Overview of the Data Assimilation Research Testbed

Draft: 25 April, 2002

Goals:

Data assimilation is the term used in atmospheric and oceanic sciences for the process of merging observations with a model. Data assimilation makes observations more useful by converting diverse and heterogeneous observations to regularly spaced and uniform quantities that can be interpreted more easily. At the same time, observational error can be reduced and information about model errors can be generated. In more advanced applications of data assimilation, models can be improved by confronting them with data. Data assimilation can also be used to perform Observing System Simulation Experiments (OSSE's) which evaluate the impact of existing or proposed observations for particular applications.

The data assimilation problem requires the coordination of expertise in many diverse areas. Model developers, observational specialists, and statisticians trained to do the 'filtering' that is at the core of assimilation algorithms, must all combine expertise to do this problem in an efficient fashion. However, software engineering and organizational practices have made it extremely difficult for experts in these different areas to interact. The result has been that data assimilation development efforts have generally been linked to a single model or observational set) Different assimilation methodologies have not been readily comparable because applying them to a different model has involved too much effort. At the same time, model developers and observationalists have often been tied to a single assimilation methodology without the ability to evaluate the abilities of other methodologies. The result is an inefficient use of resources where every research group is forced to have significant in-house expertise in all three aspects of the problem and is unable to interact well with external groups.

Both the software and organizational barriers to improving this situation can be overcome. Given ongoing community activities to build coordinated software frameworks, it is now possible to design a test-bed facility that would allow the naive combination of a numerical model, a set of observations, and a data assimilation methodology to produce assimilations. The Data Assimilation Research Testbed (DART) is a prototype for an assimilation testbed facility with two fundamental purposes: allowing assimilation algorithm developers to compare their methodologies in a fair way; allowing model developers and observationalists to explore the efficacy of various assimilation algorithms for their problems of interest. DART is designed to demonstrate that a mature test-bed could greatly increase the rate of development of improved data assimilation methodologies, in turn leading to improved datasets, better predictions, and a more efficient design of observational systems.

Implementation:

DART is a software facility that allows a hierarchy of different classes of users to experiment with data assimilation algorithms. At the core of DART is a software infrastructure which is designed to implement standardized interfaces to a wide array of models and observational sets. DART also includes an assortment of data assimilation algorithms built on top of this infrastructure, and a

wide assortment of models, ranging from highly idealized dynamical systems with a handful of variables to General Circulation Models (GCMs) which may be configured with more than a million state variables. A variety of both simulated and actual observation sets are also part of DART. Actual observations are obtained from a variety of sources and converted to a format consistent with the DART software infrastructure. The generation of simulated observation sets (a key component of OSSEs) using models and specified error characteristics for simulated instruments is one of the central capabilities of DART. DART also maintains a variety of auxiliary tools and user interfaces that allow users to experiment, combining assimilation methodologies, models, and observations in ways that can shed new light on their problems of interest.

User views:

An overview of the capabilities of DART can be given by discussing a hierarchy of 'user views' of the facility.

View 1: Analysis of previously executed data assimilation experiments:

DART provides an assortment of analysis tools that can give insight into the performance of a particular data assimilation experiment. These tools accept data from three different types of files which provide information about quantities related to observations, model state variables, and the behavior of the assimilation algorithm itself.

