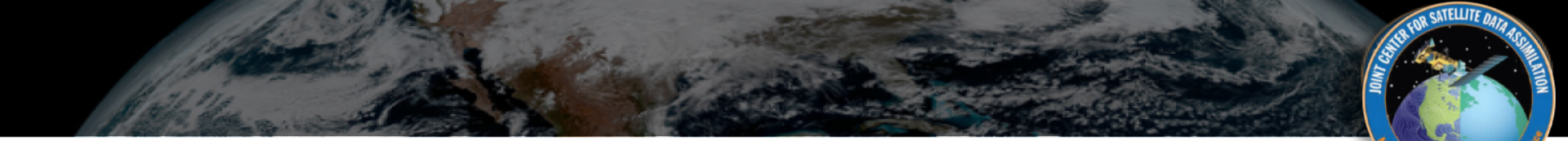
The Joint Effort for Data assimilation Integration (JEDI)

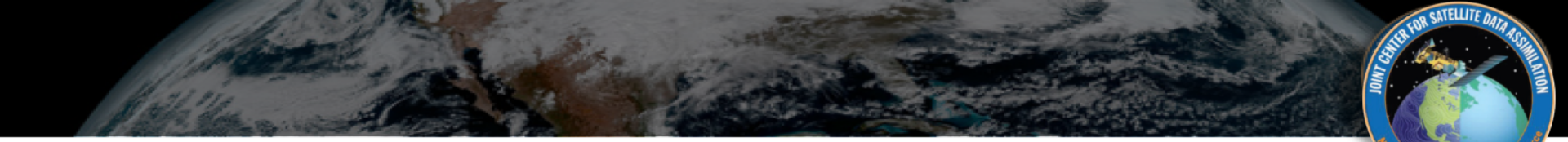
From Objectives to Code Design

Joint Center for Satellite Data Assimilation (JCSDA)

JEDI Academy – 24-27 February 2020



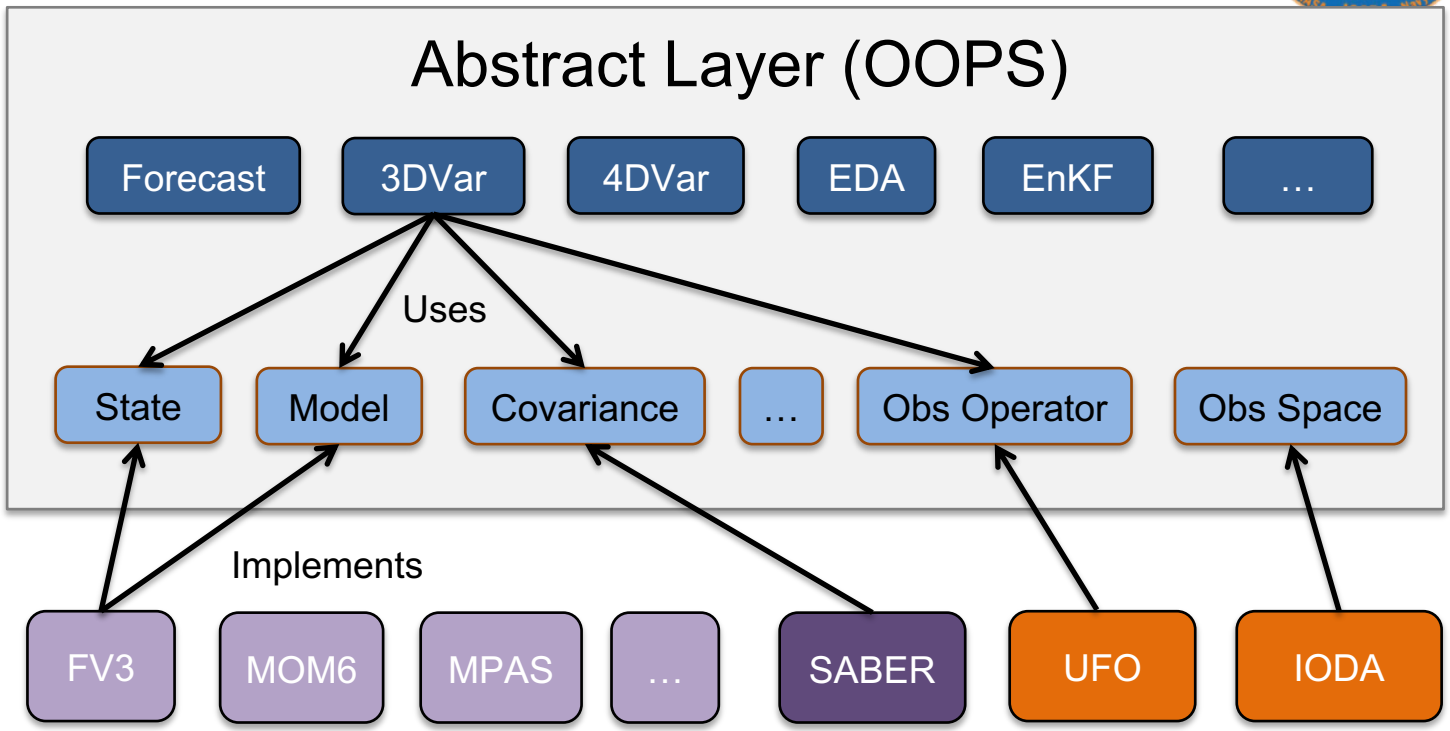
Continued:
From Abstract/Generic to Concrete

