The Journal of Open Source Software

model-traits: Model attribute definitions for scientific simulations in C++

Jacob Merson[1] **and Mark S. Shephard**[1]

1 Rensselaer Polytechnic Institute

DOI: 10.21105/joss.03389

Software

- Review
- Repository
- Archive

Editor: Patrick Diehl

Reviewers:

- @daniellivingston
- @pratikvn

Submitted: 21 May 2021
Published: 28 June 2021

Summary

`model-traits` is a C++ library for setting up scientific models and computational analysis. It provides a minimal API for applying boundary conditions (or other attributes) to the geometry of a model. `model-traits` can either be used directly as a library, or, can be used to generate input files for an existing analysis code. The library design is optimized to make adding new input and output file formats easy and maintainable without patching the core library.

Statement of need

Setting up scientific simulations is often a time consuming process that typically requires hand crafting input files. Most analysis codes take a mesh-first approach to model setup, and mesh storage. When setting up a model for analysis this typically means that the user will have to recreate the input deck for any change in the mesh even if the desired boundary conditions are identical. This problem also exists with many commercial FEM codes such as Abaqus and LS-Dyna (Hallquist, 2006; Smith, 2009).

The essence of the problem is that the mesh is the wrong level of abstraction to apply boundary conditions. This problem also exists with respect to using a mesh-first approach to mesh databases. Many modern meshing databases such as PUMI (Ibanez et al., 2016) and MeshSim (*Simmetrix Inc. - Mesh Generation, Geometry Access*, n.d.) have shown that for many operations model geometry is a preferable level of abstraction and therefore they store the relationship between a meshes' entities and the geometric model topology (Beall & Shephard, 1997). This relationship allows for much more robust mesh adaption that can correctly refine the mesh to better approximate the geometric model.

`model-traits` takes this geometry-first approach to applying various traits on the model such as boundary conditions, solver settings, and other data which is necessary to run an analysis. It is designed as an open source alternative to the model attribute definition provided in the Simmetrix GeomSim package (O'bara et al., 2002). Kitware also develops an open source tool for model attribute definition called the Simulation Modeling Toolkit (SMTK) (O'Bara, 2015; O'Leary, 2014). The design of SMTK uses a deep inheritance hierarchy which makes its use challenging for users who have limited experience with the SMTK codebase. The goal for SMTK's design is also broader, encompassing GUI model setup and a general interface for 3rd party CAD kernels. The `model-traits` library is more narrowly focused on the association between a given model trait and a geometric model topological entity. Although `model-traits` does not provide an interface for any CAD kernels; CAD types, such as a pointer to a CAD kernel's face topological object, can pass through the library without modification through a templated interface. A typical use case is to use two integral types to denote a geometric entities dimension and unique ID rather than explicitly using a CAD kernel in the analysis code.

In `model-traits` templates and type erasure are used to help limit class inheritance. This allows for a relatively simple API, but also affords extension without modification of the core library. For example, readers and writers of third party model attribute data can be written without any library modifications, generic geometry types can be used through the templated interface, and any serializable expression types can be used through type erasure.

Features

This section will describe some of the major features of `model-traits`. The goal of the software design is to maintain a simple to use API while providing user extensibility. This is accomplished by using only two conceptual types of data and two means of interaction. Thinking of the model attributes as being stored in a directed acyclic graph (DAG), this means there are two types of nodes and two graph representations. The nodes of the graph can either be a category, or a model trait (attribute). Category nodes can have a name (string), a type (string), and point to other categories and model traits. For example, an analysis may have multiple cases. Each case is represented by a category node where the type is "case" and the name is anything that is useful for the analyst such as "uniaxial extension." Model traits appear at the leaves of the graph and are the data that is associated with a given geometric type. These two node types can be seen in figure Figure 1 and Figure 2.

The first graph representation, also called the unassociated graph, is optimized for building a model where it is important to be able to add a single attribute definition to multiple geometric entities at a time Figure 1. The associated representation of the graph is optimized for geometric entity lookup which is typically done during the setup phase of an analysis case to be executed by a selected code. Figure 2.

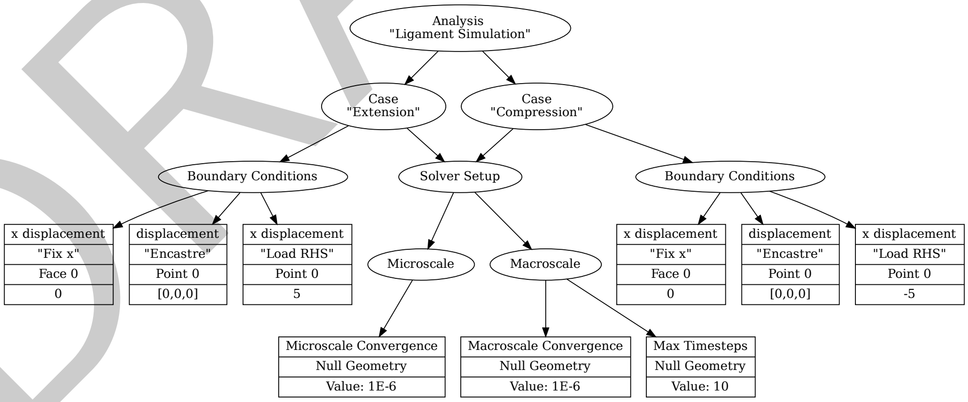

Figure 1: Unassociated graph representation of an example multiscale finite element simulation. Node types are listed on all nodes. If a node has an optional name it is surrounded in quotations.