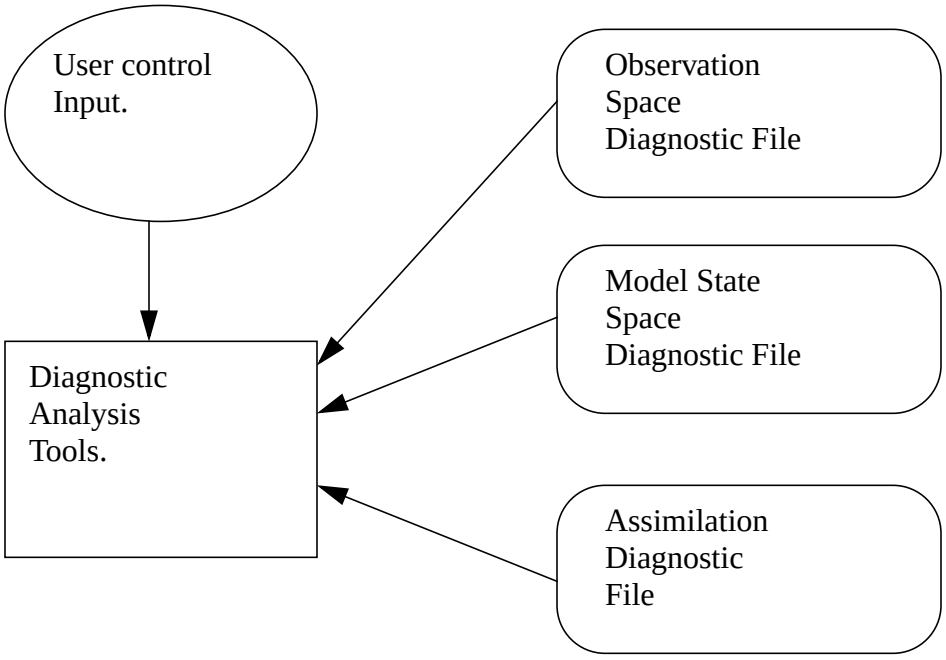
Observation space diagnostic files can provide information about the observing system itself such as the values of observations assimilated, meta-data defining the location and kind of the observations, error covariances of the observations, and, in synthetic observation assimilation experiments, the 'true' value of the observed quantity (what would have been measured with a hypothetical instrument with no error). In addition, information from the assimilation algorithm related to observations is also available in these files. These can include information about the (prior and posterior) estimates of the observations generated during an assimilation experiment. Since these (prior and posterior) estimates are formally probability distributions, information could include an estimate of the mean, estimates of higher order moments, a set of samples from the distribution, or a variety of other information. However, all data in observation space diagnostics files must have associated meta-data that describes the observations themselves. Diagnostic tools are available to produce such things as plots of error as a function of time for a repeated observation, overall assessments of error from observations as a function of time or space, etc.

State space diagnostics files provide information about the model state variables (or extended state variables that are functions of the state variables). These files contain meta-data describing the location and kind of model state variables and associated data. For many models and assimilation experiments, state space diagnostic data can be (quasi-)regularly distributed in space and time, but this is not a requirement. Often, it is natural to provide state space output of the (prior and posterior) model state estimate at the times for which observations are available. For instance, one might have the value of the (prior and posterior) model state estimates, some estimates of the error associated with these estimates, the 'true' value in synthetic observation experiments, etc. Diagnostic tools are available to produce cross section plots of the space-time state estimate. In a GCM experiment these could include time sections of state estimates at a particular point, plots of

the (error of the) state estimate on a particular vertical or horizontal surface, time series of globally integrated error of a particular field, etc.

Assimilation diagnostic files provide information about the behavior of the assimilation algorithm itself...

View 1 of DART: Analysis of previously existing assimilation experiments



When available should have some examples of diagnostic output, initially from one-dimensional models, here.

