The Journal of Open Source Software

The MOOSE Thermal Hydraulics Module

Joshua Hansel [1], **David Andrs** [1], **Lise Charlot** [1], and **Guillaume Giudicelli** [1]

1 Idaho National Laboratory

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Kyle Niemeyer

Reviewers:

- @atismer
- @powersrobert

Submitted: 30 October 2023
Published: unpublished

Summary

The Multiphysics Object-Oriented Simulation Environment (MOOSE) is an open-source object-oriented finite element framework written in C++ (A. D. Lindsay et al., 2022). The Thermal Hydraulics Module (THM) is an optional MOOSE physics module that provides capabilities for studying thermal hydraulic systems. Its core capability lies in assembling a network of coupled components, for instance, pipes, junctions, valves, etc.

THM provides several new systems to MOOSE to enable and facilitate thermal hydraulic simulations, most notably the `Components` system, which provides a higher-level syntax to MOOSE's lower-level objects. This system is extensible by the user, but the current library primarily includes components based on a 1D, single-phase, variable-area, compressible flow model, as well as heat conduction.

Statement of need

Numerous engineering applications employ fluids as mediums for transferring heat. Power generation applications notably transfer heat from a source, such as a boiler, to an energy conversion system, such as a turbine. Thermal hydraulic systems are well-suited for the task, due to their ability to move energy efficiently over the scale of a plant. These systems vary widely in their size and complexity and may feature a large number of components coupled together.

A notable example requiring thermal hydraulic systems analysis is a nuclear reactor system. These systems typically involve a large network of components to perform the conversion of nuclear power to electrical power and additional heat transfer components to remove the heat during accidental transients. For example, there may be a primary flow "loop" of pipes that extract heat from the fuel, pumps to force circulation, one or more heat exchangers exchanging heat between this primary loop and a secondary loop, turbomachinery components like turbines and generators, etc. Accurate systems analyses require models that capture the coupling between all of these components.

A wide variety of applications have been built using the MOOSE framework. A suite of applications supporting various domains of nuclear reactor systems analysis is under active development at Idaho National Laboratory. THM provides a foundation for thermal hydraulic applications in this area, including RELAP-7 (Berry et al., 2016), which models two-phase flow in light-water reactors and coolant flow in gas-cooled reactors, and Sockeye (Hansel et al., 2021), which models heat pipes used in nuclear microreactors.

Core capabilities

Components system

The `Components` system allows users to add "components", which are very flexible in their use, but in general represent "pieces" of a simulation, which may be coupled together. Common uses for components include adding meshes (1D, 2D, or 3D), variables, equations, and output. Components provide a higher level syntax that hides lower level MOOSE objects, such as `Kernels`, `BoundaryConditions`, etc. While `Actions` can also be used to create a higher level syntax, components provide much more convenience, particularly when multiple components interact.

Usually, components represent physical pieces in a system, such as pipes, solid bodies, or junctions; however, they can also be abstract, for example, coupling other components together, providing some source or boundary conditions, or just adding any other MOOSE objects in a convenient manner.

The `Components` system is abstract and provides several base classes that could be utilized by a variety of physical applications. All components share the base class `Component`; some intermediate base classes are:

- `Component1D`: generates a 1D mesh in 3D space, defined by a starting point, direction, length, and discretization.
- `Component2D`: generates a 2D mesh in 3D space, defined by the same axial parameters as `Component1D`, plus transverse direction, length, and discretization. This may represent either a Cartesian or axisymmetric coordinate system.
- `FileMeshComponent`: generates a mesh read from an external file (any dimension).
- `Component1DBoundary`: used for applying boundary conditions to a `Component1D` component.
- `Component1DJunction`: used for applying coupled boundary conditions between `Component1D` components.

The following subsections describe some of the currently available components.

Flow components

THM has numerous components that support a 1D, variable-area, single-phase compressible flow model (Berry et al., 2016). Components related to this flow model use the suffix `1Phase`, with the main component being the straight-channel component called `FlowChannel1Phase`. Other components related to this flow model are summarized as follows:

- Boundary conditions, such as inlets and outlets (with various formulations) and walls.
- Junctions, which allow flow to occur between multiple flow channels by providing coupled boundary conditions to each connected channel. These also include valves, which can partially or completely close flow paths.
- Volumetric heat sources, such as heat from a provided function, a convection condition, or a coupled heat flux.
- Volumetric form loss sources, such as those arising from flow blockages.
- Turbomachinery components, such as pumps, compressors, and turbines, which are particular types of junction that add source terms to the momentum and energy equations to simulate turbomachinery.

