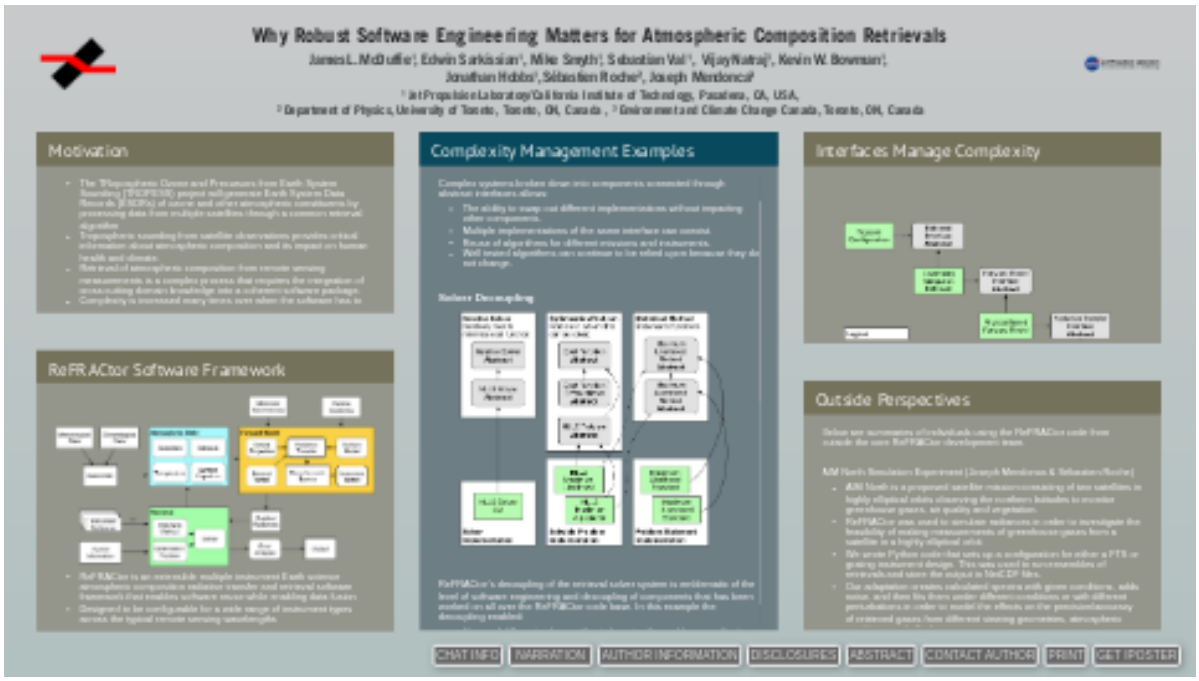
Why Robust Software Engineering Matters for Atmospheric Composition Retrievals



James L. McDuffie[1], Edwin Sarkissian[1], Mike Smyth[1], Sebastian Val[1], Vijay Natraj[1], Kevin W. Bowman[1],
Jonathan Hobbs[1],Sébastien Roche[2], Joseph Mendonca[3]

[1] Jet Propulsion Laboratory/California Institute of Technology, Pasadena, CA, USA,
[2] Department of Physics, University of Toronto, Toronto, ON, Canada , [3] Environment and Climate Change Canada, Toronto, ON, Canada

