The Joint Effort for Data assimilation Integration (JEDI)

OOPS Data flow; Interface classes

Joint Center for Satellite Data Assimilation (JCSDA)

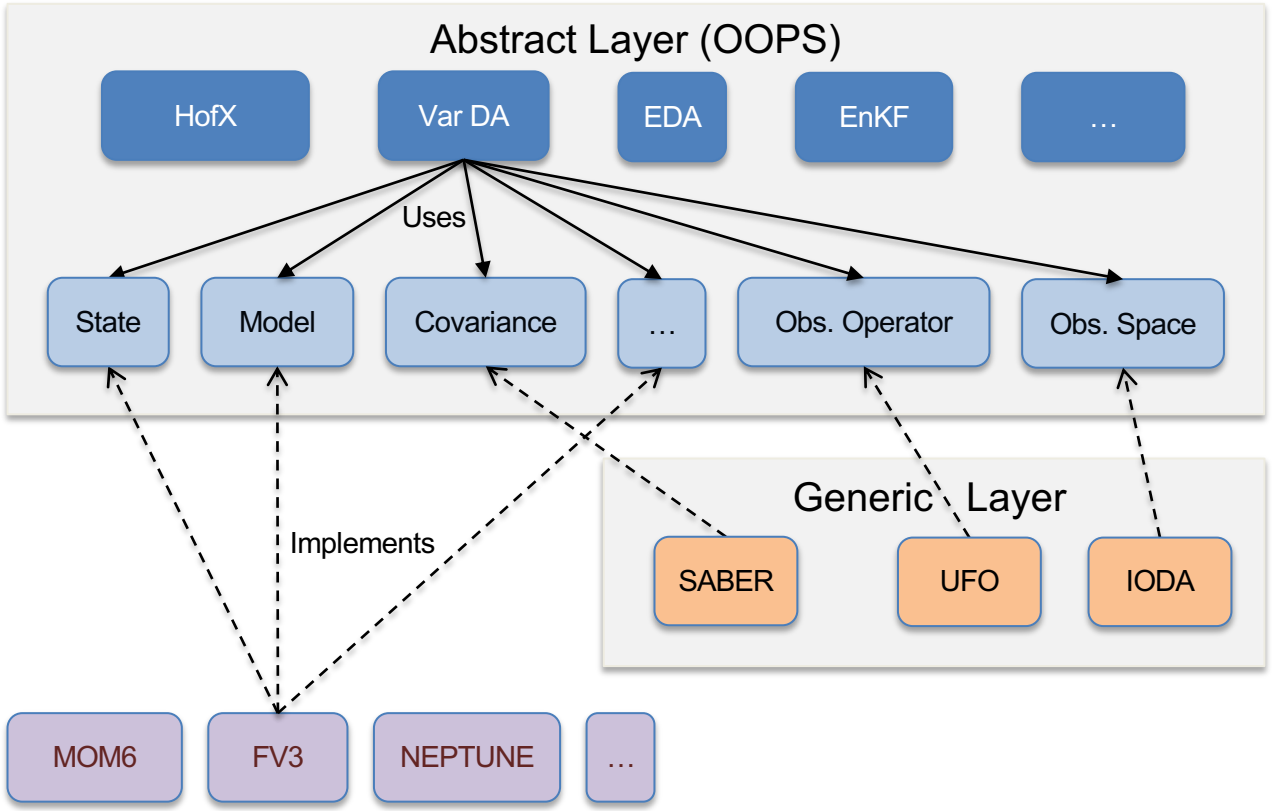
JEDI Academy – 21-25 June 2021

JEDI: Motivations and Objectives

Develop a unified data assimilation system:

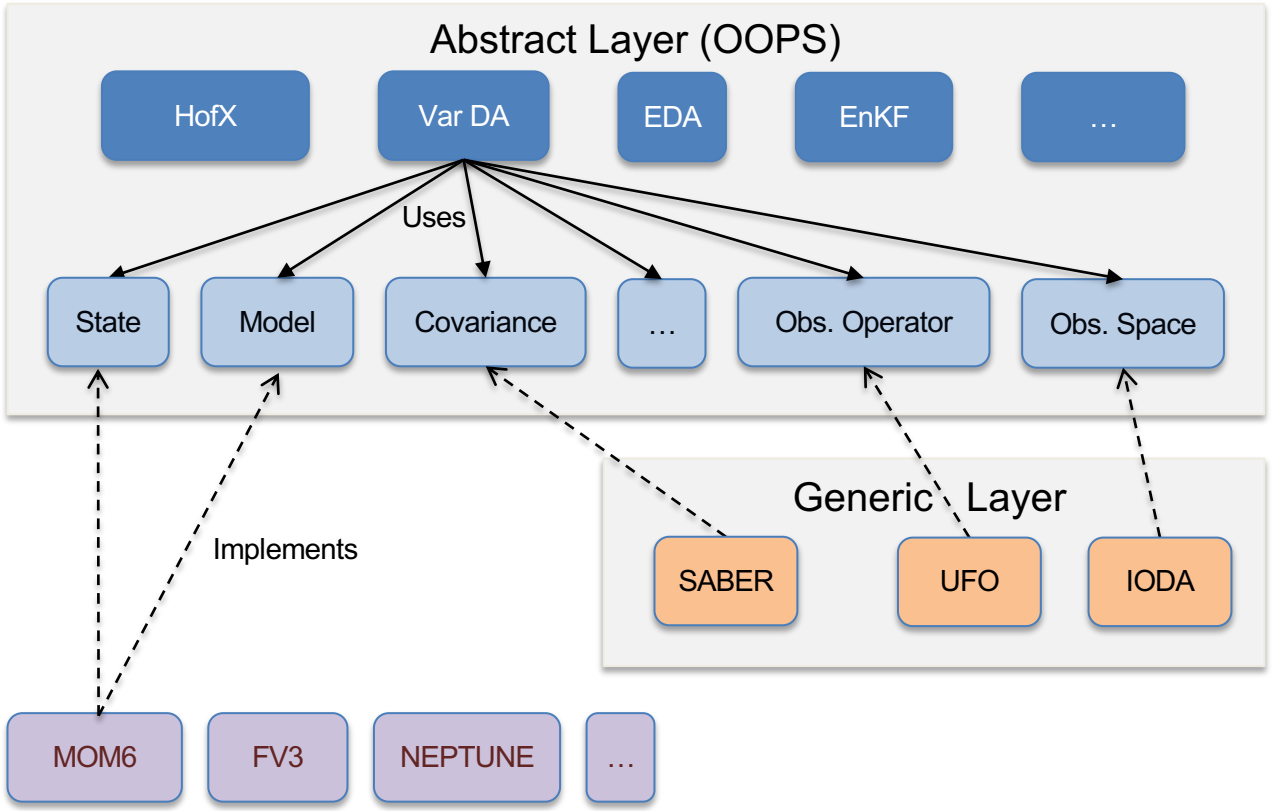
- From toy models to Earth system coupled models
- Unified observation (forward) operators (UFO)
- For research and operations (including O2R2O)
- Share as much as possible without imposing one approach (one system, multiple methodologies/configurations)

