ORNL/TM-2015/530

An Update on NiCE Support for BISON (MS-15OR0401031)

Alexander J. McCaskey
Jay Jay Billings
Jordan Deyton
Anna Wojtowicz

September 2015

OAK RIDGE NATIONAL LABORATORY
MANAGED BY UT-BATTELLE FOR THE US DEPARTMENT OF ENERGY

ORNL/TM-2015/530

Computer Science and Mathematics Division

An Update on NiCE Support for BISON (MS-15OR0401031)

Alexander J. McCaskey
Jay Jay Billings
Jordan Deyton
Anna Wojtowicz

Date Published: September 2015

Prepared by
OAK RIDGE NATIONAL LABORATORY
P.O. Box 2008
Oak Ridge, Tennessee 37831-6285
managed by
UT-Battelle, LLC
for the
US DEPARTMENT OF ENERGY
under contract DE-AC05-00OR22725

CONTENTS

LIST OF FIGURES

EXECUTIVE SUMMARY

The Nuclear Energy Advanced Modeling and Simulation program (NEAMS) from the Department of Energy's Office of Nuclear Energy has funded the development of a modeling and simulation workflow environment to support the various codes in its nuclear energy scientific computing toolkit. This NEAMS Integrated Computational Environment (NiCE) provides extensible tools and services that enable efficient code execution, input generation, pre-processing visualizations, and post-simulation data analysis and visualization for a large portion of the NEAMS Toolkit. A strong focus for the NiCE development team throughout FY 2015 has been support for the Multiphysics Object Oriented Simulation Environment (MOOSE) and the NEAMS nuclear fuel performance modeling application built on that environment, BISON. There is a strong desire in the program to enable and facilitate the use of BISON throughout nuclear energy research and industry. A primary result of this desire is the need for strong support for BISON in NiCE.

This report will detail improvements to NiCE support for BISON. We will present a new and improved interface for interacting with BISON simulations in a variety of ways: (1) improved input model generation, (2) embedded mesh and solution data visualizations, and (3) local and remote BISON simulation launch. We will also show how NiCE has been extended to provide support for BISON code development.

1. Introduction

BISON is an engineering-scale nuclear fuel performance application built on top of the Multiphysics Object Oriented Simulation Environment (MOOSE) framework [2]. MOOSE is a parallel, finite-element framework developed at Idaho National Laboratory (INL) that enables the quick and efficient development of coupled, multiphysics applications [1]. The NEAMS Integrated Computational Environment (NiCE) provides mature support for MOOSE-based applications by efficiently enabling activities such as input generation, simulation execution, and data visualization, as a core part of the platform. As a MOOSE-based application, BISON immediately benefits from, and can leverage this strong support.

As of the summer of 2015, that support has been greatly updated to incorporate comments from the MOOSE and BISON internal development teams, as well as other MOOSE application developers. The updates to NiCE support provide a new unified interface for interacting with MOOSE-based applications like BISON, as well as new tools for the embedded visualizations of problem meshes, simulation output meshes, and MOOSE Postprocessor XY plotting. Additionally, we have improved our BISON tooling to incorporate real-time updating from a running BISON simulation back to NiCE. Users can now specify a list of Postprocessor values they care about, launch their simulation, and see, in real-time, those Postprocessor plots updating.

This report will discuss in detail these new updates. It will detail a new NiCE installation mechanism, BISON input generation, existing BISON input file loading, simulation launch, and post-simulation data visualization. Additionally, we will show how NiCE now supports BISON application development.

1.1 NiCE Installation

The NiCE team has developed a new and improved mechanism for BISON users to install NiCE and its dependencies. To do so, users need to simply navigate to https://sourceforge.net/projects/niceproject/files and download the new *NiCE_installer.sh* script. To install NiCE, simply run this script in a directory of your choosing. For example:

```
mkdir ICE_install
cd ICE_install
wget https://sourceforge.net/projects/niceproject/files/ICE_installer.sh .
chmod +x ICE_installer.sh
./ICE_installer.sh
```

Running this script will pull down the correct version of NiCE, VisIt, and HDFJava for your operating system, unpack them, and configure NiCE to use the downloaded VisIt and HDFJava dependencies. For Mac users, this script will additionally configure NiCE to use the current user's environment variables and register those environments with NiCE through the Apple Application Launch Service. All of this will enable you to simply double click the created NiCE application icon (~/Applications/NiCE.app for Mac, and for Linux, an NiCE application icon in the Programming category of the applications menu).

For Windows users, we also provide MSI installers at the same URL. Just download the installer specific to your architecture, run it, and you'll be ready to use NiCE.

1.2 MOOSE Perspective

NiCE comes bundled with a *MOOSE Perspective* that organizes the various views and windows that a MOOSE user or developer would need to efficiently work with MOOSE in NiCE. This perspective is

perfect for a typical BISON user, as it shows the Item Viewer, Resources View, Project Explorer, MOOSE Data Tree View, and Properties tab to the user to enable efficient interaction with the framework and BISON.

