Observation operators in JEDI

Unified Forward Operator (UFO)

- The idea is to have observation operators as independent from the models as possible, so the UFOs can be easily shared
- As a result, the part of the observation operator that is grid/model-specific has to be implemented outside of UFO (simple example is horizontal interpolation)

- If the “full” observation operator H_{full} (that takes full state on input) can be written as

$$H_{full}(x_{full}) = H\left(Int(x_{full})\right) = H(x_{loc})$$

where Int is horizontal interpolation (to obs lat-lon) operator,

then UFO ObsOperator is the H part, x_{full} is the full State, and x_{loc} is the interpolated to observation location state (called GeoVaLs in UFO)

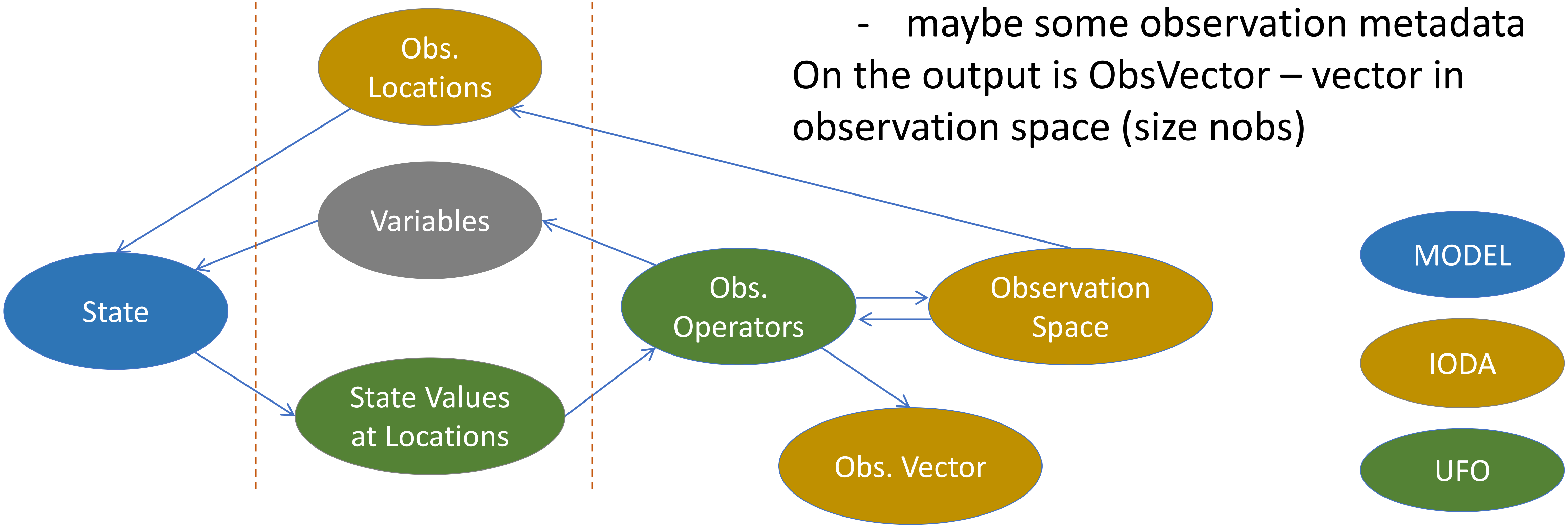
UFO ObsOperator

ObsOperator to compute $H(x)$:

Needs to know:

- a list of variables to get from the model state
- some observation space information:
  - observations locations
  - maybe some observation metadata

On the output is ObsVector – vector in observation space (size nobs)



Example of C++ level of ObsOperator in UFO

https://github.com/JCSDA/ufo-training/blob/develop/src/ufo/atmosphere/radiosonde/ObsRadiosonde.h

Tangent-linear and adjoint observation operator

- Separate C++ class for TL and AD observation operator
- Three important methods:
  - Set trajectory: calculate the Jacobian $\mathbf{H} = \left.\frac{\partial H}{\partial x}\right|_{x=x_0}$. Input: GeoVaLs x_0. $\mathbf{H}$ is then saved internally for future use in the TL and/or AD.
  - Calculate tangent-linear $\mathbf{H}dx.$ Input: GeoVaLs dx, output: ObsVector $\mathbf{H}dx$
  - Calculate adjoint $\mathbf{H}^{\boldsymbol{T}}dy$. Input: ObsVector dy, output: GeoVals $\mathbf{H}^{\boldsymbol{T}}dy$
- Note: to call TL or AD, first have to call the method that calculates $\mathbf{H}$

Example of C++ level of ObsOperatorTLAD in UFO

https://github.com/JCSDA/ufo-training/blob/develop/src/ufo/atmosphere/radiosonde/ObsRadiosondeTLAD.h

GeoVaLs: state interpolated to obs locations

- C++ level:

https://github.com/JCSDA/ufo-training/blob/develop/src/ufo/GeoVaLs.h and

https://github.com/JCSDA/ufo-training/blob/develop/src/ufo/GeoVaLs.cc

- Fortran level:
- https://github.com/JCSDA/ufo-training/blob/develop/src/ufo/GeoVaLs.interface.F90 , mostly getting objects from the keys and passing to the routines in
- https://github.com/JCSDA/ufo-training/blob/develop/src/ufo/ufo_geovals_mod.F90

GeoVaLs data structure (Fortran)

```
type :: ufo_geovals
  integer :: nobs         !< number of observations
  integer :: nvar         !< number of variables (supposed to be
                          !  The same for same obs operator

  type(ufo_geoval), allocatable :: geovals(:)  !< array of interpolated
                                               !  vertical profiles
                                               !  for all obs (nvar)

  type(ufo_vars) :: variables    !< variables list

  logical :: lalloc              !< .true. if type was initialized and
                                 !  allocated (only geovals are allocated,
                                 !  not the arrays inside of the ufo_geoval
  logical :: linit               !< .true. if all the ufo_geoval arrays
                                 !  inside geovals were allocated and have
                                 !  data
end type ufo_geovals
```

GeoVaLs data structure (Fortran)

```
type :: ufo_geovals

  integer :: nobs
  integer :: nvar


  type(ufo_geoval), allocatable :: geovals(:)


  type(ufo_vars) :: variables


  logical :: lalloc
  logical :: linit

end type ufo_geovals
```

One element of this array size(nvar) is for one model variable (e.g., temperature vertical profile, humidity vertical profile, SST, surface wind, etc)

Variables names

Single “geoval” (one variable) structure

```
type :: ufo_geoval

  real(kind_real), allocatable :: vals(:,:)

  integer :: nval

  integer :: nobs

end type ufo_geoval
```

GeoVaLs for a specific variable, size(nval, nobs)

Number of values in one ”profile”, can vary depending on application:
= number of model levels for variables like atmospheric temperature
= 1 for surface variables like SST, surface wind, etc
= number of ice categories for variables like sea ice concentration in sea ice model

Useful function: ufo_geovals_get_var, returns a pointer to ufo_geoval for a given variable