JEDI: Abstraction and Genericity



Abstract, model-agnostic DA system

JEDI: Abstraction and Genericity



Abstract, model-agnostic DA system

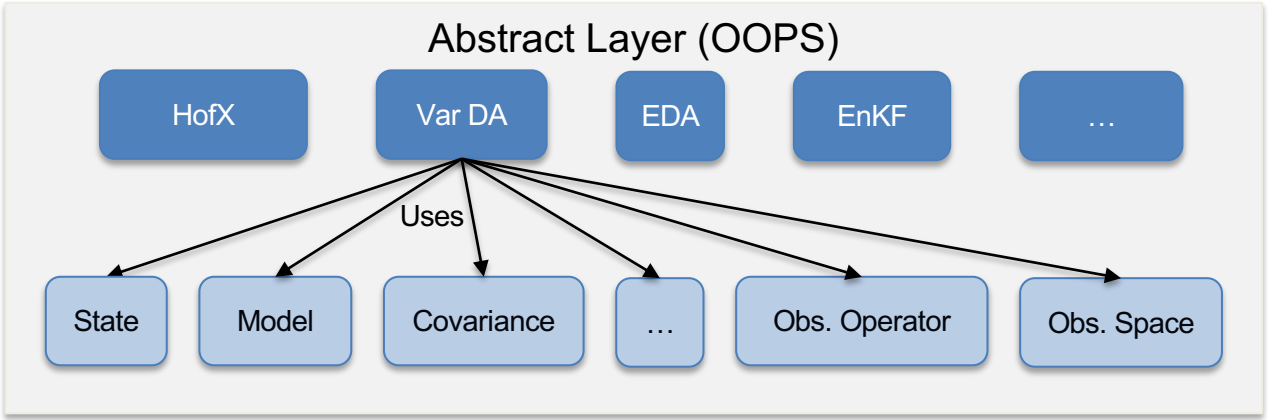
Separation of concerns

- **OOPS** is independent of the underlying model and physical system.
- **The implementations** do not know about the high level algorithm:
  - All actions driven by the top level code,
  - All data, input and output, passed by arguments.
- **Interfaces** must be general enough to cater for all cases, and detailed enough to be able to perform the required actions.

Separation of concerns

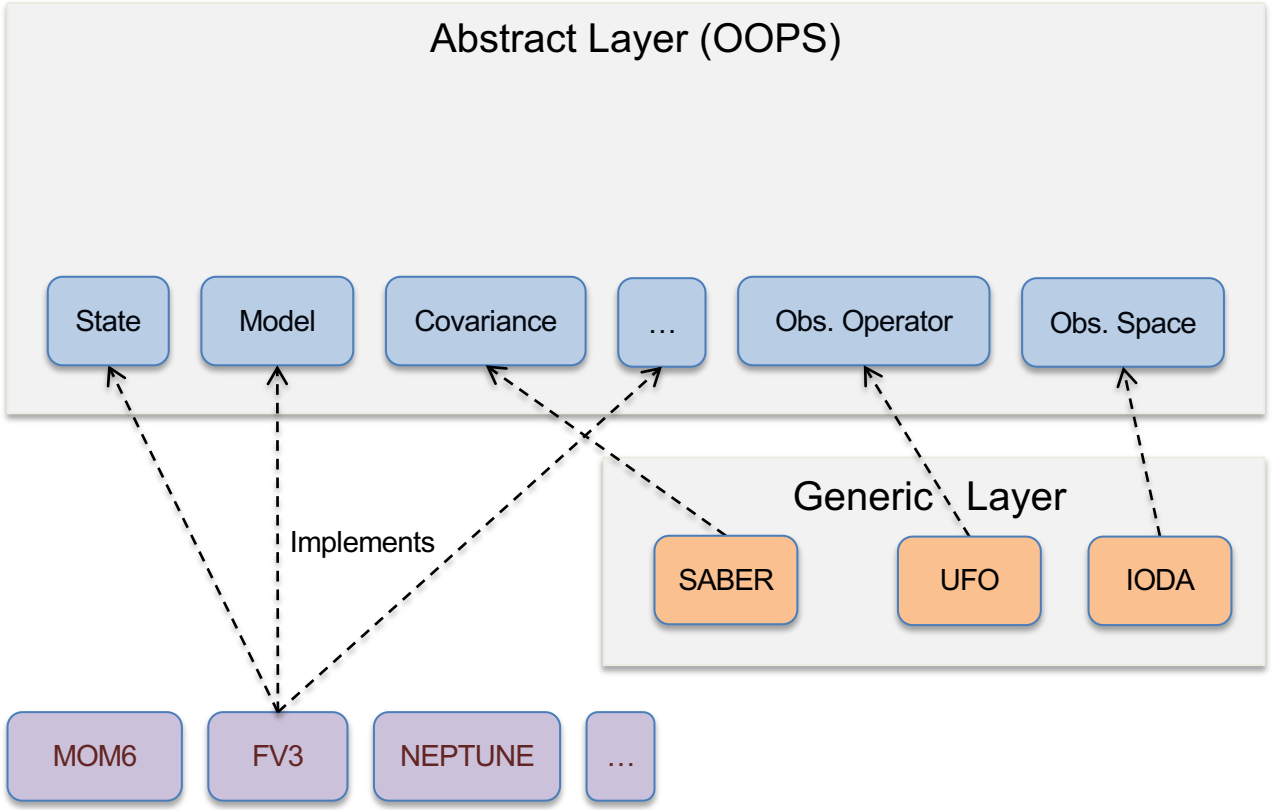
Generic Algorithms

Abstract Interfaces



Abstract, model-agnostic DA system

Separation of concerns



Abstract Interfaces

Generic Implementations

Specific Implementations

Abstract, model-agnostic DA system

OOPS Analysis and Design

- All data assimilation methods require the same limited number of entities.
- For future (unknown) developments these entities should be easily reusable.
- These entities are the basic (abstract) classes that define the system.
- No details about how any of the operations are performed, how data is stored or what the model represents: separation of concerns.