To access the *MOOSE Perspective*, use the the NiCE toolbar at the top and navigate to *Window* > *Open Perspective* > *Other...* Select *MOOSE* in the window that pops up and click *OK*. Alternatively, you can also access the same pop-up menu by clicking the *Open Perspective* button in the upper right-hand corner of the NiCE workbench. Once the MOOSE Perspective opens, you should notice the workbench now contains

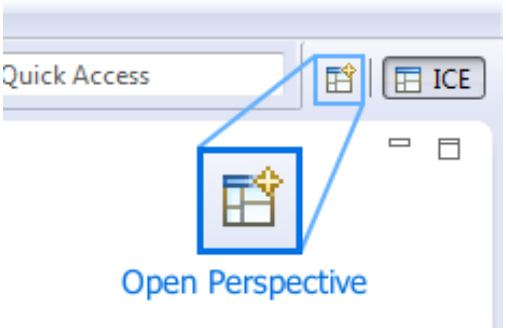

Fig. 1. The NiCE Open Perspective button will let NiCE users select a perspective to show, like the custom MOOSE perspective.

fewer UI components. Furthermore, you can arrange these views in any manner you wish, simply grab the tab for the view and move it around as you see fit.

2. Connecting to VisIt

To view embedded visualizations within the NiCE BISON tools, you must specify a connection to one of NiCE's Visualization Services, such as VisIt (limited Paraview support is also available). In order to do so, simply open the Preferences menu item in the top toolbar (on Mac, *Eclipse NiCE* > *Preferences*, on Linux, *Window* > *Preferences*). In the Preferences wizard, navigate to the *Visualization* Preference node in the left tree view, open it, and select the *VisIt* node (*Preferences* > *Visualization* > *VisIt*). In that preferences view, simply add a connection to VisIt by clicking the add button and specifying the path to the VisIt executable.

3. The MOOSE Workflow Item

Previous versions of MOOSE and BISON support in NiCE presented the MOOSE user with two separate Items for interaction with MOOSE: the MOOSE Model Builder and the MOOSE Launcher. The NiCE team has done away with that separation and now allows users to construct one MOOSE Workflow Item, which presents a unified interface for MOOSE simulation development and execution in a more efficient manner. To create a new MOOSE Workflow Item simply click the green "+" button in the *Item Viewer* (Figure 2), located on the left-hand side of the NiCE workbench and MOOSE perspective. This will prompt a window to pop up; select the *MOOSE Workflow* Item, and click *Finish* (Figure 3). Alternatively, if you'd like to import an existing `*.i` input file to modify, click the yellow item import arrow located at the top of the NiCE workbench (Figure 4). A wizard will pop up prompting you to specify two things: the

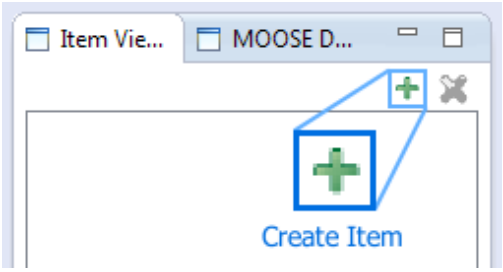

Fig. 2. Click the green plus button in the Item Viewer to create a new MOOSE Workflow Item.

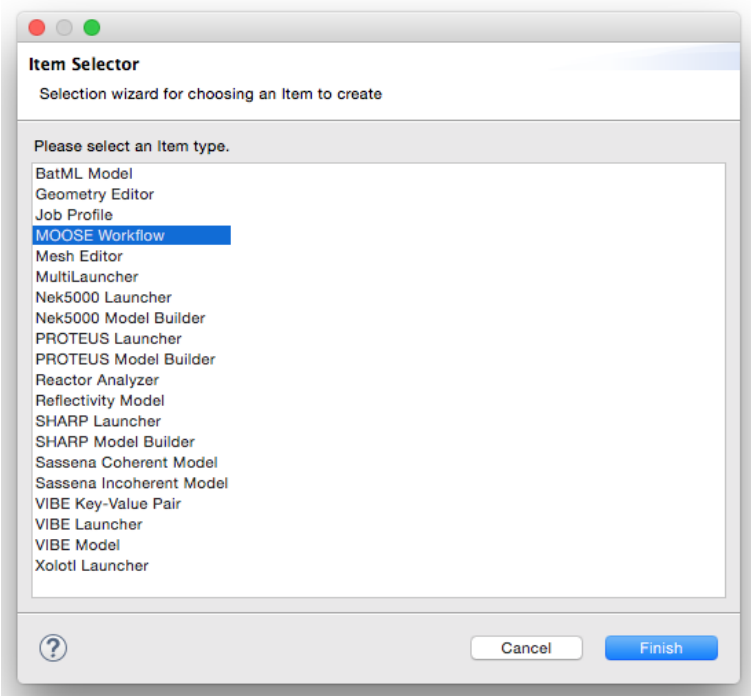

Fig. 3. The NiCE Item Selector wizard.

