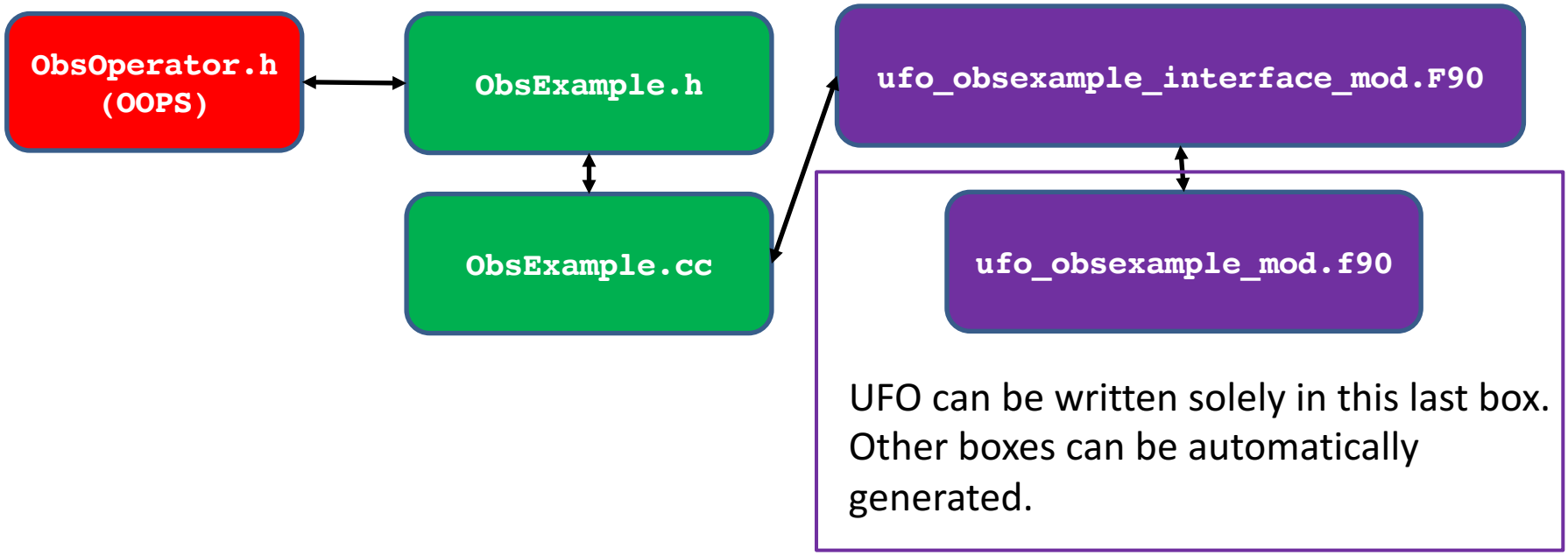
File structure for ObsOperator

ObsOperator (and ObsOperatorTLAD) classes follow the same file structure:



ObsOperator Interface in OOPS

```
template <typename MODEL>
class ObsOperator {

public:
  ObsOperator(const eckit::Configuration &, const util::DateTime &,
              const util::DateTime &);
  ~ObsOperator();

/// Obs Operator
  void simulateObs(const GeoVaLs_ &, ObsVector_ &, const ObsAuxControl_ &) const;

/// Getting model variables that are needed for ObsOperator
  const Variables & variables() const;  // Required input variables from Model
/// Getting observation “variables” and “locations”
  const Variables & observed() const;   // Observed variables produced by H
  Locations_ locations(const util::DateTime &, const util::DateTime &) const;

}
```

Example C++ implementation of ObsOperator class in UFO

Class implementing the OOPS interface has to implement the methods from previous slide.

Inside those methods, Fortran routines can be called, for example:

```
void ObsExample::simulateObs(const GeoVaLs & gv, ioda::ObsVector & ovec,
                             const ObsBias & bias) const {
  ufo_example_simobs_f90(keyOper_, gv.toFortran(), odb_, ovec.size(),
                         ovec.toFortran(), bias.toFortran());
  oops::Log::trace() << "ObsExample: observation operator run" << std::endl;
}
```

Calling Fortran routine for observation operator

Example Fortran implementation of ObsOperator in UFO


```
! ------------------------------------------------------------------------------
! TODO: put code for your nonlinear observation operator in this routine
! Code in this routine is for example only, please remove and replace
subroutine ufo_example_simobs(self, geovals, hofx, obss)
implicit none
class(ufo_example), intent(in)    :: self
type(ufo_geovals),  intent(in)    :: geovals
real(c_double),     intent(inout) :: hofx(:)
type(c_ptr), value, intent(in)    :: obss

! Local variables
type(ufo_geoval), pointer :: geoval
integer :: ierr, nlocs
real(kind_real), allocatable :: obss_metadata(:)

! check if some variable is in geovals and get it (var_tv is defined in ufo_vars_mod)
call ufo_geovals_get_var(geovals, var_tv, geoval, status=ierr)

! get some metadata from obsspace
nlocs = obsspace_get_nlocs(obss)
allocate(obss_metadata(nlocs))
call obsspace_get_db(obss, "MetaData", "some_variable", obss_metadata)

! put observation operator code here


end subroutine ufo_example_simobs
```

Generating C++ and Fortran files for ObsOperator and ObsOperatorTLAD

- There is “example” C++ and Fortran code for ObsOperator and ObsOperatorTLAD in ufo/tools/new_obsop/example. The code just provides interfaces, but doesn’t do anything. We will keep this code up-to-date with OOPS interfaces.
- Script create_obsop_fromexample.sh in ufo/tools/new_obsop copies example, renames files to the ObsOperator name of choice, and replaces “example” with this name.
- After that you can start working on ufo_obsopname_mod.F90 to implement observation operator and ufo_obsopname_tlad_mod.F90 to implement TL/AD