Prototype Forcing File Generator

Matthew Russell

February 12 2013

1 Proposed Architecture

Figure 1 shows a high level process diagram for how we propose forcing files can be generated. This process shows that forcing files would be generated in post processing. This choice includes several advantages and disadvantages. Namely, this model limits the process to only being able to use hour concentration output rather than concentrations at time steps only visible during the simulation. However, using this model, forcing term generation does not have to be incorporated into CMAQ and as such, the tools that can be used to implement the software are not limited to Fortran, CMAQ does not require re-compilation for every update, and the process can be implemented in a user-friendly graphical user interface using cross-platform compatible software.

Figure 2 shows that the application would first require domain parameters to be specified, then general inputs (shown in fig. 2 in orange), then inputs specific to the chosen adjoint cost function (shown in blue.)

2 Requirements

Here and after, the process to generate forcing files will be referred to as "the application".

The application must:

- be user-friendly
- be easy for users to incorporate their own custom forcing functions into
- be easily re-run for multiple simulations (i.e. settings can be saved and re-used from the command line)
- not be platform specific
- depend on software easily available

2.1 Specification

The application will be implemented in Python. This choice was made as Python is a popular language in the research community, is available on virtually all platforms, very well supported, can be incorporated with Fortran (not used here, but leaves open many possibilities), easy to learn and has many online educational resources, and can be used to create graphical user interfaces.

Required packages (generally all available via package managers): python-wxglade (for wxpython), python-numpy, python-netcdf, python-dateutil. Winder manager library is *wxpython*.