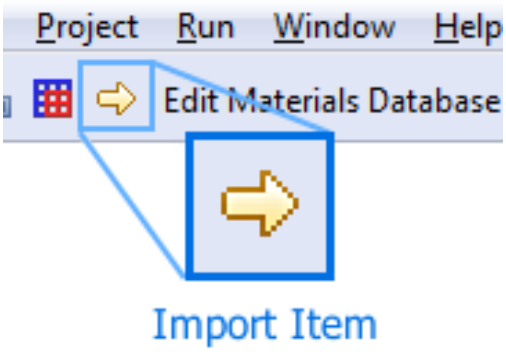

Fig. 4. Click the yellow arrow in the toolbar to import an existing NiCE Item.

`*.i` file you'd like to import from your filesystem, and what kind of *Item* you'd like to import it into. Browse to the file and select the *MOOSE Workflow* Item from the list.

Now let's take a look at the use of the MOOSE Item in each of these two cases.

3.1 Creating a new MOOSE Workflow Item

When you select the MOOSE Workflow Item for the first time, you'll be presented with the form shown in Figure 5. To begin using the MOOSE Workflow Item for BISON, you'll need to specify where

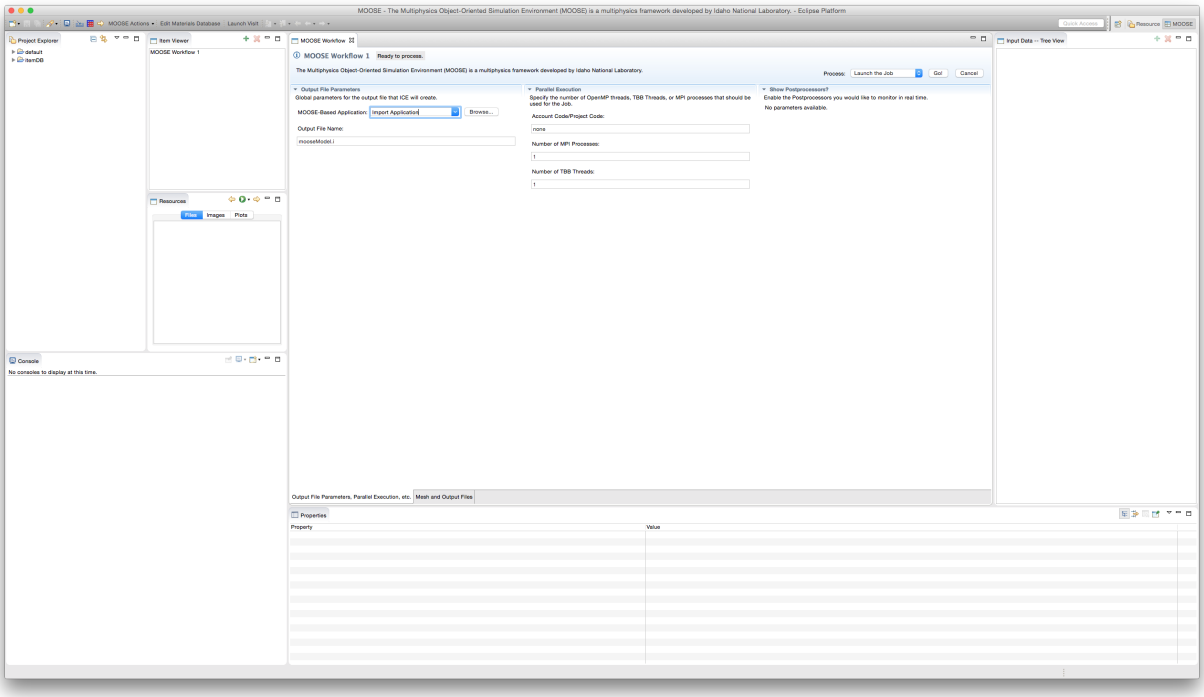

Fig. 5. A view of the updated MOOSE Workflow Item.

the BISON executable is located in the *MOOSE-based Application* Entry. As you can see in Figure 6, this is the first time we've used the MOOSE Workflow Item, so there are no available applications to select and NiCE is indicating that we must import one.

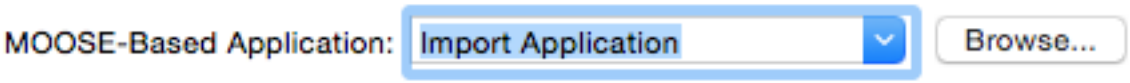

Fig. 6. The MOOSE-based application selection Entry.

This Entry allows the user to navigate to a BISON executable that is hosted either locally or remotely. Once the application has been specified, this Entry saves the value for the future, i.e. it persists between NiCE executions so that the user can reuse the same application again. To specify an application, click the *Browse* button. Upon doing so, you'll be presented with a dialog (Figure 7) to specify whether this application is hosted locally or remotely. Select local, for now, and you'll be presented with a File Dialog

(Figure 8), search for your application and click finish. With your application imported, the next step is to

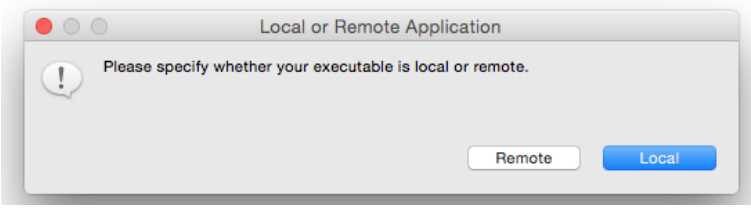

Fig. 7. This dialog is presented when you choose to browse to a new MOOSE application. The user can find a MOOSE application on the local machine or on some remote host.