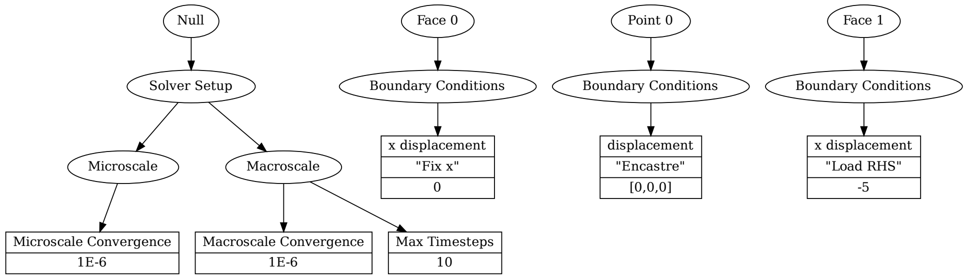

Figure 2: Associated Graph representation of the "uniaxial compression" analysis case. Node types are listed on all nodes. If a node has an optional name it is surrounded in quotations.

The library provides a YAML reader and writer and a smd (Simmetrix native attribute file) reader. The implementations of these readers and writer use the extensible IO interface and can be used as an example for alternative IO formats. To add an IO format the user only needs to provide template specializations for the `Read` and `Write` functions which are visible in the `mt` namespace. Additionally, `model-traits` supports the use of arbitrary geometric entity types which do not require wrapper classes or any class inheritance.

During the design phase a trade off between having an open set of IO backends or an open set of model trait types and shapes had to be made. Based on the authors modeling experience, there is a finite set of model trait types and shapes that are used in practice, whereas there is an unbounded set of analysis code input file formats. Therefore, the following data shapes are supported: scalar, dynamic sized vector, and dynamic sized dense matrix. Each of these shapes can have any of the following underlying types: double or float, boolean, integer, string, null, and expressions.

Expressions can be any invocable type that returns a double and takes up to 4 double arguments and can be converted to a string through `to_string` for serialization. This interface is enabled by the `NamedFunction` class which uses type erasure (similar to `std::function`) to provide a single interface type which does not require inheritance. It is constructible by any invocable which can be converted to a string through `to_string`. Additional constructors are provided which can be used to name an invocable which does not provide a `to_string` method such as a lambda or function pointer. For string based expressions the exprtk library is used for parsing and evaluation (Partow, 2021). The `ExprtkFunction` class is a functor that wraps the exprtk interface and is compatible with `NamedFunction`.

Acknowledgments

This work was supported in part by the National Institutes of Health (NIH) through Grand No. U01 AT010326-06. This material is based in part upon work supported by the National Science Foundation Graduate Research Fellowship under Grant No. DGE-1744655.

References

Beall, M. W., & Shephard, M. S. (1997). A general topology-based mesh data structure. *International Journal for Numerical Methods in Engineering*, *40*(9), 1573–1596. https://doi.org/10.1002/(SICI)1097-0207(19970515)40:9%3C1573::AID-NME128%3E3.0.CO;2-9

Hallquist, J. O. (2006). *LS-DYNA Theory Manual - March 2006*. Livermore Software Technology Corperation.

Ibanez, D. A., Seol, E. S., Smith, C. W., & Shephard, M. S. (2016). PUMI: Parallel Unstructured Mesh Infrastructure. *ACM Transactions on Mathematical Software*, *42*(3), 1–28. https://doi.org/10.1145/2814935

O'Bara, R. (2015). *Computational model builder for multi-dimensional models*. KITWARE INC CLIFTON PARK NY.

O'bara, R. M., Beall, M. W., & Shephard, M. S. (2002). Attribute management system for engineering analysis. *Engineering with Computers*, *18*(4), 339–351. https://doi.org/10.1007/s003660200030

O'Leary, P. (2014). *Open-source integrated design-analysis environment for nuclear energy advanced modeling & simulation phase I final report*. Kitware, Inc., Clifton Park, NY (United States).

Partow, A. (2021). *Exprtk*. https://github.com/ArashPartow/exprtk

Simmetrix Inc. - Mesh Generation, Geometry Access. (n.d.). Retrieved December 7, 2018, from http://simmetrix.com/

Smith, M. (2009). *Abaqus Theory Guide, Version 6.9*. Simulia.