Jet Propulsion Laboratory
California Institute of Technology

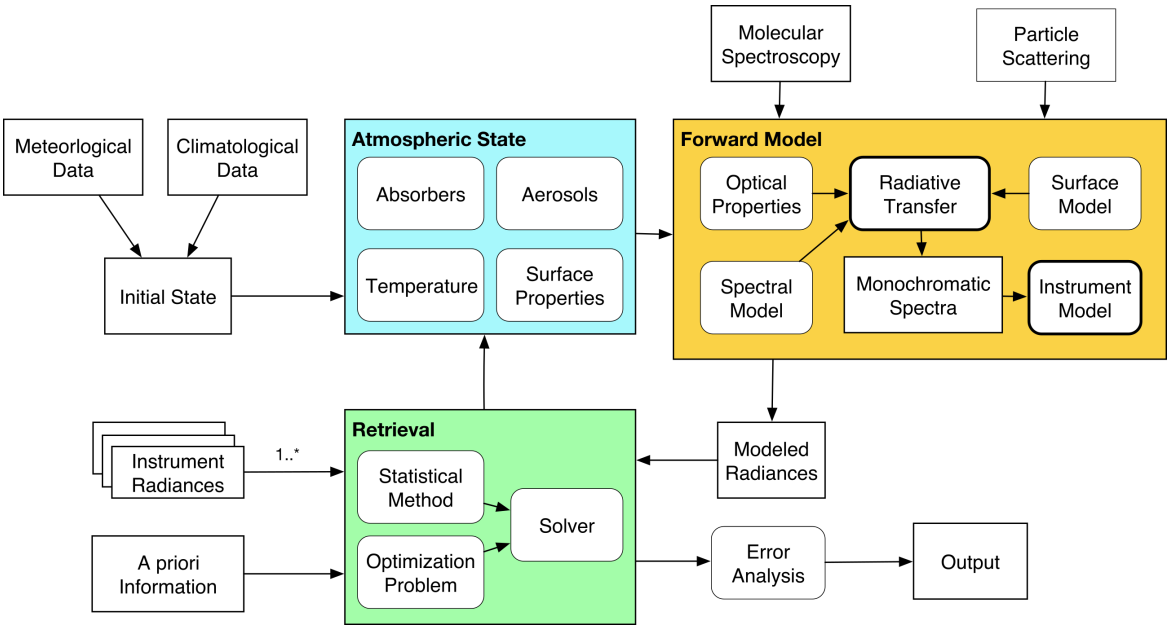
PRESENTED AT:

AGU FALL MEETING
Online Everywhere | 1–17 December 2020

MOTIVATION

- The TRopospheric Ozone and Precursors from Earth System Sounding (TROPESS) project will generate Earth System Data Records (ESDRs) of ozone and other atmospheric constituents by processing data from multiple satellites through a common retrieval algorithm
- Tropospheric sounding from satellite observations provides critical information about atmospheric composition and its impact on human health and climate.
- 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.
- Complexity is increased many times over when the software has to handle multiple types of instruments, operating in different spectral regions, each with their own peculiarities.
- Managing this complexity requires robust software engineering practices, it requires a partnership between software developers and scientists
- Our software framework, ReFRACtor addresses these issues through robust component interfaces enabling flexibility and reusability
- 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

REFRACTOR SOFTWARE FRAMEWORK



- ReFRACtor is an extensible multiple instrument Earth science atmospheric composition radiative transfer and retrieval software framework that enables software reuse while enabling data fusion
- Designed to be configurable for a wide range of instrument types across the typical remote sensing wavelengths
- 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
- Versatile, extensible and configurable retrieval system
- 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 from an open source Atmospheric and Environmental Research (AER) provided tool that 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 existing particle types can be utilized.
- Multiple well-tested implementations of Non-Linear Least Squared (NLLS) solvers are provided with 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.
- Have adaptations for the CrIS, OMPS, OMI and OCO-2 instruments, beginning work on TROPOMI adaptation.

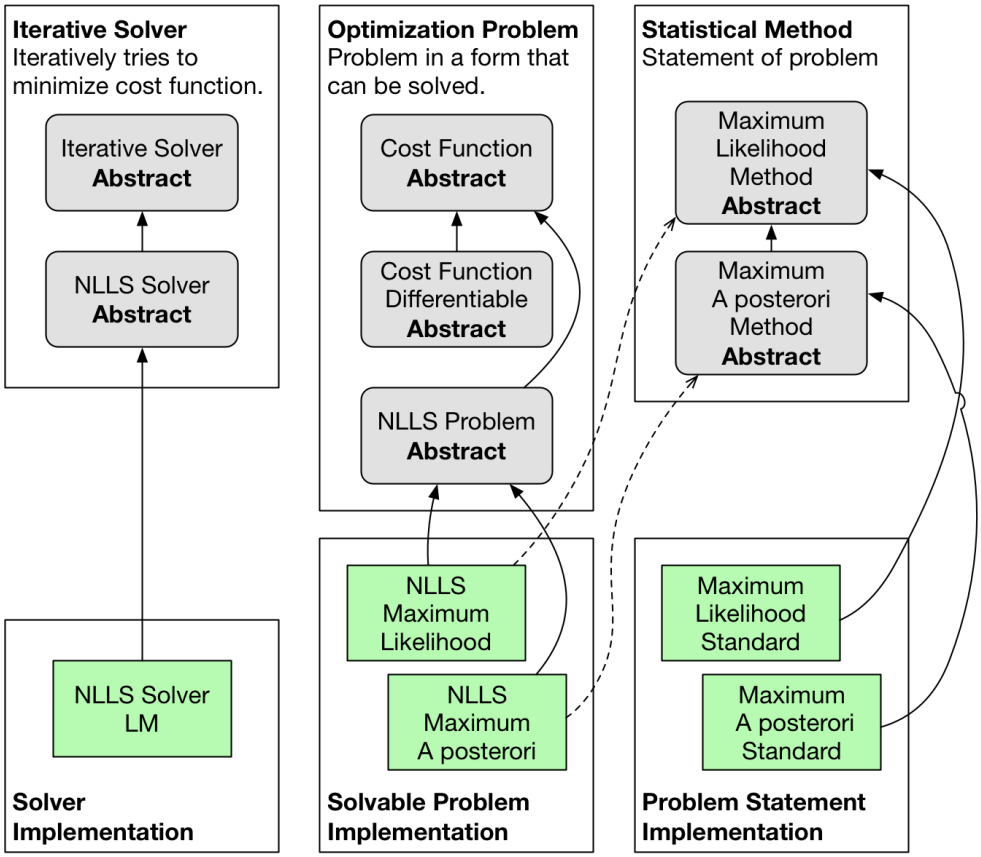
- Provides a Python interface whereby components can be used piecemeal or through a robust configuration system