Basic building blocks for DA

What is data assimilation?

Data assimilation is finding the best estimate (analysis) of the state of a system given a previous estimate of the state (background) and recent observations of the system.

use variational assimilation and minimize:

$$J(\Delta x) = \frac{1}{2}\Delta x^T \mathbf{B}^{-1}\Delta x + \frac{1}{2}\left(y_o - H(x_b + \Delta x)\right)^T \mathbf{R}^{-1}(y_o - H(x_b + \Delta x))$$

use Kalman filter:

$$\Delta x_a = \mathbf{BH^T}\left(\mathbf{HBH^T} + \mathbf{R}\right)^{-1}(y_o - H(x_b))$$

Model space: State

$$J(\Delta x) = \frac{1}{2}\Delta x^T \mathbf{B}^{-1}\Delta x + \frac{1}{2}\left(y_o - H(x_b + \Delta x)\right)^T \mathbf{R}^{-1}(y_o - H(x_b + \Delta x))$$

$$\Delta x_a = \mathbf{BH^T}\left(\mathbf{HBH^T} + \mathbf{R}\right)^{-1}(y_o - H(x_b))$$

Examples: background, analysis, forecast state

Operations allowed on state:

- Input, output (raw or post-processed).
- Move forward in time (using the Model).
- Copy, assign.

From DA point of view no need to know how operations are performed, or how states are represented and stored.

Model space: Increment

$$J(\Delta x) = \frac{1}{2}\Delta x^T \mathbf{B}^{-1} \Delta x + \frac{1}{2}(y_o - H(x_b) - \mathbf{H}\Delta x)^T \mathbf{R}^{-1}(y_o - H(x_b) - \mathbf{H}\Delta x)$$

$$\Delta x_a = \mathbf{B}\mathbf{H}^{\mathbf{T}}\left(\mathbf{H}\mathbf{B}\mathbf{H}^{\mathbf{T}} + \mathbf{R}\right)^{-1}(y_o - H(x_b))$$

Examples: perturbation to a state, analysis increment, ensemble perturbation

Operations allowed on Increments:

- Basic linear algebra operators,
- Evolve forward in time linearly and backwards with adjoint.
- Compute as difference between states, add to state.

Observations

$$J(\Delta x) = \frac{1}{2}\Delta x^T \mathbf{B}^{-1}\Delta x + \frac{1}{2}\left(y_o - H(x_b + \Delta x)\right)^T \mathbf{R}^{-1}\left(y_o - H(x_b + \Delta x)\right)$$

$$\Delta x_a = \mathbf{BH^T}\left(\mathbf{HBH^T} + \mathbf{R}\right)^{-1}\left(y_o - H(x_b)\right)$$

Examples: observation values, model-simulated observation values.

Operations allowed on Observations:

- Input, output.
- Simulate observation given a state (observation operator).
- Copy, assign.

From DA point of view no need to know how operations are performed, or how observations are represented and stored.

Departures

$$J(\Delta x) = \frac{1}{2}\Delta x^T \mathbf{B}^{-1}\Delta x + \frac{1}{2}(y_o - H(x_b) - \mathbf{H}\Delta x)^T \mathbf{R}^{-1}(y_o - H(x_b) - \mathbf{H}\Delta x)$$

Examples: departures, ensemble perturbations in the observation space.

Operations allowed on Departures

- Basic linear algebra operators,
- Compute as difference between observations, add to observations,
- Compute linear variations in observation equivalent as a result of variations of the state (linearized observation operator).
- Output (for diagnostics).