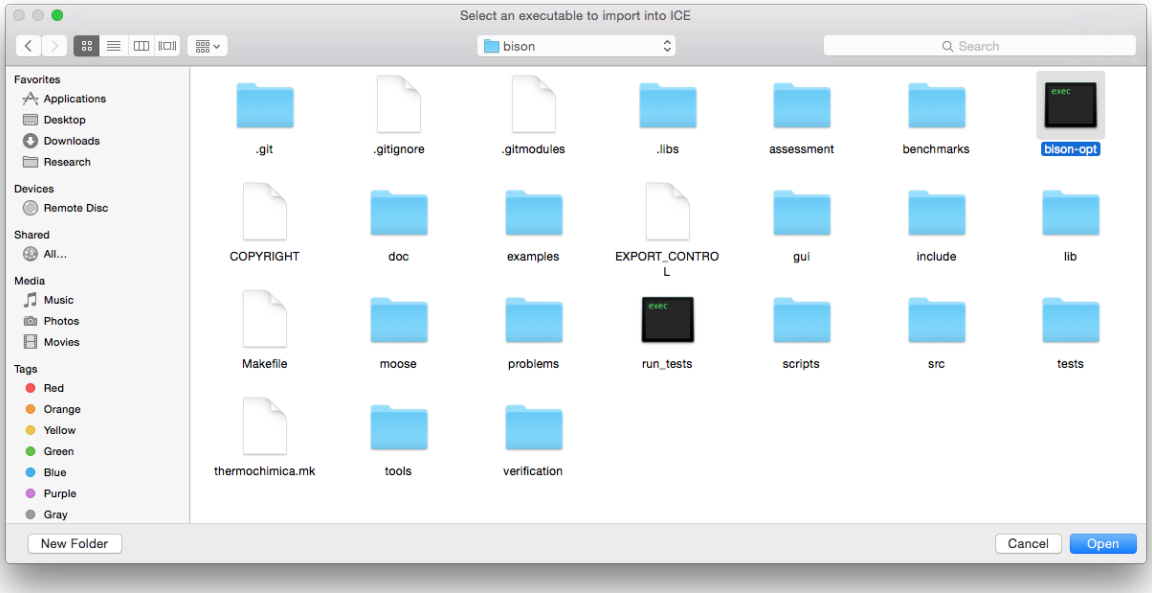

Fig. 8. A view of the browser for local MOOSE applications.

save the current MOOSE Workflow Item to trigger the generation of the BISON input tree. From this tree, you can start constructing your BISON input file. You can add children to the tree nodes, delete children, and set properties for each block in the Properties View.

3.2 Updates to the MOOSE Tree

The BISON tree has been improved and made *smarter*. Blocks that require a variable to be specified, such as the Kernels block, now only let users specify a variable that has been created in the Variables block. Adding or removing variables from the Variables block will update those blocks with variable entries to display only a list of the currently available variables (see Figure 10). Additionally, all file entries in the BISON tree will dynamically show up in the Output File Parameters Data component on the MOOSE Workflow form (see Figure 11).

