Why Robust Software Engineering Matters for Atmospheric Composition Retrievals

AGU Poster Narration Script

Motivation

Tropospheric sounding data from satellite observations provides critical information about atmospheric composition and its impact on human health and climate.

Our project, TRopospheric Ozone and Precursors from Earth System Sounding, TROPESS for short, will generate Earth System Data Records of ozone and other atmospheric constituents by processing data from multiple satellites through a common retrieval algorithm. The science team will contribute to international initiatives such as the Tropospheric Ozone Assessment Report.

The retrieval of atmospheric composition from remote sensing measurements is a complex process that requires the integration of cross cutting domain knowledge into a coherent software package. Where we are combining knowledge from such subjects as physics, information theory, mathematical optimization and computing.

This complexity increases when trying to create a software package that can handle data from multiple instrument, each operating in different spectral regions, each with their own peculiarities. Complexity is further compounded when trying to combine information from these instruments together into joint retrievals.

Yet, there are standard practices and algorithms that have been developed that are reusable. For instance the radiative transfer and retrieval techniques used by various missions are usually very similar. But the complexity asserts itself when connecting these well know techniques and algorithms together. The complexity arises in how these techniques are interfaced with each other. The fact that atmospheric composition software is continually rewritten is because it is often written for a specific mission without the foresight into its adaptability in the future. It is the interfaces, the book keeping and configuration aspects that most impact whether a software package can be reused or not.

Interfaces Manage Complexity

Managing this complexity requires robust software engineering practices. It requires a partnership between software developers and scientists.

The key software engineering aspect we believe helps manage the complexity of atmospheric composition retrieval software is the use of well thought out abstract interfaces.

It is a technique we have used in the architectural design of our ReFRACtor software. This framework will be applied to a combined suite of hyper-spectral thermal infrared, near-infrared, and ultraviolet instruments to generate Earth System Data Records (ESDRs) of Earth's tropospheric composition, including ozone, carbon monoxide, and water vapor deuterium. These ESDRs will have accuracies superior to composition measurements derived from any single instrument.

Abstract interfaces decouple behavior from implementation. They define an expected contract that implementations need to uphold. These interface contracts need to be designed to make the fewest assumptions on implementation as possible.

High level modules should not depend on low level modules. Instead they should depend on abstractions. Abstractions should not depend on details. Instead, details should depend on abstractions.

The diagram in the top right panel gives an example way of breaking up behavior into interfaces and implementation. The configuration layer knows there is some retrieval object that gets used. It uses this interface to, for instance, call the retrieval component's "solve" method to start the retrieval process without knowing the underlying details of how that happens. It then reports the results of the retrieval to an output product using other interface methods.

A specific implementation of a retrieval system might be based on the Levenberg-Marquardt solver. To perform its operations it would need a forward model to compare modeled radiances against measured radiances. The interface to a forward model object need only return some set of radiances on the same grid as the instrument itself. How this occurs is not the concern of the solver.

An example forward model might take a physics based approach and hence need to use a radiative transfer algorithm to perform its work. Or an alternative implementation might be to use a machine learning approach, or some statistical modeling method. With the right interfaces these different methods can be interchangeable as long as they meet the interface contract.

In this example the forward model knows to use some radiative transfer component. But it does not care what type of radiative transfer is used as long as it gets the desired monochromatic radiances. It will then modify these radiances to match the instrument using components such as a spectral response function and solar model if relevant. Each of these components would in turn have an interface specification as seen in the forward model box in the diagram in the lower left panel.

With the correct interfaces we can swap out the radiative transfer implementation. We can also model more complex behavior by utilizing multiple radiative transfers components together. We can set up the system to use, for instance, an optimized single scattering rT alongside an optimized multi-scattering model combined together for their speed advantages. The forward model does not care how it gets the radiances it requests, only that the contract is met.

Well-designed interfaces create a system that is “open for extension.” Meaning that behavior can be extended as requirements change. The system would also be “closed for modification.” Existing working and tested software should not have to change to meet requirements.

Representing components as connected through abstract interfaces allows flexibility that leads to reuse. You gain the ability to swap out different implementations without impacting other components. Older versions of an algorithm can exist alongside newer version making testing algorithmic upgrades much easier.

You gain the ability to reuse generic algorithms for different missions and instruments because those that are mission specific can be substituted elsewhere.

Well tested algorithms can continue to be relied upon because they do not change so long as the interface does not change.

ReFRACtor Software Framework

We have built the Reusable Framework for Atmospheric Composition, ReFRACtor for short, using these principles. It is an extensible multi-instrument atmospheric composition retrieval framework that supports and facilitates data fusion of radiance measurements from different instruments in the ultraviolet, visible, near-infrared and thermal-infrared.

This framework is being developed to provide a community available software package that uses robust software engineering practices with well tested, community accepted algorithms and techniques. ReFRACtor is geared not only for the creation of end to end production systems, but also for use by independent investigative scientists who need a software package to help answer atmospheric composition questions.

The diagram in the bottom left panel contains a representation of a high level view of the components of the ReFRACtor system. This high level view glosses over many details. Each of the boxes in this diagram have abstract representations with one or more implementations.

ReFRACtor is designed to be configurable for a wide range of instrument types across the typical remote sensing wavelengths.

It enables retrieving physical quantities from many different instruments not only individually but in joint retrievals that facilitates improved sensitivity to the entire vertical column of air through data fusion.

It contains a versatile, extensible and configurable retrieval system.

It provides a radiative transfer solution that can model radiances from the thermal to ultraviolet spectral regions with multiple scattering and thermal emission capabilities.

Spectroscopy information is provided through an open source tool built for us by the Atmospheric and Environmental Research company. It unifies the usage of absorption coefficient tables and cross section tables for a seamless transition from thermal to ultraviolet in terms of data interfaces.