View 2: Exploration of Assimilation Experiments

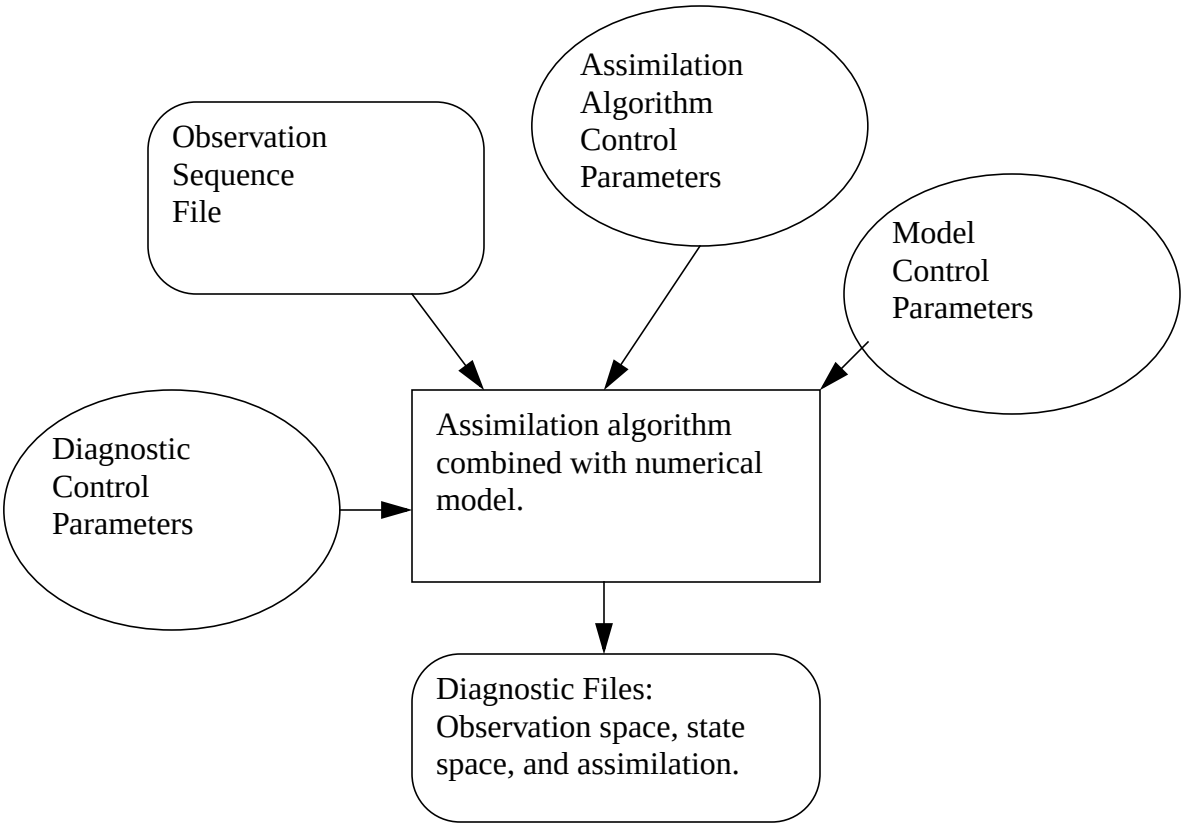
DART allows users to run a previously configured assimilation experiment, a program combining an assimilation algorithm and a numerical model, with modified parameters or different sets of previously defined observations. DART has been designed to make many options of both numerical models and assimilation algorithms available for run-time modification. Users can modify details of the numerical model (for instance the value of a diffusion coefficient in a GCM), details of the assimilation algorithm (for instance, the number of ensemble members used in an ensemble

Kalman filter), or details of the output diagnostics (for instance, selecting some specific subset of model state variables for more intensive diagnostic output).

DART assimilation experiments ingest an observation sequence file which provides meta-data defining the available observations as well as the observations themselves. A variety of pre-defined observation sequence files are available in DART to allow users to explore the impact of different observing systems on an assimilation experiment. The observation sequence files are in the same format as observation space diagnostic files and can be analyzed using the diagnostic tools described in View 1 above.

