
Mike Fisher

ECMWF

15 December 2010
Outline

1. Main Characteristics of Data Assimilation
2. What is Wrong with our Existing Codes?
3. What do we need from a Data Assimilation Framework?
4. What is Object Oriented Programming
5. Objects for Data Assimilation
6. Which Language
7. Status and Conclusions
Outline

1. Main Characteristics of Data Assimilation
2. What is Wrong with our Existing Codes?
3. What do we need from a Data Assimilation Framework?
4. What is Object Oriented Programming
5. Objects for Data Assimilation
6. Which Language
7. Status and Conclusions
Characteristics of Data Assimilation

Data Assimilation is the combination of models and observations to provide an estimate of the state of a system (e.g. the atmosphere).

A variety of algorithms exist to perform this task:

- **Variational Methods:**
  - 3D-Var
  - 4D-Var
  - Weak-constraint 4D-Var
  - PSAS
  - etc.

- **Ensemble Kalman Filters:**
  - LETKF
  - Ensemble square-root filters
  - Perturbed-observation filters
  - etc.
Characteristics of Data Assimilation

All assimilation algorithms can be expressed in terms of a relatively small number of entities:

- $\mathbf{x}$: The state of the system (e.g. the model fields).
- $\delta \mathbf{x}$: A correction (increment) to the state.
- $\mathbf{y}$: A vector of observations.
- $\mathcal{H}$: An observation operator: $\mathcal{H}(\mathbf{x})$ is the model-equivalent of $\mathbf{y}$.
- $\mathcal{M}_{t_1,t_2}$: An numerical model (integrated from time $t_1$ to time $t_2$).
- $\mathcal{H}, \mathcal{M}_{t_1,t_2}$: Linearisations of $\mathcal{H}$ and $\mathcal{M}_{t_1,t_2}$.
- $B, R, Q$: Covariance matrices.

For example, 4D-Var defines the analysis as the state that minimizes:

$$J(\mathbf{x}) = \frac{1}{2} (\mathbf{x} - \mathbf{x}_b)^T B^{-1} (\mathbf{x} - \mathbf{x}_b)$$

$$+ \frac{1}{2} \sum_{k=0}^{N} (\mathcal{H}_k(\mathcal{M}_{t_0,t_k}(\mathbf{x})) - \mathbf{y}_k)^T R_k^{-1} (\mathcal{H}_k(\mathcal{M}_{t_0,t_k}(\mathbf{x})) - \mathbf{y}_k)$$
Characteristics of Data Assimilation

We can derive new assimilation algorithms by re-arranging the equations:

- **PSAS:**

  \[
  \begin{align*}
  w & = (HBH^T + R)^{-1}(y - Hx_b) \\
  x_a & = x_b + BH^T w
  \end{align*}
  \]

- **Perturbed-observation Ensemble Kalman Filter:**

  \[
  \begin{align*}
  x_{b}^{(k)} & = M_{t_{k-1}, t_k} \left( x_a^{(k)}(t_{i-1}) \right) + \eta^{(k)} \\
  P^b & = \frac{1}{K-1} \sum_{k=1}^{K} (x_b^{(k)} - \bar{x}_b)(x_b^{(k)} - \bar{x}_b)^T \\
  K & = P^b H^T \left( HP^b H^T + R \right)^{-1} \\
  x_a^{(k)}(t_i) & = x_b^{(k)} + K \left( y + \epsilon^{(k)} - Hx_b^{(k)} \right)
  \end{align*}
  \]
Outline

1. Main Characteristics of Data Assimilation
2. What is Wrong with our Existing Codes?
3. What do we need from a Data Assimilation Framework?
4. What is Object Oriented Programming
5. Objects for Data Assimilation
6. Which Language
7. Status and Conclusions
What is Wrong with our Existing Codes?

The current data assimilation code used at ECMWF and Météo France suffers from two problems:

1. It is intimately tied to a single model.
   - It is not possible to extract the assimilation code and use it with another model.
   - (Options to run a few simplified models exist within the IFS code, but these are poorly maintained and difficult to use. Introducing a new model into the framework would be difficult.)

2. The vectors and operators required to express the algorithms are not clearly identified.
   - There is no single entity corresponding to the observation vector $y$.
   - $H^T R^{-1} H$ corresponds to a single subroutine. The individual operators are not separately available.
   - The model state exists as loose collection of structures corresponding to various spectral fields and gridpoint fields.
   - There is a single model state. Many algorithms (e.g. ensemble Kalman filters) require several model states to exist simultaneously.
What is Wrong with our Existing Codes?

- The limitations of the existing code make it difficult to develop new algorithms. This is becoming a real barrier to scientific development in data assimilation.

- The close linking between the model and the data assimilation code leads to high maintainance cost. Code changes that should only impact the model can have unforseen consequences for data assimilation, and vice versa.

- A single monolithic code leads to a steep learning curve for new scientists and visitors. Collaboration with external (e.g. university) groups is difficult.

