The Journal of Open Source Software

The Babelizer: language interoperability for model coupling in the geosciences

Eric W. H. Hutton[1], **Mark D. Piper**[1], **and Gregory E. Tucker**[123]

1 Community Surface Dynamics Modeling System, University of Colorado Boulder **2** Cooperative Institute for Research in Environmental Sciences (CIRES), University of Colorado Boulder **3** Department of Geological Sciences, University of Colorado Boulder

DOI: 10.21105/joss.03344

Software

- Review
- Repository
- Archive

Editor: Kristen Thyng

Reviewers:

- @lheagy
- @cheginit

Submitted: 09 April 2021
Published: 19 March 2022

License
Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License (CC BY 4.0).

Summary

The *babelizer* is a Python utility that generates code to import libraries from other languages into Python. Target libraries must expose a Basic Model Interface (BMI) (Hutton et al., 2020; Peckham et al., 2013) and be written in C, C++, or Fortran, although the *babelizer* is extendable, so other languages can be added in the future. The *babelizer* provides a streamlined mechanism for bringing scientific models into a common language where they can communicate with one another as components of an integrated model.

Statement of need

With an integrated multicomponent approach to modeling, scientists–not just software developers–connect components to form integrated models, where plug-and-play components can easily be added or removed (Collins et al., 2005; David et al., 2013; Gregersen et al., 2007; Tucker et al., 2022). This is in contrast to older methods, where a single modeling group would construct a monolithic model built on the tight integration of software written within an isolated framework. A single person or group would control model development. Outside contributors would go through a gatekeeper to ensure compatibility. The software elements that made up the model would be tied to the larger model and, generally, not used outside of the framework.

Component modeling democratizes model building by empowering the larger scientific community to develop model components. This allows for more innovation and experimentation driven from the bottom up by a community. It reduces redundancy–rather than reinventing software, scientists can find and use models that suit their needs–and it allows scientists to focus on new and unsolved problems.

There are disadvantages, however. Without a single group to guide model development, there is a greater risk that community-developed models will become incompatible with one another. With hundreds of scientists developing models in isolation, there is a greater likelihood models will be written with idiosyncratic designs, incompatible grids, incompatible time steps, and in different programming languages. The Earth-surface modeling community has developed tools to help solve some of these problems. For example, the Basic Model Interface standardizes model interactions. The Earth System Modeling Framework (ESMF) (Collins et al., 2005) grid mappers are able to map quantities from one grid to another. The Python Modeling Toolkit *pymt* (Hutton & Piper, 2020) performs time interpolation, grid mapping, and unit coversion. In this paper, we present a solution to the language incompatibility problem.

Overcoming the language incompatibility problem

To get an idea of the range of programming languages used in Earth-surface modeling, we can look to the Community Surface Dynamics Modeling System (CSDMS) model repository. As of June 2020, the repository holds over 370 open source models and tools submitted by the community. These contributions span a range of languages, with Python, C, and Fortran being the most popular (Figure 1). The mix of languages raises an interesting challenge in creating an interoperable modeling framework. Our solution is to use a hub-and-spoke approach, where Python is the hub language that connects to other languages. We chose Python as the hub because of its popularity in the scientific community, its extensive collection of third-party libraries (including model coupling frameworks such as the *pymt*), and its existing ability to communicate with other programming languages. We have built the *babelizer* to generate the spokes that connect Python to other languages.