Aerosols are handled through the specification of scattering and extinction moments. These values can be supplied by the user or our existing particle types can be utilized.

Multiple well-tested implementations of Non-Linear Least Squared solvers are provided, each making different assumptions about the problem state.

The retrieval interfaces separate the definition of the atmospheric retrieval problem and statistical approach from the solver itself while allowing for custom cost function implementations.

Retrievals can be performed using a single solver in a global pan spectral minimization or using a multiple step approach. The multiple step approach chains together multiple solver configurations that minimize different species and spectral regions while fixing the rest of the atmospheric state as static.

We have adaptations for the CrIS, OMPS, OMI and OCO-2 instruments and are beginning work on TROPOMI.

And finally ReFRACtor provides a Python interface whereby components can be used piecemeal or through a robust configuration system. The system can be run from the command line or interactively through Jupyter notebooks using the same configuration. In this manner you can load pieces of the system into interactive notebooks for exploration and debugging.

Much of the core of ReFRACtor is written using C++ and Fortran. But, our Python interface allows for many of those components to be directly replaceable by Python classes.

Complexity Management Examples

The first diagram in the middle panel is an example of ways in which ReFRACtor decouples components from each other through abstract interfaces.

This retrieval solver system design is emblematic of the level of software engineering and the decoupling of components that has been worked on throughout the ReFRACtor code base.

In this example the decoupling has enable the usage of different solvers without changing the problem specification. This allows for missions to fine tune the problem implementation without modifying the core solver algorithms. Different solvers can be used at different stages of a multi-step retrieval.

On top of the solver architecture we have another layer of that allows for a generic way to map the representation of retrieval components to their forward model representation. Components

modeling the representation of physical quantities do not directly handle translation of values from their retrieval representation to their forward model representation. Instead these components use a mapping that defines the translation. This mapping interface allows for the definition of transformation methods that can be reused by components that represent different types of values. It allows adding new mapping mechanisms without changing physical quantity representation classes.

In the second diagram of the middle panel you can see circled the components from the ReFRACtor high level diagram that are the minimum needed to be changed when adapting the system for a new instrument.

First off, a new instrument radiance reader will need to be written to translate mission specific input files to the ReFRACtor interfaces. Next an instrument model definition will need to be defined. This involves the definition of an instrument's spectral response function. Should the spectral response function be representable as a table of values, then ReFRACtor provided classes can be reused. But there is always the option to extend the interfaces to provide for instance an empirically defined spectral response. In fact we have used this approach in some of our instrument adaptations. Spectroscopy information and be provided through our ABSCO tool or new components can be written to compute this information as long as they adapt to the ReFRACtor interfaces.

Outside Perspectives

Lastly, we are always willing to learn from what works and what doesn't from individuals outside the core development team using our software.

Joseph Mendonca (men-don-ka) & Sébastien Roche (row-sha) worked with ReFRACtor for the AIM North Simulation Experiment.

AIM North is a proposed satellite mission consisting of two satellites in highly elliptical orbits observing the northern latitudes to monitor greenhouse gases, air quality and vegetation.

They wrote Python code that sets up a configuration for either a Fourier transform spectrometer or grating spectrometer instrument design. This was used to run ensembles of retrievals and store the output in NetCDF files.

This adaptation creates calculated spectra with given conditions, adds noise, and then fits them under different conditions or with different perturbations in order to model the effects on the precision/accuracy of retrieved gases from different viewing geometries, atmospheric conditions and albedos.

For them, having a Python interface meant that it was faster to code, run simulations, and make diagnostics. Being able to make quick tests in IPython or within Jupyter notebooks made developing much more interactive.

Using ReFRACtor interfaces they implemented custom instrument functions, radiance readers and noise models to match the AIM-North instrument.

However, they were slowed down by a learning curve due to needing to learn about dependency installation, configuration and input specification.

Furthermore, there are still behaviors in the framework that they wanted to modify but they have not been made available for user adaptation due to assumptions made in the heritage implementations.

The code was flexible enough in some areas they could easily implement their own Python objects but difficult in other areas where they needed help from the ReFRACtor developers to make the necessary changes. Ideally, they would like the flexibility that each object in the config file could easily be swapped out for a Python object.

Overall they think ReFRACtor was very helpful to their work and was much easier to use and adapt to than an older code that was also considered.

Jonathan Hobbs has been working with ReFRACtor to drive his uncertainty quantification experiments for OCO-2.

He thinks that ReFRACtor's capability to distinguish the forward model specification from the retrieval setup greatly aids the UQ experiment setup and execution. The experiment's synthetic radiances can be generated repeatedly with a forward model evaluation call and any retrieval execution can be deferred to a later step. He thinks this capability will facilitate research on alternative retrieval methods, such as Markov chain Monte Carlo. Researchers will be able to develop customized cost functions and sampling algorithms that may only need to interface with a forward model evaluation versus a full nonlinear optimization.

However, he would like to see a standard output file specification that would include retrieved states, spectral fit information (e.g. cost function, spectral residual summaries), optimization diagnostics (iterations, outcome, etc.), and error analysis where possible.

Overall he thinks that the engineering approach devised for ReFRACtor makes it as useful or more useful than the standard mission software for many UQ investigations.

Conclusion

It is impossible for one system to cover all use cases, but you can go a long way using rigorous software design that is agnostic to implementation details. We try to design robust interfaces to the best of our knowledge. But, as use cases expose edge conditions we seek to adapt and improve. We have built ReFRACtor with the intention of it becoming a community model. We welcome new partnerships that help improve the overall software design and architecture so that

we all benefit. Please contact us if you interested in working together with us. Thank you for your attention.