- The current uncertainty about which data assimilation methods will be used in the future requires that we have a more flexible code.

- It should be possible to develop algorithms in simple models, and then transfer them directly to the IFS, without requiring months of re-coding.
Outline

1. Main Characteristics of Data Assimilation
2. What is Wrong with our Existing Codes?
3. What do we need from a Data Assimilation Framework?
4. What is Object Oriented Programming
5. Objects for Data Assimilation
6. Which Language
7. Status and Conclusions
What do we need from a Data Assimilation Framework?

- The code should provide access to the basic building blocks of data assimilation: states, increments, covariance matrices, observation operators, models.
- It should be possible to re-combine these elements in different ways to express different algorithms.
- The building blocks should be independent from each other.
- It should be possible to understand one building block without knowing the others.
- The data assimilation framework should be isolated from the specific implementations of these building blocks.
- It should be possible to use the same assimilation code with a variety of different models, observations, and covariance specifications.
What do we need from a Data Assimilation Framework?

Data Assimilation is not alone in its need to deal with complex, interacting codes, or in its requirements for flexibility, re-usability and reliability.

The software industry also faces these challenges.

The methodology, almost universally adopted within the software industry, that has developed to address these challenges is Object Oriented Programming.

We believe that Object Oriented programming is the only way to achieve our aims of a truly flexible, model-independent data assimilation framework.
Outline

1. Main Characteristics of Data Assimilation

2. What is Wrong with our Existing Codes?

3. What do we need from a Data Assimilation Framework?

4. What is Object Oriented Programming

5. Objects for Data Assimilation

6. Which Language

7. Status and Conclusions
What is Object Oriented Programming

A typical Fortran code can be described as maximum information at the top level:

- All possible components are included in the highest level, most visible data structures.
- Each component is switched on/off depending on the configuration.
- The code consists largely of subroutines and functions that act on this globally-defined data.
- Since the data structures are globally visible, adding a new component requires widespread changes to the code.

By contrast, an object oriented code has minimum information at the top:

- The high level descriptions of the data contain a minimum of detail.
- Only the most basic properties are described at the top level
- Changing the implementation (e.g. adding a component) affects only a small amount of implementation-specific code.
Outline

1. Main Characteristics of Data Assimilation
2. What is Wrong with our Existing Codes?
3. What do we need from a Data Assimilation Framework?
4. What is Object Oriented Programming
5. Objects for Data Assimilation
6. Which Language
7. Status and Conclusions
Objects for Data Assimilation

What objects and properties do we need to describe data assimilation?

State:

- Create, Destroy, Copy.
- Add an increment.
- Propagate forward in time using a model.
- Interpolate (e.g. to observation locations).
- Change resolution.
- Read and Write.
class State {
public:
    explicit State(Model &);
    State(const State &);
    ~State();
    State & operator=(const State &);
    State & operator+=(const Increment &);
    void propagate();
    void interpolate(const Locations &, AnyColumns &) const;
    void changeResolution(const State &);
    void read(const XmlDom &);
    void write(const XmlDom &) const;
private:
    FieldSet * fields_;    
    Model * model_; 
};
State related classes

- **State**
  - propagate
  - interpolate
  - changeResolution
  - operator+=
  - read
  - write
  - getDateTime

- **Increment**
  - propagateTL
  - propagateAD
  - interpolateTL
  - interpolateAD
  - operator+
  - operator-
  - read
  - write
  - getDateTime

- **Model**
  - tstep
  - tstepTL
  - tstepAD
  - adjointTest

- **FieldSet**
  - interpolate
  - interpolateTL
  - interpolateAD
  - changeResolution
  - operator=
  - operator+
  - read
  - write
  - time

- **Lorenz**
  - tstep
  - tstepTL
  - tstepAD

- **QG**
  - tstep
  - tstepTL
  - tstepAD

- **SpectralField**
  - interpolate
  - ...

- **GaussianGrid**
  - interpolate
  - ...

---

Mike Fisher (ECMWF)  Object Oriented Prediction System  15 December 2010
Objects for Data Assimilation

Observation vector:

- Create, Destroy, Copy.
- Read and Write.
- Calculate model-equivalents of the observations.
class Observation {
public:
    explicit Observation(const XMLDom &);
    Observation(const Observation &);
    ~Observation();

    void obsOperator(const State &, const utils::Duration &, const utils::Duration &);

    void save() const;
private:
    const XMLDom * config_;
    std::vector<ObsType> obs_; 
};
Observation related classes

ObsType
- obsOperator
- generate
- save
- prtInfos

DepType
- obsOperatorTL
- obsOperatorAD
- operator=
- operator+
- diff
- save
- prtInfos

ObsEquivalent
- obsEquiv
- obsEquivTL
- obsEquivAD
- adjointTest
- distribute

L95Obs
- obsEquiv
- obsEquivTL
- obsEquivAD
- distribute

QGwind
- obsEquiv
- obsEquivTL
- obsEquivAD
- distribute

ObsHandler
- putdb
- getdb
- datetimes
- locations

ODB
- getdb
- putdb

ObsTable
- getdb
- putdb
Objects for Data Assimilation

Background Error Covariance Matrix:

- Create, Destroy.
- Multiply an increment by $B$, $B^{-1}$, $B^{-1/2}$.
- Multiply (part of) a control vector by $B^{1/2}$.