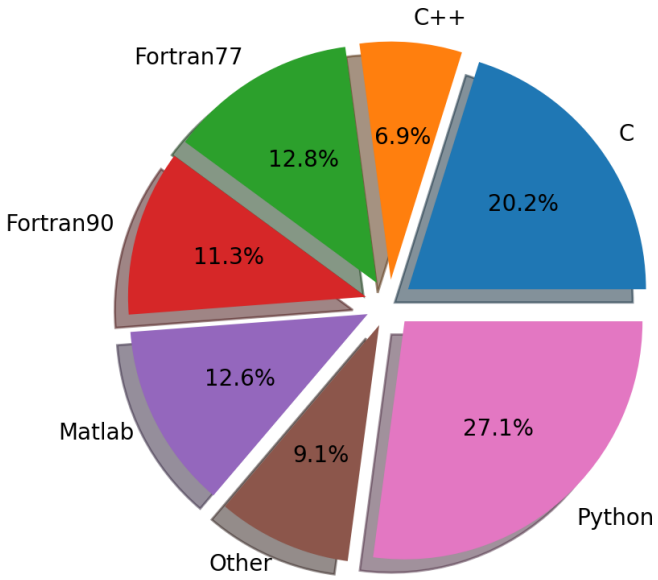

Figure 1: The distribution of programming languages used in the models in the CSDMS model repository. (Data from https://csdms.colorado.edu/wiki/CSDMS_models_by_numbers.)

Using the CSDMS model repository as a guide, if we build translators for the open source languages C, C++, Fortran, and Python, we will cover 80 percent of the contributed models. A drawback of using Python is that it can be relatively slow compared to compiled languages like C and Fortran; however, the models being wrapped are compiled and run in their native language, which is where the bulk of the computation takes place, with the *babelizer* providing only a thin wrapper layer.

The Common Component Architecture

The *babelizer* is, in part, inspired by the work of Epperly et al. (2012) and their development of the Common Component Architecture (CCA) tools and, in particular, the multi-language compiler, *babel*. *Babel* is not a compiler *per se* but, rather, a code generator that produces glue code to provide cross-language interoperability. The generated code is then passed to a

traditional compiler to build libraries. Whereas the *babelizer* uses Python as the hub language to connects its supported languages, *babel* itself acts as the hub, generating spokes to each supported language so that languages talk directly to one another. That is, *babel* is capable of generating bridges from each of its supported languages directly to every other supported language. *Babel* also supports arbitrary interfaces while the *babelizer* is only able to wrap libraries that expose a BMI.

Grpc4BMI

Another alternative solution to the language interoperability problem is to treat models as web services that expose a BMI; e.g., *Grpc4BMI* (Hut et al., 2021). In such a framework, models are built on separate servers or within their own software container (e.g. Docker) and interact with clients through network ports. Isolating models within environments eliminates the potential of dependency conflicts between models and requires them to only be built within one specific environment or operating system.

Design of the babelizer

The *babelizer* is a command-line utility that generates the glue code to bring a model exposing a BMI from another language into Python. Because the BMI is a well-defined standard, the *babelizer* requires only a small amount of metadata to generate the glue code. The metadata depends somewhat on the language being wrapped, but includes the name of the library providing the BMI, the name of an entry point into the library, the language the library was written in, and any necessary compiler flags. With this metadata, the *babelizer* creates a new *git* repository, a Python package containing the Python interface to the model, documentation, and sets up continuous integrations and a test suite for the model's BMI. The model can then be imported and run through Python.

The user provides metadata describing their model through a *toml*-formatted file (see Figure 2 for an example). The *babelizer* uses the metadata to fill a set of *jinja*-formatted template files to construct the new repository (or update an existing repository). The entire repository is almost completely auto-generated, which means it can easily be regenerated. The only files a user need edit are the main configuration file, `babel.toml`, and any optional model data files, which are installed along with the new component.

Data files provided to a babelized component are intended to be used either by a user of the new component or by a separate framework that imports the component. There is little restriction on the contents of the files, but typically they are sample input files that a user of the component can use to run the model. Another use-case is where the component will be used within a separate modeling framework and that framework may require additional metadata. As an example, the *pymt* is a modeling framework able to work with generic BMI models. In addition to the BMI, the *pymt* requires descriptive information about the model (e.g. authors, license, references, summary of what it does, etc.) as well as *jinja*-formatted sample input files. The *pymt* uses the template input files as part of a utility for a user to programmatically generate model input files without having to know anything about the idiosyncratic details of those model input files. Within such a framework, therefore, a user is given model components with a standardized way to create input files, as well as a common Python interface to run and interact with the model.

```
[library]
[library.PRMSSurface]
language = "fortran"
library = "bmiprmssurface"
header = ""
entry_point = "bmi_prms_surface"

[build]
undef_macros = []
define_macros = []
libraries = []
library_dirs = []
include_dirs = []
extra_compile_args = []

[package]
name = "pymt_prms_surface"
requirements = ["prms", "prms_surface"]

[info]
github_username = "pymt-lab"
package_author = "Community Surface Dynamics Modeling System"
package_author_email = "csdms@colorado.edu"
package_license = "MIT"
summary = "PRMS6 surface water process component"

[ci]
python_version = ["3.9"]
os = ["linux", "mac", "windows"]
```