In addition to these components, there is also `FlowComponentNS`, which leverages a selection of flow formulations from MOOSE's Navier-Stokes module (A. Lindsay et al., 2023), with a mesh provided by an external file.

The fluid properties needed by these components are defined using the MOOSE Fluid Properties module. For simulations with multiple loops with different fluids, a `FluidProperties` object can be created for each fluid and used in the corresponding THM flow components.

The Journal of Open Source Software

Heat conduction components

Thermal hydraulic systems feature not only flow components but also solid bodies that transfer heat with the flow, such as the walls of a heat exchanger. In THM, these bodies are 2D or 3D components referred to as "heat structures," which solve the transient heat conduction equation:

- `HeatStructurePlate`: derived from `Component2D` and using a Cartesian coordinate system, representing a "plate" geometry.
- `HeatStructureCylindrical`: derived from `Component2D` and using an axisymmetric coordinate system, representing a "cylinder" or "shell" geometry.
- `HeatStructureFromFile3D`: derived from `FileMeshComponent`, representing a general 3D geometry.

In addition to the heat structure components themselves, there are components that interact with them:

- Boundary conditions, such as Dirichlet, provided heat flux function, convection, and radiation.
- Volumetric heat sources.
- Interface conditions, which couple heat structures to other heat structures or to flow channels.

`Closures` system

The `Closures` system allows users to create MOOSE objects (usually `Materials`) that specify closures for their component models. For example, a flow channel may require definitions of quantities such as friction factors and heat transfer coefficients. The selection of closure relations may depend on the application. Closure relations may be generic correlations or custom relations, such as correlations from experimentally measured data.

While these definitions could be made inside a component, it is advantageous to have closure definitions separately both for code-reuse purposes and to avoid duplicating objects. There may be a large number of closure choices, each with their own user parameters. In large systems of components, `Closures` objects can be reused when the closures apply to many objects.

`ControlLogic` system

The `ControlLogic` system is an extension of MOOSE's `Controls` system, which is used to control input parameters to various objects during a simulation. Unlike standard controls objects, control logic objects may declare new control data that is not associated with input parameters and may retrieve control data declared in other control logic objects, allowing control operations to be chained together, which is not possible in the standard `Controls` system. This is necessary to mirror real control systems in thermal hydraulic systems, which may feature various controllers in series. Examples of controls in THM's library include a transient function control, a proportional-integral-derivative control, a delay control, a trip control, and a termination control.

Integrity checking

Systems simulations can have a very large number of components, and the input for such large models is potentially complex and error-prone, perhaps involving tens or hundreds of input errors in a user's first attempt. To enable a practical input file-writing workflow, it is important that errors can be reported in batch, instead of stopping execution at the first error encountered, which is the usual behavior in MOOSE. To address this need, THM provides a logging capability. An application creates a single "logger" object, and then various objects (such as components and closures objects) can log errors and warnings. Execution continues until the application chooses to print out all of the errors and warnings, in a very palatable, condensed format. Then the user can view and address these errors all at once, significantly decreasing the number of input file iterations.

Documentation

THM's documentation is extensive and follows the same structure as the code base. Each object is accompanied by a documentation page that:

- Describes it, including its equations or a figure as relevant
- Lists its parameters along with metadata about the parameters
- Lists all the input files in the repository that use this object
- Lists all the classes deriving from this object.

Major groups of objects, usually derived from a single base class, are documented through the syntax documentation, which describes how these objects are instantiated and used in a simulation. For example, `Components` and `Closures` are examples of syntax unique to THM that also correspond to base classes of groups of THM objects.

This documentation page is hosted on the module website (Idaho National Laboratory, 2023). The website also notably hosts the software quality assurance (SQA) records, such as the testing requirement matrix or failure analysis reports for example. The interested reader is referred to the MOOSE SQA plan (*Software Quality Assurance Plan for MOOSE and MOOSE-Based Applications (PLN-4005)*, 2020) for more information.