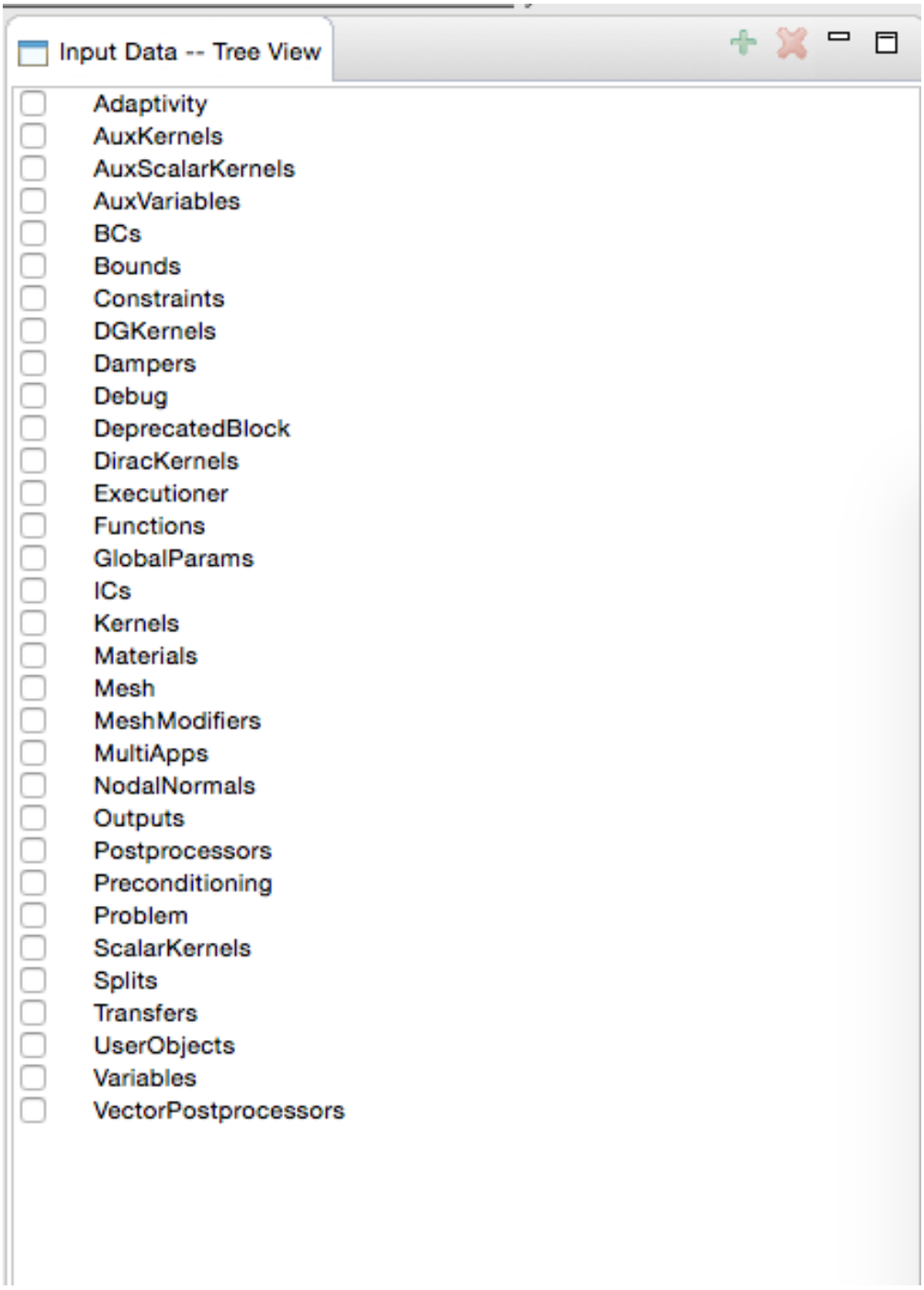

Fig. 9. A view of the MOOSE input tree.

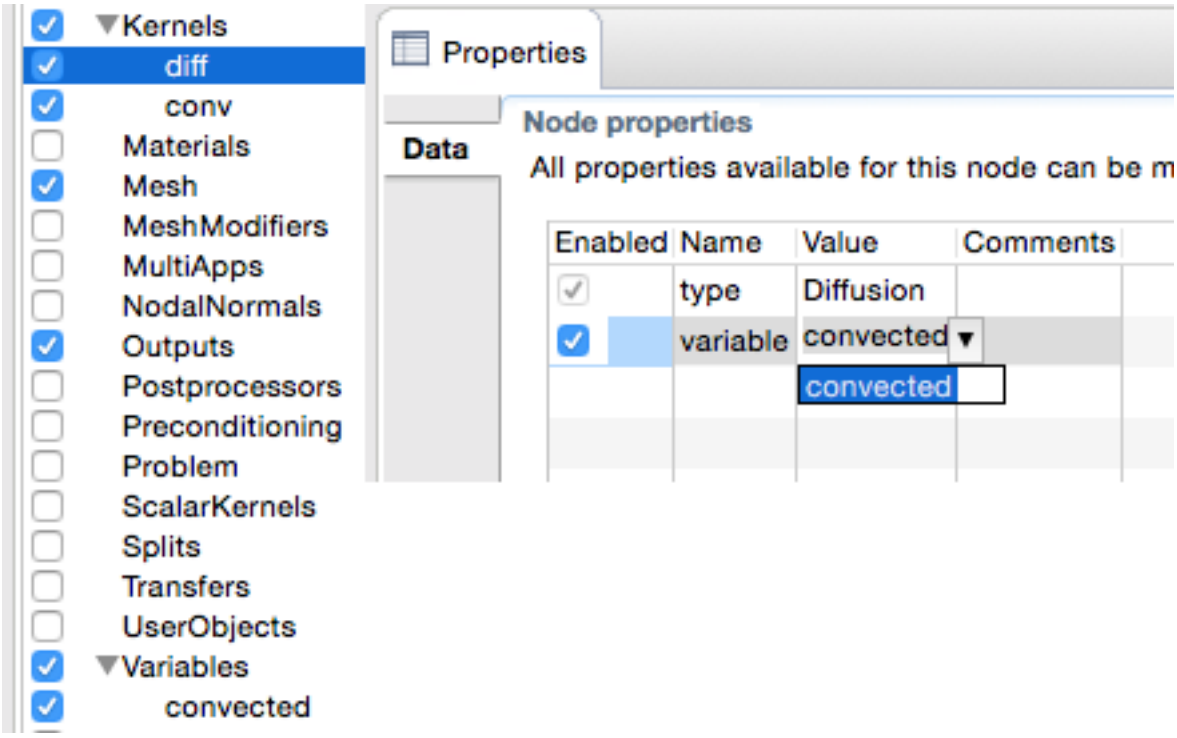

Fig. 10. A view of the input tree, showing the smart updates to Variable entries in the tree.