COMPLEXITY MANAGEMENT EXAMPLES

Complex systems broken down into components connected through abstract interfaces allows:

- The ability to swap out different implementations without impacting other components.
- Multiple implementations of the same interface can coexist.
- Reuse of algorithms for different missions and instruments.
- Well tested algorithms can continue to be relied upon because they do not change.

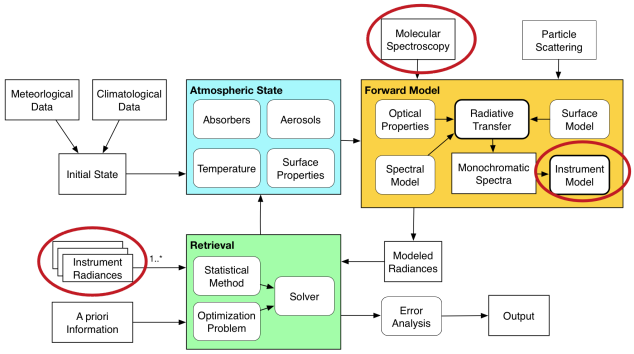
Solver Decoupling



ReFRACtor's decoupling of the retrieval solver system is emblematic of the level of software engineering and decoupling of components that has been worked on all over the ReFRACtor code base. In this example the decoupling enabled:

- Usage of different solvers without changing the problem specification.
- The ability for missions to fine tune the problem implementation without modifying the core solver algorithms.
- Multiple solver usage at different stages of a multi-step retrieval.
- Another layer of interface designed, not shown above, allows for a generic way to map the representation of retrieval components to their forward model representation.

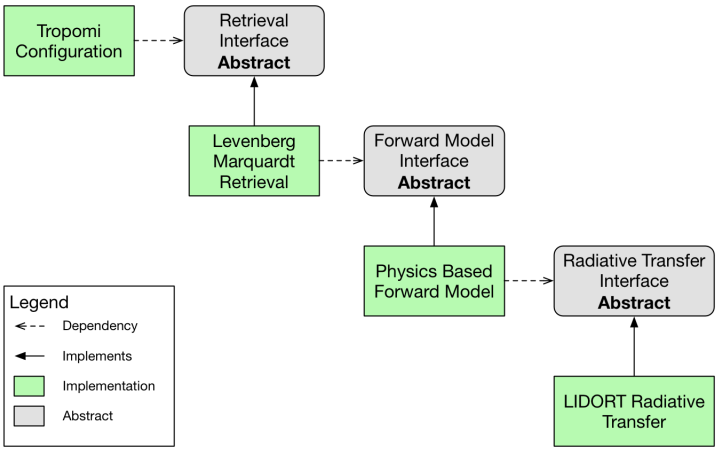
Instrument Adaptations



New instrument implementations as seen circled above nominally need only provide:

- Instrument specific configuration pulling together desired components
- A way to read instrument radiances
- Instrument model definition
  - Can reuse existing models if providing spectral response as a table
  - But empirical methods can be easily implemented
- Spectroscopy for molecular absorption
  - Can use our ABSCO tool to generate tables from thermal to UV
- Absorption coefficient tables generated for the instrument spectral regions using the framework provided tool.

