and QG models.
  - 3D-Var with IFS (AMSU-A only).
- Most of the work is in refactoring the Fortran code (in Fortran).
  - 10+ person-years out of 12 to 13 person-years in total for the project.
  - The rest is training, coordination and C++ development.
- Refactored Fortran code will be easier to maintain.
- OOPS is small but brings very high flexibility while maintaining efficiency.
- OOPS prepares us for scalability improvements and future scientific developments.