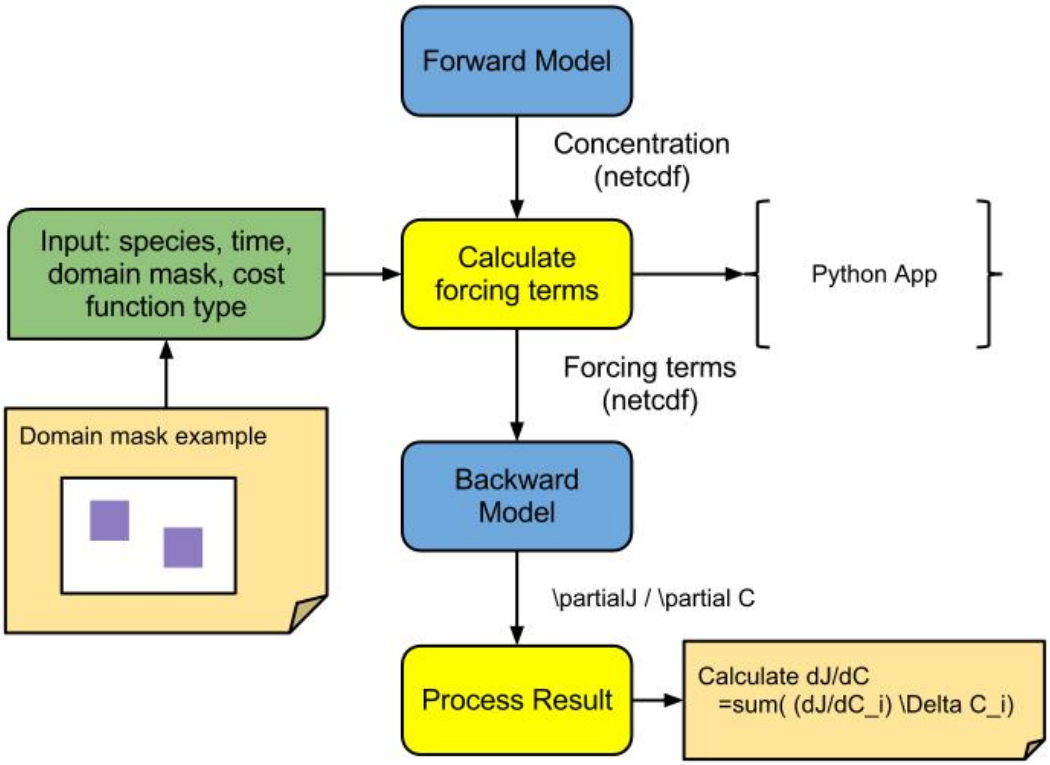

Figure 1: Flowchart depicting general process flow

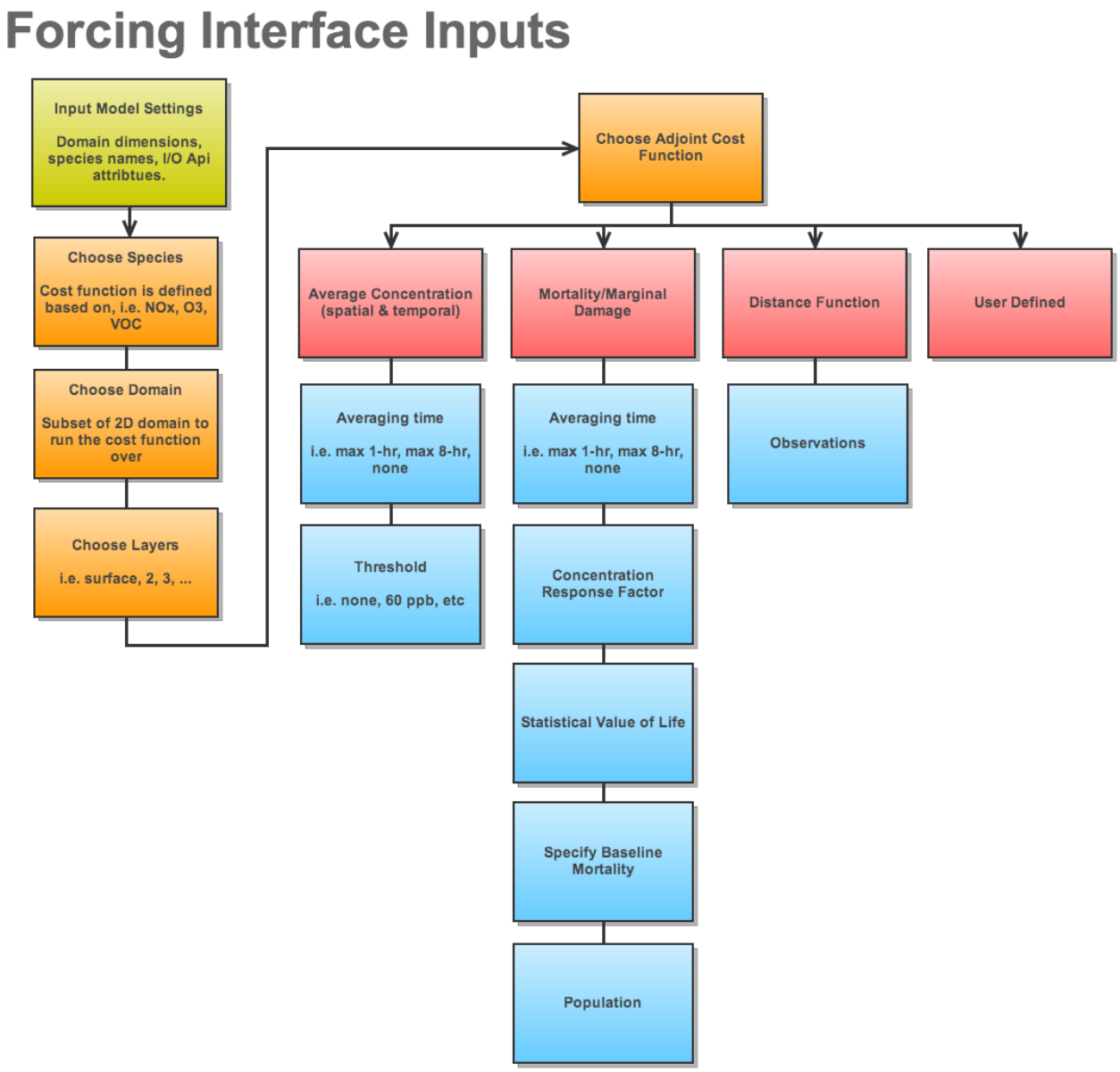

Figure 2: Digram illustration the different parameters required for each adjoint cost function.