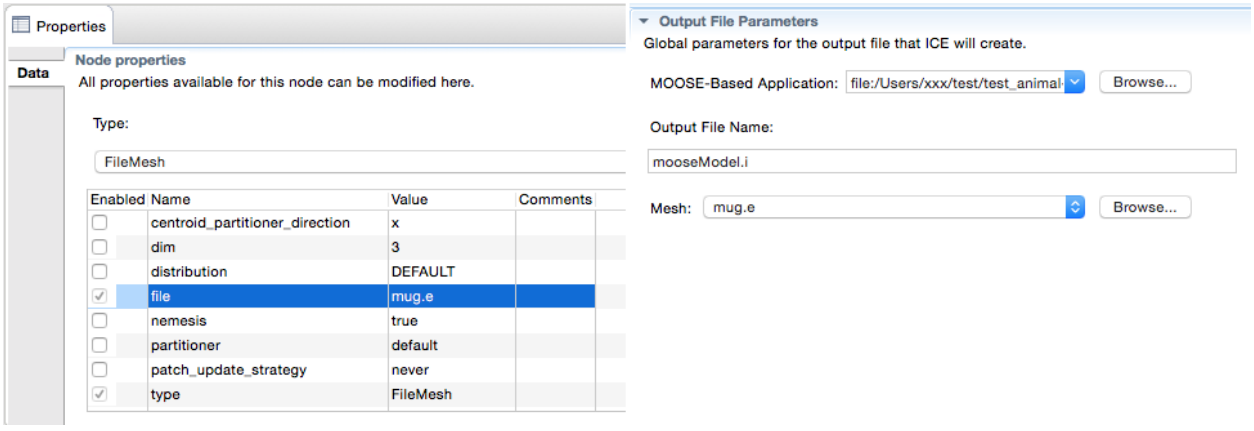

Fig. 11. A view of how file entries in the tree are tied to browser entries in the Form.

3.3 Importing an Existing MOOSE Workflow Item

As mentioned above, you can import an existing BISON input file as a new MOOSE Workflow Item. Clicking the yellow import arrow icon in the toolbar and selecting the input file and MOOSE Workflow Item will import a new MOOSE Workflow Item into the main form editor area in the MOOSE perspective. Its tree view will already be populated with the contents of the file. Next, select the application this MOOSE Workflow Item corresponds to and click save. This will merge the current tree corresponding to the input file with the application's YAML tree (see Figure 12).

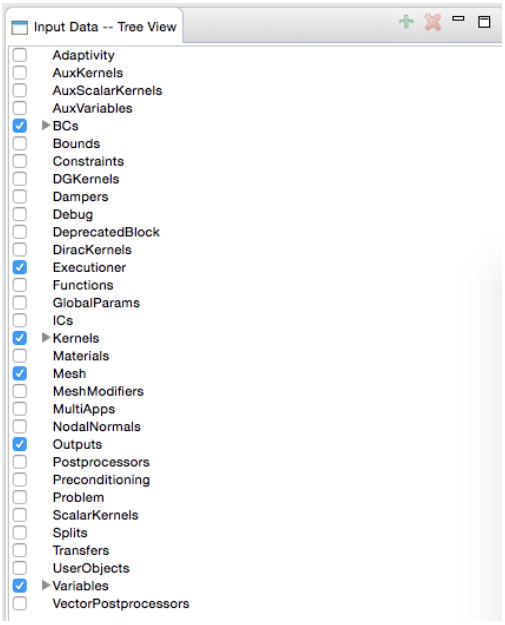

Fig. 12. A view of the input tree after an existing input file import.

3.4 Using a Remote BISON Application

NiCE let's users point to remotely hosted BISON executables just as easily as locally hosted ones. To use a remote application, simply select Remote on the Local/Remote dialog after clicking the Browse button on the MOOSE-Based Application entry. You will be presented with a remote connection dialog, shown in Figure 13. Fill out the New Connection wizard with the information needed to connect to the remote host. After filling this out, the remote file browser will load up and you can navigate to the remotely hosted application (Figure 14).

