Geoscientific Model Development Discussions

Geosci. Model Dev. Discuss., referee comment RC2
https://doi.org/10.5194/gmd-2021-419-RC2, 2022

Comment on gmd-2021-419

Ethan Coon (Referee)

Referee comment on "UniFHy v0.1.1: a community modelling framework for the terrestrial water cycle in Python" by Thibault Hallouin et al., Geosci. Model Dev. Discuss., https://doi.org/10.5194/gmd-2021-419-RC2, 2022

This manuscript describes a first release of a new open-source codebase aiming to enable flexible, configurable, component-based modeling of integrated hydrology, specifically land surface processes, overland routing, and subsurface flows. They describe a set of APIs enabling codes that meet, or can be wrapped to meet, these three component APIs to be coupled for hydrologic applications. The code surrounding these components provides for a standard way of configuring these components using YAML files, mapping between these components using the ESMF codebase, and supplying input dataset files by reading CF-netCDF files. The manuscript describes how the framework was used to run three demo simulations on catchments, using three contributed codes that meet the API – Artemis, RFM, and SMART. They show how the model components can be swapped with data-driven components or "null" components where the flux is zero. Finally, they describe a "roadmap" for future work within the codebase.

In general, I find the key motivation appropriate, and agree strongly with their motivating goals to enable work in computational hydrology, specifically that the resolution of land system coupling is too coarse, and that a focus on vertical exchange limits lateral redistribution. The authors successfully argue that there is room to bridge the Earth system/land surface modeling communities with the watershed-scale hydrologic modeling communities to generate a new suite of intermediate-scale resolution models for system-level understanding. They argue that a coupling framework is important to bridge these scales and investigate how process representations at these scales affect other modeling components.

As a result, I find this work to be timely and appropriate for GMD. I was happy to see this article as a more formal attempt at coupling codes at these scales, as opposed to the majority of "hard-coded interfaces" currently used to directly couple two existing codes without thought for substitution or other advantages provided by a formal API.

Finally, the article is stylistically very well written, with concise yet appropriate descriptions of the capability. Similarly, the source code was reasonably well commented, named, and organized, and followed many open source "best practices."

That said, there are three major missing considerations in this manuscript that keep me from recommending it be published. First, it is unclear what in this work is new or novel relative to existing efforts. Second, as this paper is a "code description," it should clearly be tracible and reproducible so that a user could execute the demos described here and adapt them for their own work, but I was unable to do more than install the codebase. Third, it is unclear how flexible this framework really is, as it appears their choice of flux variables and fixed-granularity component API limits the scope of a given component to a very narrow set of codes. Each of these three points are described in detail below. Following these, I raise several minor comments and concerns.

What is new in this framework relative to existing work?

The authors provide a reasonably complete description of existing coupling tools, including key players such as underlying technologies for "coarse-grained" coupling technologies (MCT, CPL, ESMF, CSDMS/LandLab, etc). However, they fail to describe what makes their framework new or even an improvement upon these technologies. In fact, I'm fairly sure that everything described in this framework could have been directly implemented in the CSDMS codebase with no new framework. I believe CSDMS already implements all described features of this codebase, including existing time-stepping and spatial schemes, python-based drivers and configuration of models, remapping capabilities through third-party libraries, and existing interfaces for passing data between these models. If one were to eliminate the requirement of "in python," I suspect several other tools would meet the needs of this effort as well (e.g. MCT, BMI, others), although maybe not with bent toward hydrology.

A major revision is required to make clear what, if anything, is unique or novel about this framework relative to these other coupling frameworks.

To be clear, if there is nothing novel about this work other than it is more specifically tailored for hydrologic models and some hydrologic models have already been adapted to this framework, I don't think that it would immediately disqualify this from being published as a code description. But it must be clear how the work relates to existing work.

Reproducibility and Software Quality

As noted above, this submission clearly fits GMD's concept of a "model description" paper. In particular, "model description" papers should be held to a high standard of tracibility and/or reproducibility – the reader should be able to download and install the code, run a basic example, and gain some concept of how to use the model for any "model description" paper. The following describes my experience in trying to download the code, run the tests, use the tutorial, and finally to use the user's guide and training material.