Figure 2: The *babelizer* configuration file (`babel.toml`) for the Precipitation-Runoff Modeling System v6 surface water component, *PRMSSurface* (Piper et al., 2020). Running the `babelizer` with this file produces most of the repository https://github.com/pymt-lab/pymt_prms_surface.

Acknowledgements

This work is supported by the National Science Foundation under Grant No. 1831623, *Community Facility Support: The Community Surface Dynamics Modeling System (CSDMS)*.

References

Collins, N., Theurich, G., Deluca, C., Suarez, M., Trayanov, A., Balaji, V., Li, P., Yang, W., Hill, C., & Da Silva, A. (2005). Design and implementation of components in the earth system modeling framework. *The International Journal of High Performance Computing Applications*, *19*(3), 341–350. https://doi.org/10.1177/1094342005056120

David, O., Ascough, J. C., Lloyd, W., Green, T. R., Rojas, K. W., Leavesley, G. H., & Ahuja, L. R. (2013). A software engineering perspective on environmental modeling framework design: The object modeling system. *Environmental Modelling & Software*, *39*, 201–213. https://doi.org/10.1016/j.envsoft.2012.03.006

Epperly, T. G., Kumfert, G., Dahlgren, T., Ebner, D., Leek, J., Prantl, A., & Kohn, S. (2012). High-performance language interoperability for scientific computing through babel.

The International Journal of High Performance Computing Applications, *26*(3), 260–274. https://doi.org/10.1177/1094342011414036

Gregersen, J., Gijsbers, P., & Westen, S. (2007). OpenMI: Open modelling interface. *Journal of Hydroinformatics*, *9*(3), 175–191. https://doi.org/10.2166/hydro.2007.023

Hut, R., Drost, N., Giesen, N. van de, Werkhoven, B. van, Abdollahi, B., Aerts, J., Albers, T., Alidoost, F., Andela, B., Camphuijsen, J., Dzigan, Y., Haren, R. van, Hutton, E., Kalverla, P., Meersbergen, M. van, Oord, G. van den, Pelupessy, I., Smeets, S., Verhoeven, S., … Weel, B. (2021). The eWaterCycle platform for open and FAIR hydrological collaboration. *Geoscientific Model Development Discussions*, 1–31.

Hutton, E. W. H., & Piper, M. D. (2020). *The python modeling toolkit* (Version v1.0.0) [Computer software]. Zenodo. https://doi.org/10.5281/zenodo.3644240

Hutton, E. W. H., Piper, M. D., & Tucker, G. E. (2020). The basic model interface 2.0: A standard interface for coupling numerical models in the geosciences. *Journal of Open Source Software*, *5*(51), 2317. https://doi.org/10.21105/joss.02317

Peckham, S. D., Hutton, E. W., & Norris, B. (2013). A component-based approach to integrated modeling in the geosciences: The design of CSDMS. *Computers & Geosciences*, *53*, 3–12. https://doi.org/10.1016/j.cageo.2012.04.002

Piper, M., McDonald, R., Hutton, E., Markstrom, S., Parker, N., & Tucker, G. (2020). Coupling hydrologic models with data services in an interoperable modeling framework. *Earth and Space Science Open Archive ESSOAr*. https://doi.org/10.1002/essoar.10504855.1

Tucker, G. E., Hutton, E. W. H., Piper, M. D., Campforts, B., Gan, T., Barnhart, K. R., Kettner, A. J., Overeem, I., Peckham, S. D., McCready, L., & Syvitski, J. (2022). CSDMS: A community platform for numerical modeling of earth surface processes. *Geoscientific Model Development*, *15*(4), 1413–1439. https://doi.org/10.5194/gmd-15-1413-2022