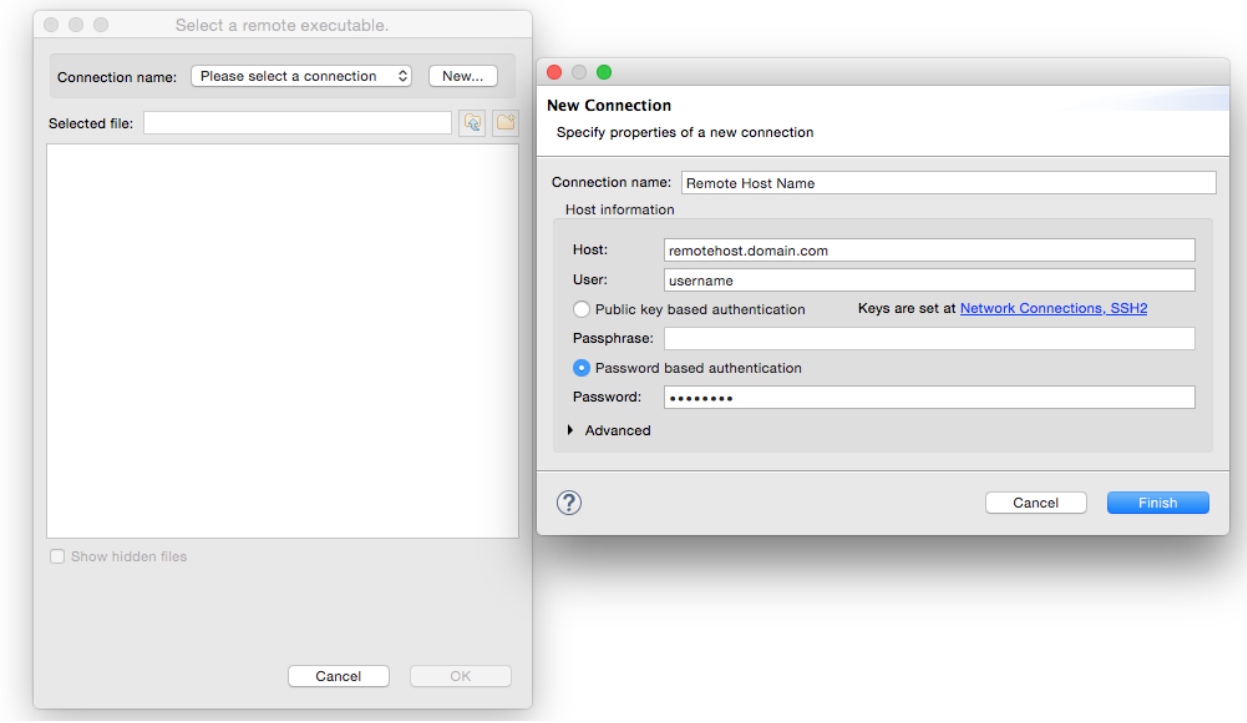

Fig. 13. Users can configure a new host for use with a remote MOOSE application.

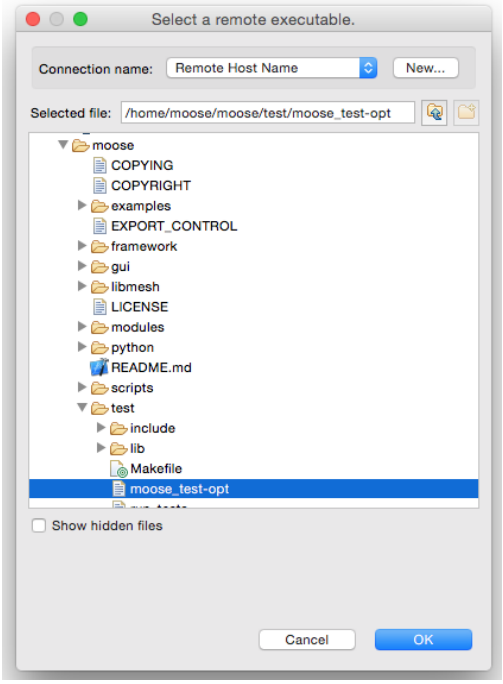

Fig. 14. Remote application selection.

3.5 Viewing the Mesh

NiCE now has support for embedded visualizations, and the BISON support in NiCE provides this embedded view to allow users to view the mesh for a given application run. To view the mesh for a given MOOSE Workflow Item and input tree, simply look in the Resources View for the mesh file and double click it (with a valid VisIt connection present) (see Figure 15). The mesh will open in an embedded view in the MOOSE Workflow Item's Mesh and Output Files tab, as seen in Figure 16. It is completely interactive, you can rotate the view and zoom in and out to get a good look at the mesh you are using for the current BISON application launch.