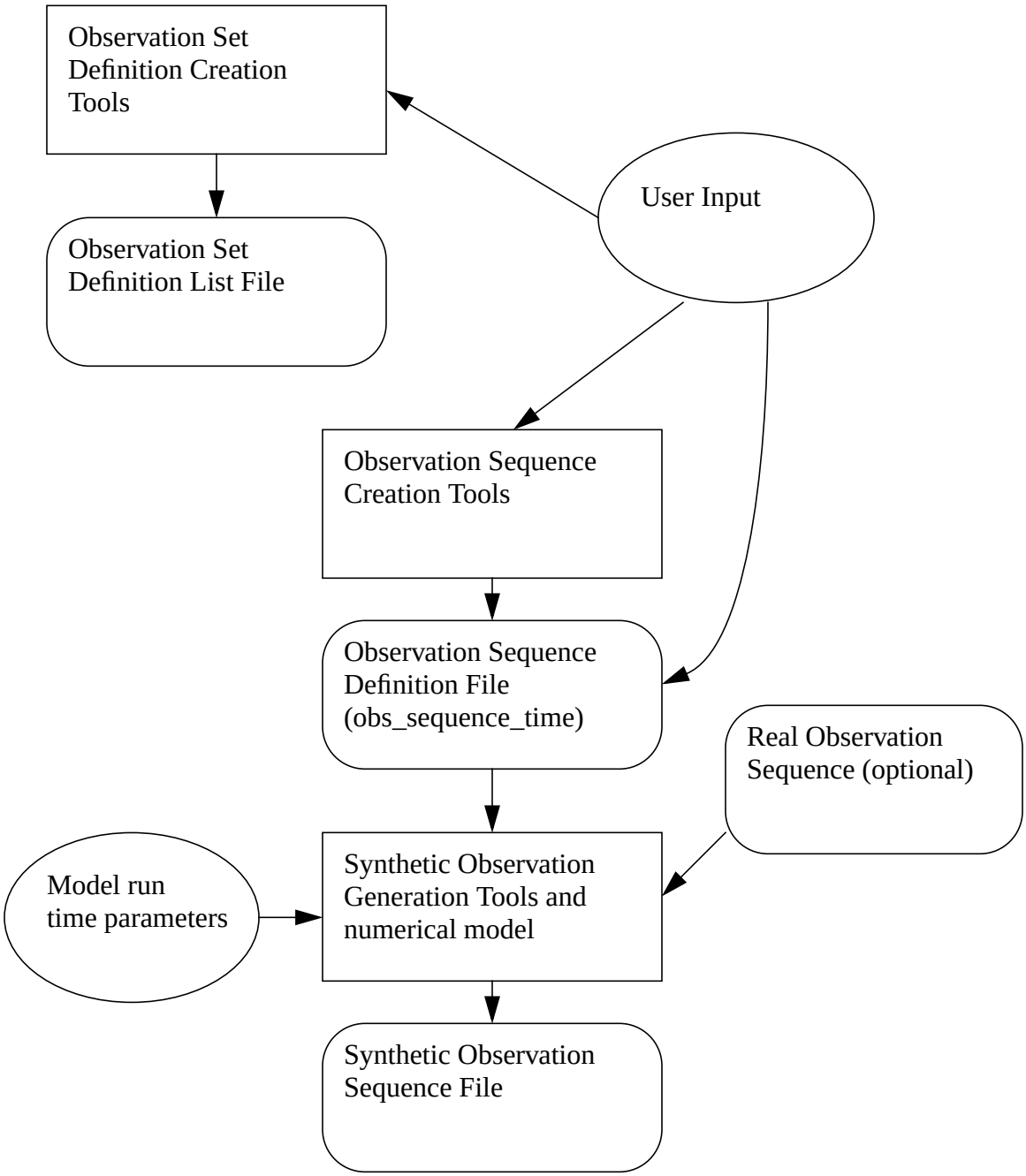
The programs that execute the assimilation experiments are currently implemented as Fortran-90 programs. Users can obtain access to particular configurations by either running previously created executables on DART computing facilities, or by obtaining source code along with configuration management software that allows them to create executables on their own platform.

View 2: Assimilation Experiment Exploration



View 3: Synthetic observation / Observation System Simulation Experiments

Many interesting applications of data assimilation involve the use of synthetic observations, where a model and a specified observing system (including both observation locations and times as well as error characteristics) are used to generate observations that can then be assimilated by the model. The same software that is used to perform assimilations can be used to generate synthetic observations.



DART allows users to specify sets of related observation definitions (a definition here refers to all information about an observation except its time and the actual value of the observation). These observation definitions can then be combined with information about time to generate sequences of observation definitions. A sequence of observation definitions can be in concert with a numerical model to generate synthetic observations using the model integration as input. The results is an obs_sequence with synthetic values which can be used as input to View 2 experiments. Synthetic observations can also be combined with real observations to explore the potential impact of enhancing an existing observing system.