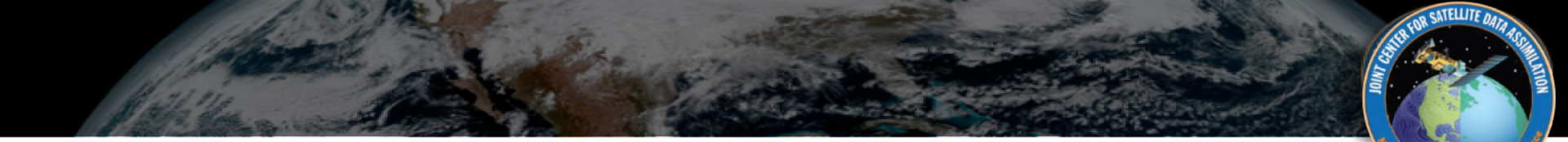


Generic applications

Interface layer

Specific implementations

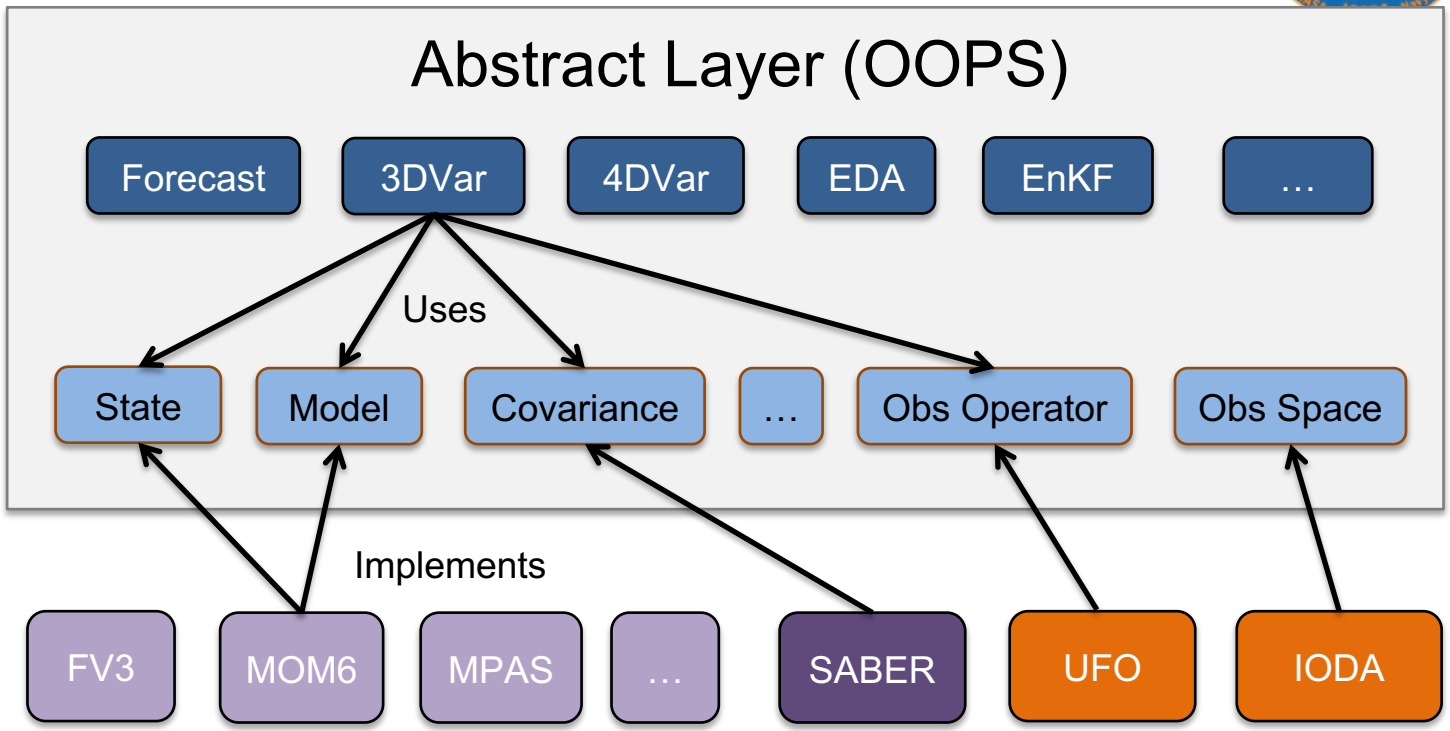




Generic applications

Interface layer

Specific implementations



oops directory structure

oops/src/oops/

assimilation	DA classes (minimizer, cost functions, 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, need to be implemented)
parallel	files relevant to mpi communications
runs	applications (Variational, HofX, EDA, etc)
util	utilities (datetime, timers, etc)

Variational application

Variational application (oops/src/oops/runs/Variational.h):

- creates cost function
- runs IncrementalAssimilation (computes cost function, runs minimizer)
- type of minimizer and type of cost function (3D-Var, 4D-Var, 4D-Weak, 4D-Ens-Var) are controlled through yaml

EDA application

EDA (ensemble of data assimilations):

- runs several *Var analyses in parallel, with perturbed observations
- the application (oops/src/oops/runs/EDA.h) splits MPI communicator and runs application Variational with relevant communicator for this task.