First, I followed the cited DOI, which took me to the github repository. At first glance, the software repository seemed well organized. Browsing through the repository suggested that the code could be installed via pip (a requirements.txt file existed), and the code source was readable, reasonably well annotated/commented, and included template for new components. Unfortunately, it was not obvious to me as a user how to begin to use the codebase.

Please add a link to documentation on the README page and in the manuscript – as the DOI points to the github repo, the "new" user will not know how to find the documentation. In the end, I was able to guess where to find the documentation by looking for github pages, but this should not be required.

Next, I tried to install the code, following the installation documentation. I cloned the latest version, noting the requirements.txt file, and tried to install via pip, with both the recommended `pip install unifhy` and `pip install -r requirements.txt`. These failed with the message:

>ERROR: Could not find a version that satisfies the requirement esmpy (from unifhy) (from versions: none)

>ERROR: No matching distribution found for esmpy

I was able to install the base framework package via conda – success! Next, I noted that the codebase had unit tests, which I was pleased to see. Running the tests via `cd tests;

python run_all_tests.py` resulted in:

>Ran 90 tests in 539.397s

>FAILED (errors=104, expected failures=5)

Note that this is probably not a big deal, as probably I am running the tests incorrectly, but it would be good if the user's guide described how to run the tests. Successful tests give the user confidence that the code is still supported and that they have correctly installed the needed requirements.

I was pleased to see a tutorial; I tried to follow the tutorial in the documentation at: https://unifhy-org.github.io/unifhy/users/tutorial.html. It does not appear that this was intended to actually be executed? The required netCDF files were not in my repository, and there did not seem to be a repository called "tutorial" or similar which contained these files. None of the documentation described where to get these files, so I gave up and moved on.

Next, I saw there was a training repository in unifhy-org. I cloned this, started up a jupyter notebook, and attempted to run the demo problem there. Immediately I ran into the problem that the contributed models (e.g. artemis, rfm) are not provided with the main repository. Furthermore, there was no documentation in the manuscript, training repository, or the corresponding artemis/rfm repositories on how to install these component models. Again, I began to guess, trying conda (no luck) before successfully installing them by cloning their respective repositories and running `python setup.py install`. This got me to the next step in the demo problem, before, once again, running into an issue where the demos relied on netCDF files that did not exist anywhere that I could find – the notebook suggests they should be at ` /data/demo-data/` but I suspect this demo is intended for local users of a private machine somewhere.

In all, while tutorials and demos existed (good) I was unable to run even a simple example (bad). The documentation, while it does exist (good) is incomplete and misses key "how to" descriptions to help the user get started (bad). If a GMD "model description" paper is intended to make clear that a new code is available and usable by the community, that code should be clearly usable by the community, which it does not appear to be. Prior to acceptance, this would have to be significantly improved to the

point that a user could follow reasonable descriptions to run a demo problem using the code on their own machine.

Finally, as far as I can find, no "assets" or data repositories are provided with regards to input files or other files needed to trace or reproduce the “selected configuration” simulations run and described in the manuscript.

In summary, I would expect to, with reasonable effort, be able to:

- Download and install the code and all requirements
- Run the tests
- Run an example or tutorial
- Have access to the complete input files needed to reproduce the demos shown in the paper.

I was only able to do #1 successfully, and that required some guessing to even find the documentation.

As a result of this effort, I believe that the reproducibility and traceability of this paper is insufficient to meet the standards of GMD.

How flexible is the defined API?

In this work, the authors define a very narrow API for a fixed set of fluxes between exactly three components. This raises several concerns. As examples:

- What happens if a component does not supply all the needed variables? For instance, one might want to use a simple empirical model, e.g. the Priestley-Taylor model, to provide a total evapotranspiration flux. While this could probably be used within another component to supply the remaining fluxes (e.g. PRMS implements a Priestley-

Taylor model and could supply the rest of the fluxes), it does not appear that the API is flexible enough to "split" the API across multiple, smaller-grained components.
- What happens if a model would like to integrate more tightly than the current flux-based coupling? For instance, integrated hydrologic codes (e.g. those described in Kollet et al WRR 2017) often tightly couple surface and subsurface flow, meaning that there is no "permanent open water." Codes such as these would benefit from an API to provide a coupling to land surface codes. It is unclear whether this API would support extension to include not just fluxes but also primary variables e.g. head or pressure (likely not) or whether lumping the surface and subsurface components together into a single component is supported (maybe?).