- Object model is such that only two classes have to be added to add a custom cost-function
- The application will have a "save state" button that can be loaded another time to re-load the settings. This state can also be called from the command line such that this application can operate form the command line or from inside scripts.
- The application is broken up into four panels, one to read the simulation domain, one for general inputs that apply to all cost functions, one for the specific cost function, and one "logging" panel where the user receives messages about what they are setting and help messages

Processing the results of the adjoint model (the second yellow box in figure 1) will be done by a different mode in the same application. At this time however, proposed interface designs are not yet ready to be presented.

2.2 Proposed Application Design

Figures 3 and 4 illustrate a preliminary design for the application. These figures are here to show the ease of use and potential of a graphical user interface. Many features in the specifications are still missing in this proposal at this time.

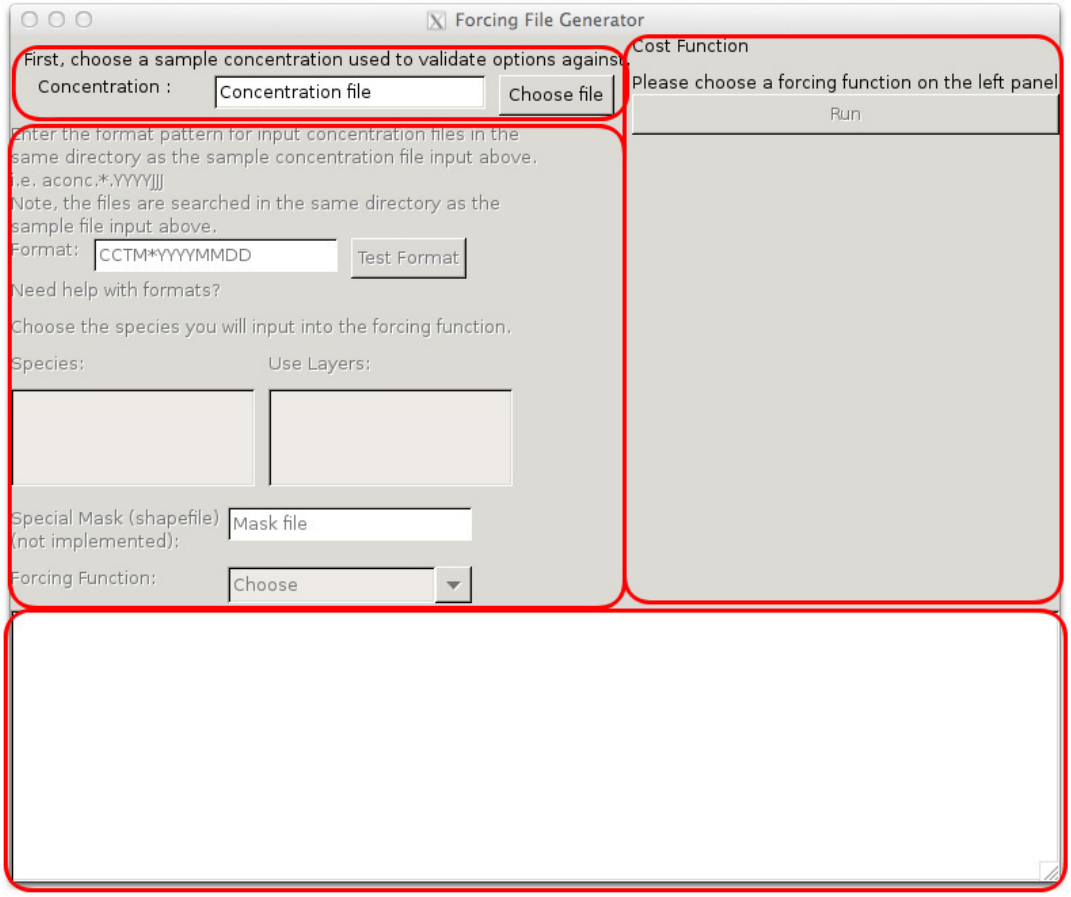

Figure 3: Example of the application before any inputs are filled in. Most inputs in this stage are disabled as the user has not yet defined the domain by providing a sample concentration file.

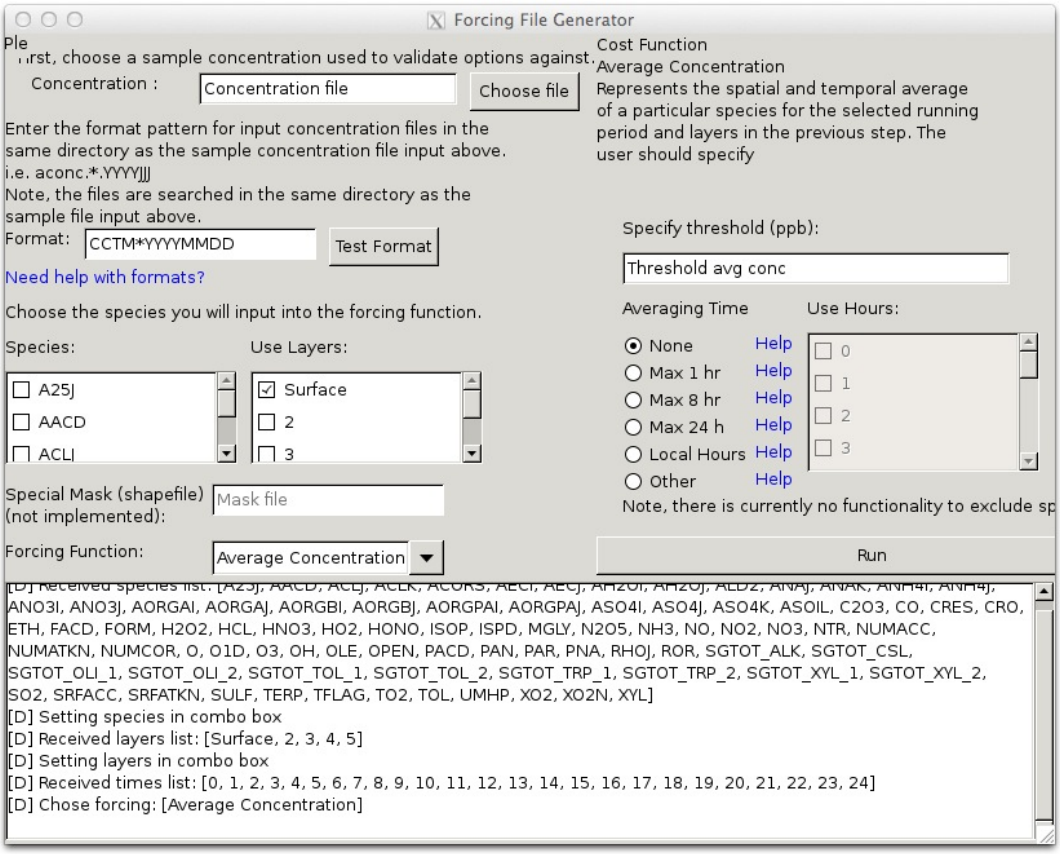

Figure 4: Example of the application when inputs are given and the Averaging Concentration adjoint cost function are chosen.