View 4: Adding new models

DART provides a great deal of software infrastructure to assist in modifying existing numerical models for use with data assimilation methods. Nevertheless, adding a model to DART will require Fortran coding by someone who is intimately familiar with the details of the numerical model. Once these interfaces are added, the model should be able to be tested with a variety of assimilation methodologies and to make use of existing observation sequences.

For a model to be compliant with DART, it must provide a set of basic capabilities.
1. The complete model state (in a formal sense) must be accessible in the form of a single long vector. A set of model state metadata associated with this long state vector must also be accessible in a standard format.

2. An ability to advance the model state in time must be provided. This facility can be either ‘synchronous’, meaning that a Fortran90 callable subroutine to advance the model given an input state vector is provided, or ‘asynchronous’, meaning that a ‘script level’ mechanism to advance the model given a file containing the long state vector is provided.

3. An ability to interpolate the model state variables to a given physical location within the model domain, for instance a given latitude, longitude, and height.

A number of additional capabilities may be required for efficient operation in certain assimilation systems and are outlined below.

Detailed requirement for a DART compliant model. At the highest level, the model is accessed through a Fortran90 module named model_mod. The following public interfaces must be provided by the model. Where feasible, any interface may be replaced with a Fortran90 stub that performs no operations.

1. function get_model_size()
    Returns the length of the model state vector as an integer.

2. subroutine adv_1step(x, Time)

Advances a model for a single time step if this operation is defined. The time associated with the initial model state is also required. This interface is only required if ‘synchronous’ model state advance is supported (the model is called directly as a Fortran90 subroutine from the assimilation programs). This is generally not the preferred method for large models which can provide a stub for this interface.

real, intent(inout) :: x(:)	State vector of length model_size.
type(time_type), intent(in) :: Time	Gives time of the initial model state. Needed for models that have real time state requirements, for instance the computation of radiational parameters. Note that DART provides a time_manager_mod module that is used to support time computations throughout the facility.

3. subroutine get_state_meta_data(index_in, location, var_type)

Returns metadata about a given element, indexed by index_in, in the model state vector. The location defines where the state variable is located (at present, a variety of simple location models for support of gridpoint models are provided) while the type of the variable (for instance temperature, or u wind component) is returned by var_type. The integer values used to indicate different variable types in var_type are themselves defined as public interfaces to model_mod if required. Low order models in which var_type is not necessarily meaningful should simply return a single integer value.

integer, intent(in) :: index_in	Index into the long state vector.
type(location_type), intent(out) :: location	Returns location of indexed state variable. The location should use a location_mod that is appropriate for the model domain. For realistic atmospheric models, for instance, a three-dimensional spherical location module that can represent height in a variety of ways is provided.
integer, intent(out), optional :: var_type	Returns the type of the indexed state variable as an optional argument.

4. subroutine model_interpolate(x, location, type)

Given model state, returns the value of variable type interpolated to a given location by a method of the model’s choosing. At present, this is the only support for forward operators that is required from the model_mod. As observations with more complex forward operators are explored a significant additional complexity may be required for forward operator interfaces.

real, intent(in) :: x(:)	Model state vector.
type(location_type), intent(in) :: location	Location to which to interpolate
integer, intent(in) :: type	Integer indexing which type of state variable is to be interpolated. Can be ignored for low order models with a single type of variable.

5. function get_model_time_step()

Returns the models base time step as a time_type. In the long run, a more general extended interface may be required that specifies the models range of time stepping possibilities.

6. subroutine end_model

Called when use of a model is completed to clean up storage, etc. Can be a stub for most applications.

7. subroutine static_init_model()

Used for runtime initialization of a model, for instance calculating storage requirements, intializing model parameters, etc. This is the first call made to a model by any DART compliant assimilation routine.