Testing

The module relies on CIVET (Slaughter et al., 2021) for continuous integration. Every pull request to the module runs the entire test suite for MOOSE, including the tests for THM. The test suite is comprehensive and aims to cover every feature available in the module. It consists of unit and regression tests, described in this section. The entire test suite is run with a wide variety of configurations, from compiling with the oldest and newest supported compilers, to running with several shared memory threads and distributed memory processes, on a variety of operating systems. The code is also checked for memory leaks using Valgrind (Nethercote & Seward, 2007).

Unit tests in THM are targeted at specific routines that can be accessed by creating the relevant object with example parameters. For example, in a `Flux` object, we can check that the formulation of the numerical flux is both consistent and symmetric.

Regression tests are typically created for every object to ensure that their behavior does not vary on a relevant test case. For example, a `Function` object can be tested on a simple mesh to ensure the field it produces is consistent. Components are often tested in the minimal configuration sufficient to satisfy the test requirement, for example, to prove conservation of mass and energy on a flow channel.

In addition to the automated testing provided by CIVET, proposed changes to the module are reviewed by at least one member of the MOOSE change control board, as detailed in MOOSE's SQA plan (*Software Quality Assurance Plan for MOOSE and MOOSE-Based Applications (PLN-4005)*, 2020), in addition to any other interested reviewers. Reviewers determine if the proposed changes have an acceptable design, follow coding standards, and are sufficiently tested.

Demonstration

THM can be used as a foundation for other MOOSE-based applications or as a stand-alone software to model a variety of systems, including nuclear systems, power conversion cycles, and

The Journal of Open Source Software

geothermal piping networks. The flexibility of THM is demonstrated with a two-loop system that is typical of a nuclear reactor system. This model is the final step of the single-phase flow THM tutorial available on the THM website (Idaho National Laboratory, 2023).

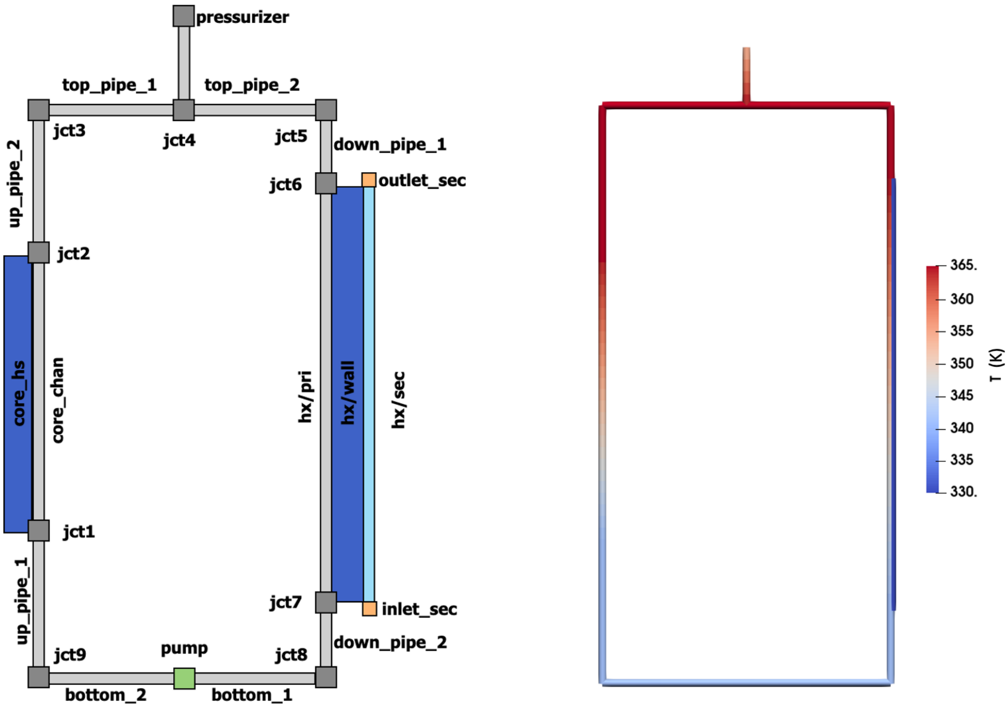

Figure 1: System diagram (left) and temperature distribution (right)

The system is shown in Figure 1. Helium circulates in the primary loop (displayed in light gray color) and extracts heat from a rod (`core_hs`). Heat is transferred to liquid water in a secondary system (displayed in light blue color) through a heat exchanger (`hx/*`). The cold helium then enters a pump, before flowing into the heated section again. Each part of the system shown in Figure 1 is defined using the appropriate `Component` object. The `ControlLogic` system is used to set the pump head to match a target mass flow rate in the primary loop. The secondary side is a flow channel with water. This example features two sets of `Closures` objects. The first set is of type `Closures1PhaseTHM` and uses classic engineering correlations for the heat transfer coefficient and friction factor; the second set is of type `Closures1PhaseNone`, used to define custom relations for the various closures for the primary side of the heat exchanger. THM can then calculate quantities of interest, such as pressure, temperature, and mass flow rate.