Other objects (Increments, FieldSets, Models, etc.) are similarly defined in terms of their basic properties.
Objects for Data Assimilation
The basic properties are enough to define the data assimilation algorithm.

- Each component is responsible for its own area of responsibility.
- Data members are private, and access to them is restricted through a well-defined public interface.
- Information hiding enforces independence of components.

Example – Observation operator:

```c
Locations locs(oper_\rightarrow coordinates());
AnyColumns gom(oper_\rightarrow variables(), nobs);
xx.interpolate(locs, gom);
oper_\rightarrow obsEquiv(gom, obsval_);
```

- The state provides model values at the requested locations.
- The observation operator is independent of the model grid.
Outline

1. Main Characteristics of Data Assimilation
2. What is Wrong with our Existing Codes?
3. What do we need from a Data Assimilation Framework?
4. What is Object Oriented Programming
5. Objects for Data Assimilation
6. Which Language
7. Status and Conclusions
Which Language?

- Fortran 90/95 allows some object oriented features to be simulated.
- However, it does not allow the level of abstraction we think is necessary to achieve a fully model-independent data assimilation framework.
- We evaluated Fortran 2003 — to the extent that we wrote an early version of OOPS entirely in Fortran 2003.
- Fortran 2003 lacks some OO features (e.g. constructors).
- We found Fortran 2003 to be verbose (e.g. requiring extensive use of SELECT TYPE blocks).
- Fortran 2003 is still very poorly supported. It is not available on all platforms, and we frequently encountered compiler bugs.
We selected C++ for the object-oriented layer.

C++ is well supported on all platforms.

- Good compilers, tools (e.g. debuggers), libraries (e.g. Boost).

However, most models we will interface with are written in Fortran.

Fortran is probably the best choice for array-based applications.

Hence, we have adopted a mixed language approach.

To achieve this, we make heavy use of the Fortran 2003 feature ISO_C_BINDINGS. Fortunately, this is implemented already in almost all Fortran 90 compilers.
Mixed Language Example — Copy a FieldSet for the QG model:

- QGFields is derived from FieldSet.
- The copy constructor for QGFields calls the Fortran Subroutine `qg_field_copy_f90` to do the copy (and allocate).

```cpp
QgFields::QgFields(const QgFields & other) {
    const F90Fields * rhflds = other.flds_;  
    qg_field_copy_f90(&flds_,&rhflds);
}
```

- The Fortran layer knows about the internal structure of a QGField.
- C++ knows nothing of the internal structure.
- C++ sees only pointers, representing the address of the QGField in memory. It does not need to know any more!
Which Language?

The Fortran code to copy a QGField looks like this:

```fortran
subroutine qg_field_copy_f90(c_self, c_other)
implicit none

type(c_ptr), intent(inout) :: c_self
type(c_ptr), intent(in)    :: c_other
type(qg_field), pointer    :: self, other

call c_f_pointer(c_other, other)

allocate(self) ! allocate space for the copy

self%nx=other%nx   ! copy
self%ny=other%ny
self%gfld3d(:, :, :) = other%gfld3d(:, :, :)
...

c_self = c_loc(self)
return
end subroutine
```
Outline

1. Main Characteristics of Data Assimilation
2. What is Wrong with our Existing Codes?
3. What do we need from a Data Assimilation Framework?
4. What is Object Oriented Programming
5. Objects for Data Assimilation
6. Which Language
7. Status and Conclusions
Status

We currently have two models available:

- A two-level quasi-gestrophic model.
- The Lorenz 1996 40-variable model.

The model is chosen at compile time, by choosing which model to instantiate. No other code changes are required.

We have implemented two versions of weak-constraint 4D-Var.

By February, we expect to have incorporated Serge Gratton’s CGMOD and RPCG minimisation algorithms (in addition to CG).

We have started work on adapting IFS/Arpege to work in the OOPS framework.

We aim to demonstrate an OOPS-IFS 3D-Var next summer. 4D-Var should follow within \( \approx 15 \) months.
Conclusions

- Data assimilation algorithms can be expressed in an abstract form, independent of any particular model.
- Expressing algorithms in this way allows a single assimilation framework to be applied to many different models.
- By providing the basic building blocks, a wide range of algorithms can be implemented.
- Object Oriented Programming is widely recognised in the software industry as the best way to deal with complex, interacting codes.
- We expect the OOPS framework to lead to greater productivity, lower maintenance costs, and improved opportunities for collaboration with external groups.