8. subroutine init_time(i_time)

Returns the time at which the model will start if no input initial conditions are to be used. This is frequently used to spin-up models from rest, but is often not meaningfully supported in comprehensive GCMs.

type(time_type), intent(out) :: i_time	Model's initial time.

9. subroutine init_conditions(x)

Returns default initial conditions for model; generally used for spinning up initial model states. For GCMs can conceivably just return 0's if initial state is always to be provided from input files.

real, intent(inout) :: x(:)	Model state vector

10. subroutine model_get_close_states(o_loc, radius, number, indices, dist)

Returns the number of state variables that are within a given radius (the units for the radius depend upon the location_mod module being used by the model) of an observation at location o_loc. The indices in the long state vector as well as the distance between each close state variable and the observation are also returned, provided there is sufficient storage available for them in the arrays indices and dist. This tends to be the most complicated routine for large models because it must be implemented in a very efficient fashion to support a number of assimilation algorithms.

type(location_type), intent(in) :: o_loc	Location of observation
real(r8), intent(in) :: radius	Maximum distance between state and observation
integer, intent(out) :: number	Number of close state variables
integer, intent(out) :: indices(:)	Indices of close state variables found in long model state vector. If allocated size of this array is too small, only a subset of the close indices is returned. The model_mod gets to decide what subset this is.

real(r8), intent(out) :: dist	Distance between observation and state variables indexed in indices array.

11. function nc_write_model_atts(ncFileID) result(ierr)

    Function to write model specific attributes to a netCDF file. At present, DART is using the NetCDF format to output diagnostic information. This is not a requirement, and models could choose to provide output in other formats. This function writes the metadata associated with the model to a NetCDF file opened to a file identified by ncFileID.

integer, intent(in) :: ndFileID	Integer file descriptor opened to NetCDF file
integer :: ierr	Returned error code.

12. function nc_write_model_vars(ncFileID, statevec, copyindex, timeindex) result(ierr)

    Writes a copy of the state variables to a NetCDF file. Multiple copies of the state for a given time are supported, allowing, for instance, a single file to include multiple ensemble estimates of the state.

integer, intent(in) :: ncFileID	Integer file descriptor opened to NetCDF file
real(r8), intent(in) :: statevec(:)	State vector
integer, intent(in) :: copyindex	Integer index to which copy is to be written
integer, intent(in) :: timeindex	Integer index of which time in the file is being written
integer :: ierr	Returned error code.

Details on advancing models in time:
As noted above, one option for advancing models in time is to provide the interface adv_1step. In many cases, large models are very difficult to cast in terms of a Fortran90 callable subroutine. DART compliant assimilation modules are required to be able to advance models either through the adv_1step interface, or by allowing the operating system (script) to advance the model state. This second choice has been used predominantly when incorporating GCMs into DART. In this case, referred to as ‘asynchronous’ model advance, DART compliant assimilation routines create a file containing the long state vector and the corresponding state time (routines for writing out this format are provided through the DART assim_model_mod module). The assimilation executable then waits for while external program(s) read the file, advance the state in time to the requested time, and write a file with the updated state. The assimilation module then reads this updated state and proceeds. Examples of this implementation can be seen in DART implementations of the GFDL B-grid dynamical core and the CAM2.0 AGCM.

View 5: Adding new observation sets

The tools used to generate synthetic observations are readily adapted to the introduction of new observational data sets into DART. Real observation sets are notoriously difficult to work with, so this process is likely to require effort by someone with expertise in a particular data set. Once the

set is available in the format of a DART observational sequence, it can be used with a variety of assimilation techniques and models.

View 6: Adding new assimilation techniques:

Although the DART infrastructure is designed to ease this process, it is likely that, at least during the early stages of the project, adding new assimilation algorithms may lead to requirements on the observation or model portions of DART that are unanticipated. This suggests that adding assimilation techniques may require some assistance from model experts in order to allow all models to be fully compliant. However, the promise of being able to make broad and fair comparisons of different assimilation algorithms is likely to encourage assimilation experts to try to place new algorithms in DART.