JEDI: Main Programs

All JEDI/OOPS applications have one (sometimes two) arguments: a yaml configuration file

Standard C++ main

```
int main(int argc,  char ** argv) {
  oops::Run run(argc, argv);
  oops::Variational<lorenz95::L95Traits> var;
  run.execute(var);
  return 0;
}
```

Run object for technical setup: read yaml configuration, start MPI, start loggers…

Create Application object: This is where the model is determined

Execute Application

Variational application (abridged)


```
template <typename MODEL> class Variational : public Application {
 public:
  Variational() {
    instantiateCostFactory<MODEL>(); //  and other factories (min, obserr, filter, etc)
  }
  int execute(const eckit::Configuration & fullConfig) const {
//  Setup cost function
    const eckit::LocalConfiguration cfConf(fullConfig, "cost_function");
    std::unique_ptr< CostFunction<MODEL> > J(CostFactory<MODEL>::create(...);
//  Initialize first guess from background
    ControlVariable<MODEL> xx(J->jb().getBackground());
//  Perform Incremental Variational Assimilation
    IncrementalAssimilation<MODEL>(xx, *J, fullConfig);
    return 0;
  }
};
```

Applications using different traits


```
int main(int argc,  char ** argv) {
  oops::Run run(argc, argv);
  oops::Variational<fv3jedi::Traits> var;
  run.execute(var);
  return 0;
}
```

fv3-jedi/src/mains/fv3jediVar.cc

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

fv3-jedi/src/fv3jedi/Utilities/Traits.h

```
int main(int argc,  char ** argv) {
  oops::Run run(argc, argv);
  oops::Variational<lorenz95::L95Traits> var;
  run.execute(var);
  return 0;
}
```

oops/l95/src/executables/Main4Dvar.cc

```
struct lorenz95::L95Traits {
...
  typedef lorenz95::StateL95        State;
  typedef lorenz95::IncrementL95    Increment;
  typedef lorenz95::ObservationL95  ObsOperator;
...
}
```

oops/l95/src/lorenz95/L95Traits.h

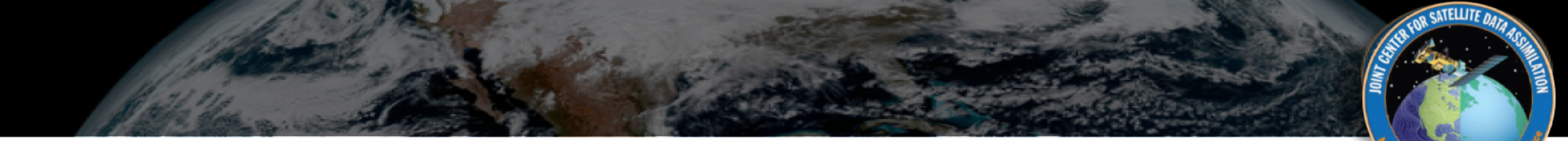
Some classes may be shared in Traits


```
int main(int argc,  char ** argv) {
  oops::Run run(argc, argv);
  oops::Variational<fv3jedi::Traits> var;
  run.execute(var);
  return 0;
}
```

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

```
int main(int argc,  char ** argv) {
  oops::Run run(argc, argv);
  oops::Variational<soca::Traits> var;
  run.execute(var);
  return 0;
}
```

```
struct soca::Traits {
...
  typedef soca::State         State;
  typedef soca::Increment     Increment;
  typedef ufo::ObsOperator    ObsOperator;
...
}
```



yaml for Variational application


```
resolution:
model:
cost_function:            # entries describing cost functions
  cost_type:              # valid types: 3D-Var, 4D-Var, 4D-Ens-Var, 4D-Weak
  window_begin:
  window_length:
  Jb:                     # entries describing Jb term of cost function
    Background:
    Covariance:
  Jo:                     # entries describing Jo term of cost function
    ObsTypes:
```

yaml for Variational application


```
variational:
  iteration:          # size of array = number of outer loop iterations
  - ninner: 10.       # number of inner loop iterations for 1st iteration
    gradient_norm_reduction: 1e-10
    resolution:
    linearmodel:
  - ninner: 10        # number of inner loop iterations for 2nd iteration
    gradient_norm_reduction: 1e-10
    resolution:
    linearmodel:
minimizer:
  algorithm: DRIPCG   # valid options: PCG, RPCG, DRPCG, IPCG, DRIPCG,
                        Planczos, Rlanczos, DRPLanczos, MINRES, FGMRES,
                        GMRESR, DRGMRESR, LBGMRESR, DRPFOM
```