INTERFACES MANAGE COMPLEXITY



- The key software engineering aspect that helps manage the complexity of atmospheric composition retrieval software is the use of well thought out abstract interfaces
  - Define an expected contract that implementations need to uphold
  - Cannot be instantiated itself
- Abstract interfaces decouple behavior from implementation.
  - Should be designed to make the fewest assumptions on implementation as possible.
- High level modules should not depend on low level modules, instead should depend on abstractions.
- Abstractions should not depend on details. Instead, details should depend on abstractions.
- Components following this policy become:
  - "Open for extension"
    - Behavior can be extended as requirements change.
  - "Closed for modification"
    - Working, tested software should not change to meet requirements.

The diagram above 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.

OUTSIDE PERSPECTIVES

Below are summaries of individuals using the ReFRACtor code from outside the core ReFRACtor development team.

AIM North Simulation Experiment (Joseph Mendonca & Sébastien Roche)

- 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.
- ReFRACtor was used to simulate radiances in order to investigate the feasibility of making measurements of greenhouse gases from a satellite in a highly elliptical orbit.
- We wrote Python code that sets up a configuration for either a FTS or grating instrument design. This was used to run ensembles of retrievals and store the output in NetCDF files.
- Our 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.
- 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.
- We implemented custom instrument functions, radiance readers and noise models to match the AIM-North instrument design using ReFRACtor's interface specifications.
- We were slowed down by a learning curve due to needing to about dependency installation, configuration and input specification.
- There are still behaviors in the framework that we wanted to modify but they have not yet been made available for user adaptation due to assumptions made in the heritage implementations.
- The code was flexible enough in some areas so we could easily implement our own Python objects but difficult in other areas where we needed help from the developer to make the necessary changes. Ideally, the code would be flexible enough that each object in the config file could be easily swapped out for a Python object.
- Overall ReFRACtor was very helpful to our work. It was much easier to use and adapt to than an older code that was also considered.

Uncertainty Quantification Experiments (Jonathan Hobbs)

- My primary use of ReFRACtor thus far has been for conducting simulation experiments for uncertainty quantification (UQ) using the OCO-2 interface.
- Having some familiarity with the OCO-2 retrieval framework, the ReFRACtor interfaces were not very foreign to me when I got started. It did take some trial and error to find certain specifications in the different layers of the objects. For example, setting the state vector to a user-specified value requires diving into the configuration hierarchy a couple of levels. Noise model uncertainty is another one that takes some digging. In the end, it is fairly intuitive to find the key elements once I got a handle on the conventions.
- 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. I also think 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.
- The primary ReFRACtor extensions I have worked on involve generating output (HDF5) with retrieval results and spectral diagnostics. I think it would be useful to eventually have (if they do not already exist) some standard Level-2 like output files produced by the various ReFRACtor instrument and retrieval interfaces. Broadly speaking, these 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 I think the engineering approach devised for ReFRACtor makes it as useful or more useful than the standard mission software for many UQ investigations.

DISCLOSURES

A portion of this work was performed at the Jet Propulsion Laboratory, California Institute of Technology, under contract with NASA

AUTHOR INFORMATION

James McDuffie is the ReFRACtor team lead. He has 20 years of scientific data software engineering experience with extensive experience with radiative transfer and retrieval algorithms. He worked on both the TES and OCO Level 2 algorithms. He led the refactoring and redesign of the OCO radiative transfer and retrieval code into software for use on GOSAT and OCO-2 data. He was PI on the AIST funded Multi-Instrument Radiative Transfer and Retrieval Framework project which produced the open source ReFRACtor software. He is currently the JPL group 398E Technical Group Lead for Next-Generation Radiative Transfer (RT) Retrieval Framework. Additionally, he is the cognizant engineer for the MAIA L2 and L4 software and Algorithm Design Environment lead for the Unity project.

ABSTRACT

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. The complexity is increased many times over when the software has to handle multiple types of instruments, operating in different spectral regions, each with their own peculiarities. This is further compounded when trying to combine information from multiple instruments for joint retrievals. Yet, there is enough overlap between the radiative transfer and retrieval techniques used by various missions that it is wasteful to continually reinvent the wheel every time.

The Reusable Framework for Atmospheric Composition (ReFRACtor) 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- 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 towards independent investigative scientists who need a software package to help answer atmospheric composition questions.

We explain how the use of succint interfaces between components provides advantages for future proofing, flexibility and reusability. Examples will be given for translating the logical separation of mathematical and scientific concepts into software components. The experience of early adopter scientists will also be discussed to give a perspective from outside the software development team.