Operators

$$J(x) = \frac{1}{2}(x_0 - x_b)^T \mathbf{B}^{-1}(x_0 - x_b) + \frac{1}{2}\sum_{i=0}^{n}\left(H\left(M_{o\to i}(x_0)\right) - y_i\right)^T \mathbf{R}^{-1}\left(H\left(M_{o\to i}(x_0)\right) - y_i\right)$$

Model operator and its linearized counterpart: $M, \mathbf{M}, \mathbf{M^T}$.

Observation operator and its linearized counterpart: $H, \mathbf{H}, \mathbf{H^T}$

oops directory structure

oops/src/oops/	assimilation	DA classes (minimizer, cost functions, local volume solvers, etc)
	base	base classes and classes build up on interface classes (state ensemble, observer, etc)
	generic	implementations that can be shared by different models/obs (diagonal obs errors, BUMP background error covariances)
	interface	interface classes (building blocks from previous slides, that need to be implemented)
	mpi	files relevant to mpi communications
	runs	applications (Variational, HofX, etc)
	util	utilities (datetime, timers, etc)

oops directory structure

oops/src/oops/	assimilation	DA classes (minimizer, cost functions, local volume solvers, etc)
	base	base classes and classes that build up on interface classes (state ensemble, observer, etc)
	generic	implementations that can be shared by different models/obs (diagonal obs errors, pseudo model)
	interface	interface classes (building blocks from previous slides, that need to be implemented)
	mpi	files relevant to mpi communications
	runs	applications (HofX, Variational, etc)
	util	utilities (datetime, timers, etc)

oops directory structure

oops/src/oops/	assimilation	DA classes (minimizer, cost functions, local volume solvers, etc)
	base	base classes and classes build up on interface classes (state ensemble, observer, etc)
	generic	implementations that can be shared by different models/obs (diagonal obs errors, BUMP background error covariances)
	interface	interface classes (building blocks from previous slides, that need to be implemented)
	parallel	files relevant to mpi communications
	runs	applications (HofX, Variational, etc)
	util	utilities (datetime, timers, etc)

H(x) application

HofX application (*oops/src/oops/runs/HofX4D.h*):

- Runs model forecast (possibly with pseudo-model, reading states from files)
- Computes H(x) for specified observations

The HofX application can also be used to generate observations for OSSE.

H(x) application

HofX application (*oops/src/oops/runs/HofX4D.h*): ~50 lines of code:

```
template <typename MODEL, typename OBS> class HofX4D {
//  Setup observation window
//  Setup Geometry
//  Setup Model
//  Setup initial State
//  Setup forecast outputs
//  Setup observations and observer

//  Run observer (includes “running model”, and computing H(x) and QC filters)
//  Perturb H(x) if needed (e.g. for OSSE)
//  Save H(x) either as observations (if "make obs" == true) or as "hofx”
}
```

Applications using different models


```
oops::HofX4D<fv3jedi::Traits, ufo::ObsTraits>
```

(in *fv3-jedi/src/mains/fv3jediHofX.cc*)

```
struct fv3jedi::Traits {
  typedef fv3jedi::State        State;
  typedef fv3jedi::Increment    Increment;
...
}
```

(in *fv3-jedi/src/fv3jedi/Utilities/Traits.h*)

```
oops::HofX4D<mpas::MPASTraits, ufo::ObsTraits>
```

```
struct mpas::MPASTraits {
  typedef mpas::StateMPAS        State;
  typedef mpas::IncrementMPAS    Increment;
...
}
```

```
struct ufo::ObsTraits {
  typedef ioda::ObsSpace        ObsSpace;
  typedef ufo::ObsOperator      ObsOperator;
...
}
```

(in *ufo/src/ufo/ObsTraits.h*)

H(x) application YAML

```
//  Setup observation window

//  Setup Geometry

//  Setup Model

//  Setup initial State


//  Setup observations and observer
```

```yaml
window begin: 2020-10-01T03:00:00Z
window length: PT6H

geometry:
  npx: 13
  npy: 13
  ...
model:
  name: ...
  ...
initial condition:
  filename_bkgd: ...
  ...
forecast length: PT6H

observations:
- obs space:
    name: Aircraft
    simulated variables: [air_temperature]
    ...
  obs operator:
    name: VertInterp
```

Questions?

Generic Algorithms

Abstract Interfaces