In general, this coupling capability seems to be very narrowly aimed at a specific class of codes (e.g. Artemis, RFM, and SMART and others that are structured grid, routing + infiltration + land surface model components), and that this "fixed granularity" of the coupling API means it is not very useful outside of a very narrow class of codes. While this is not a deal-breaker, it should be addressed by a clear statement of the class of codes that fit this API, and how or if the granularity of this API could be changed or made more flexible. *I think this addresses an important class of models, but it should be clear what class of models this is.*

For instance, another approach would be to allow arbitrary collections of models to be registered into a shared state; at runtime the driver ensures that all fluxes that are needed are supplied by a component, thereby checking that the model is "well-posed." This allows the granularity to change as needed, even within a flux-coupling concept such as this (e.g. CSDMS, Tucker et al GMD 2022). Other more general multi-physics codes (e.g. Coon et al Env. Model Software 2016) use dependency graphs to ensure that coupling needs are met and allow arbitrary components that meet a generic API to be coupled, allowing even greater flexibility in what it means to be a component. These seem much more flexible than this API, which is more of a fixed-granularity, "role-based" component concept. *The manuscript should do a better job to clarify what it means by flexible.*

Minor Points:

- The authors state that: "These models do not allow different parts of the land system to be simulated at different explicit resolutions." The same criticism is true of this effort; only structured Lat/Long meshes are described here.
- The introduction is missing a short discussion of existing ESM/Integrated Hydro Terrestrial Modeling approaches that bridge this scale in hydrologic modeling. For instance, work in so-called hyper-resolution hydrology has started to bridge this scale/community divide, and examples such as the US's National Water Model via WRF-Hydro, Maxwell et al GMD 2015's ParFlow papers, and several others have started to bridge this gap (see, for a summary, Paniconi and Putti, WRR 2015 for a good, if not

very current, summary). While these are not general-purpose coupling frameworks, they are currently providing models that cover the same set of processes and scales targeted by this framework.

- In the section on discretization, the authors describe that timestepping schemes must be integer multiples of each other in order to synchronize at the needed intervals. It is unclear why this would be true – it seems that a component model ought to be able to do whatever it needs to integrate across the interval. For instance, this would preclude adaptive timestepping, which may be needed in implicit subsurface codes.
- The authors motivate the need for a shared naming and units convention for codes to be coupled successfully. However, there has been significant efforts to identify ontologies that formalize variable names and units for use in coupling codes; this work was not considered or cited in this manuscript. Specifically, CSDMS's Standard Names and the work by Scott Peckham and collaborators formalize an ontology for land surface modeling, and the CF conventions standard names provides a framework for defining variable names in climate applications. It would be preferable to adopt one of these standards where possible rather than create a new one for this work. As one example, CF conventions defines "canopy_throughfall_flux" that could be adopted instead of "canopy_liquid_throughfall," etc.
- There are a lot of potentially useful things described in the future work discussion, but none of these are currently in the codebase, and many of them are available in other comparable products. For instance, unstructured meshes may be coupled through a variety of general-purpose remapping tools (MOAB, DTK, TempestRemap, and many others). Parallel execution is supported in MCT, CCA, and even ESMF which is used here, and has been developed for BMI by the National Water Model. I do agree that a formal API enabling subgrid connections (e.g. PFT subgrid concepts, etc) for heterogeneity is an exciting research area, however. These sections should either acknowledge that such work already exists or be removed completely.
- Lastly, object-oriented design has a lot of advantages for defining interfaces, but often sets limitations on data structures and data dependencies. I would have liked to have seen a description of the limitations of this data model. For instance, can the exchanged fluxes be GPU memory? MPI-parallel vectors?