Conclusions

THM provides a flexible framework for performing thermal hydraulic systems simulations within the MOOSE framework, with many useful capabilities that extend beyond the field of thermal hydraulics. The `Components` system provides an ideal structure for setting up large systems of connected components in MOOSE, significantly reducing the user input burden, since the higher level components syntax hides the multitude of lower level MOOSE objects. The `ControlLogic` system extends the usability of MOOSE's `Controls` system, allowing control units to be chained together.

The development of the module is driven by user application needs. Future work to THM may include improvement of existing components, as well as additional components related to

The Journal of Open Source Software

single-phase flow and heat conduction. Depending on future needs, additional flow models may be added as well. Further abstraction of the way the discretization of an equation is created on a component is underway, and should allow for the definition of a general multiphysics component, able to instantiate any equation discretized in MOOSE.

External contributions to the module are encouraged, and support will be provided to comply with MOOSE's SQA standard.

Acknowledgements

We would first like to thank Richard Martineau for his strong support and efforts to acquire funding for the project. We would also like to acknowledge the time and effort of THM's many contributors, most notably Jack Cavaluzzi, Thomas Freyman, Luiz Aldeia, and Rachel Beall.

This work was funded by the Department of Energy Nuclear Energy Advanced Modeling and Simulation (NEAMS) program. This manuscript has been authored by Battelle Energy Alliance, LLC under Contract No. DE-AC07-05ID14517 with the US Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a nonexclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes.

References

Berry, R. A., Zou, L., Zhao, H., Zhang, H., Peterson, J. W., Martineau, R. C., Kadioglu, S. Y., & Andrs, D. (2016). *RELAP-7 theory manual* (INL/EXT-14-31366). Idaho National Laboratory.

Hansel, J. E., Berry, R. A., Andrs, D., Kunick, M. S., & Martineau, R. C. (2021). Sockeye: A one-dimensional, two-phase, compressible flow heat pipe application. *Nuclear Technology*, *207*(7), 1096–1117. https://doi.org/10.1080/00295450.2020.1861879

Idaho National Laboratory. (2023). *Thermal hydraulics module*. https://mooseframework.inl.gov/modules/thermal_hydraulics/index.html

Lindsay, A. D., Gaston, D. R., Permann, C. J., Miller, J. M., Andrš, D., Slaughter, A. E., Kong, F., Hansel, J., Carlsen, R. W., Icenhour, C., Harbour, L., Giudicelli, G. L., Stogner, R. H., German, P., Badger, J., Biswas, S., Chapuis, L., Green, C., Hales, J., … Wong, C. (2022). 2.0 - MOOSE: Enabling massively parallel multiphysics simulation. *SoftwareX*, *20*, 101202. https://doi.org/https://doi.org/10.1016/j.softx.2022.101202

Lindsay, A., Giudicelli, G., German, P., Peterson, J., Wang, Y., Freile, R., Andrs, D., Balestra, P., Tano, M., Hu, R., & others. (2023). MOOSE navier–stokes module. *SoftwareX*, *23*, 101503.

Nethercote, N., & Seward, J. (2007). Valgrind: A framework for heavyweight dynamic binary instrumentation. *SIGPLAN Not.*, *42*(6), 89–100. https://doi.org/10.1145/1273442.1250746

Slaughter, A. E., J.Permann, C., Miller, J. M., Alger, B. K., & Novascone, S. R. (2021). Continuous integration, in-code documentation, and automation for nuclear quality assurance conformance. *Nuclear Technology*, *0*(0), 1–8. https://doi.org/10.1080/00295450.2020.1826804

Software quality assurance plan for MOOSE and MOOSE-based applications (PLN-4005). (2020). Idaho National Laboratory.