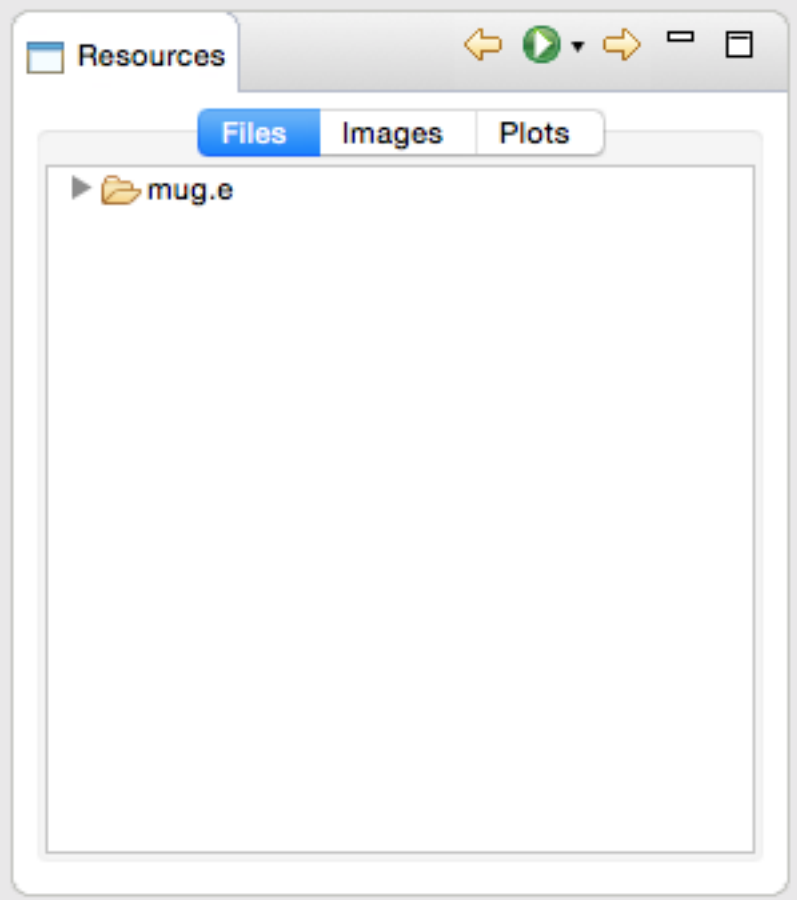

Fig. 15. When a mesh is input in the tree, it shows up in the Resources tab and can be double-clicked to view.

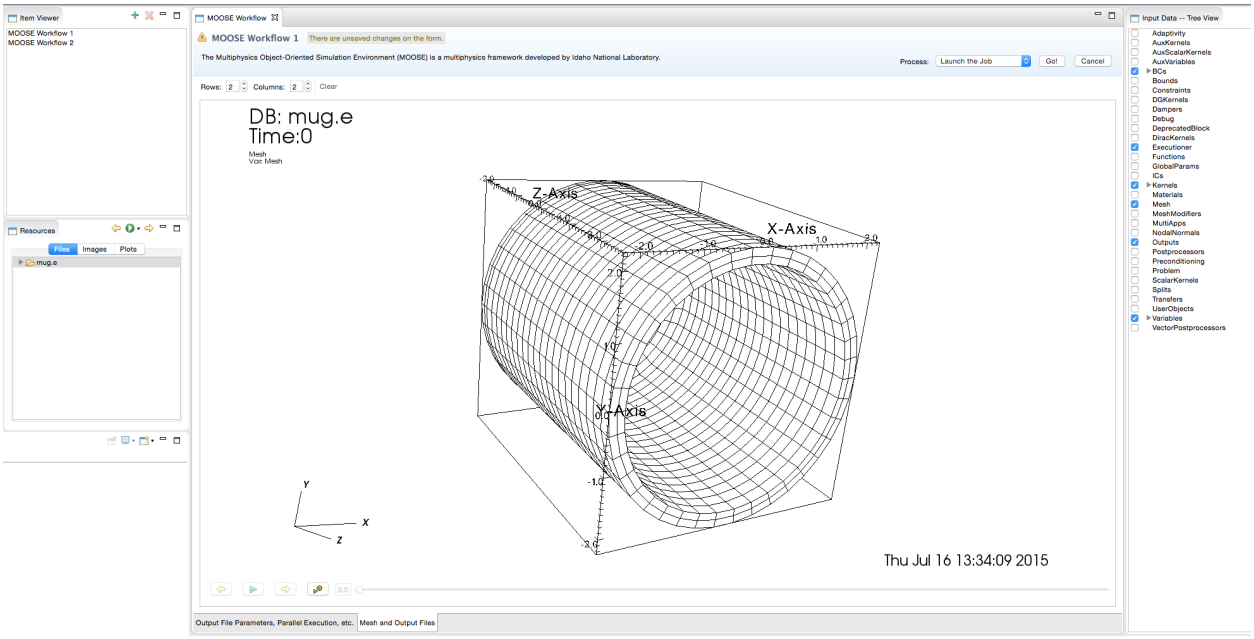

Fig. 16. A view of the embedded mesh view.

3.6 Launching the Application

Launching your BISON application is simple. If you selected a remotely hosted application, then NiCE handles the appropriate remote launch semantics (file uploads/downloads, remote invocation, etc). If you selected a local application, NiCE handles the local execution of your application in your workspace's default/jobs folder. BISON users can also specify the number of MPI processes and threads to use in the simulation execution, shown in Figure 17.

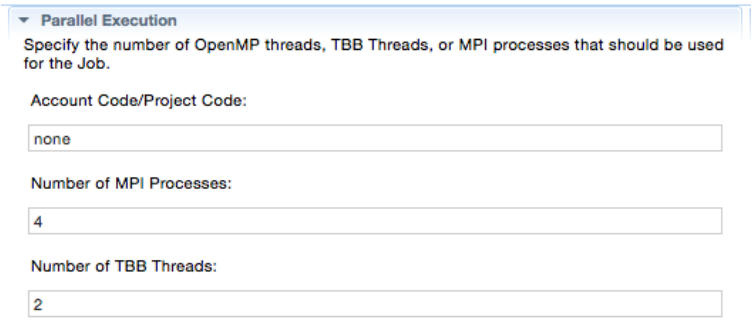

Fig. 17. Users can configure the launch to be parallel in both number of processes and threads.

To launch the application, simply click the Go button for the *Launch the Job* action in the top right of the MOOSE Workflow Item. This will execute the simulation and pipe the output to the Console view (Figure 18). When the simulation finishes, you'll notice the Resources view has been populated with a list of files generated by the simulation. These files include all Postprocessor CSV files and the simulation output Exodus file. To view them in the embedded visualization view in the Mesh and Outputs tab, simply

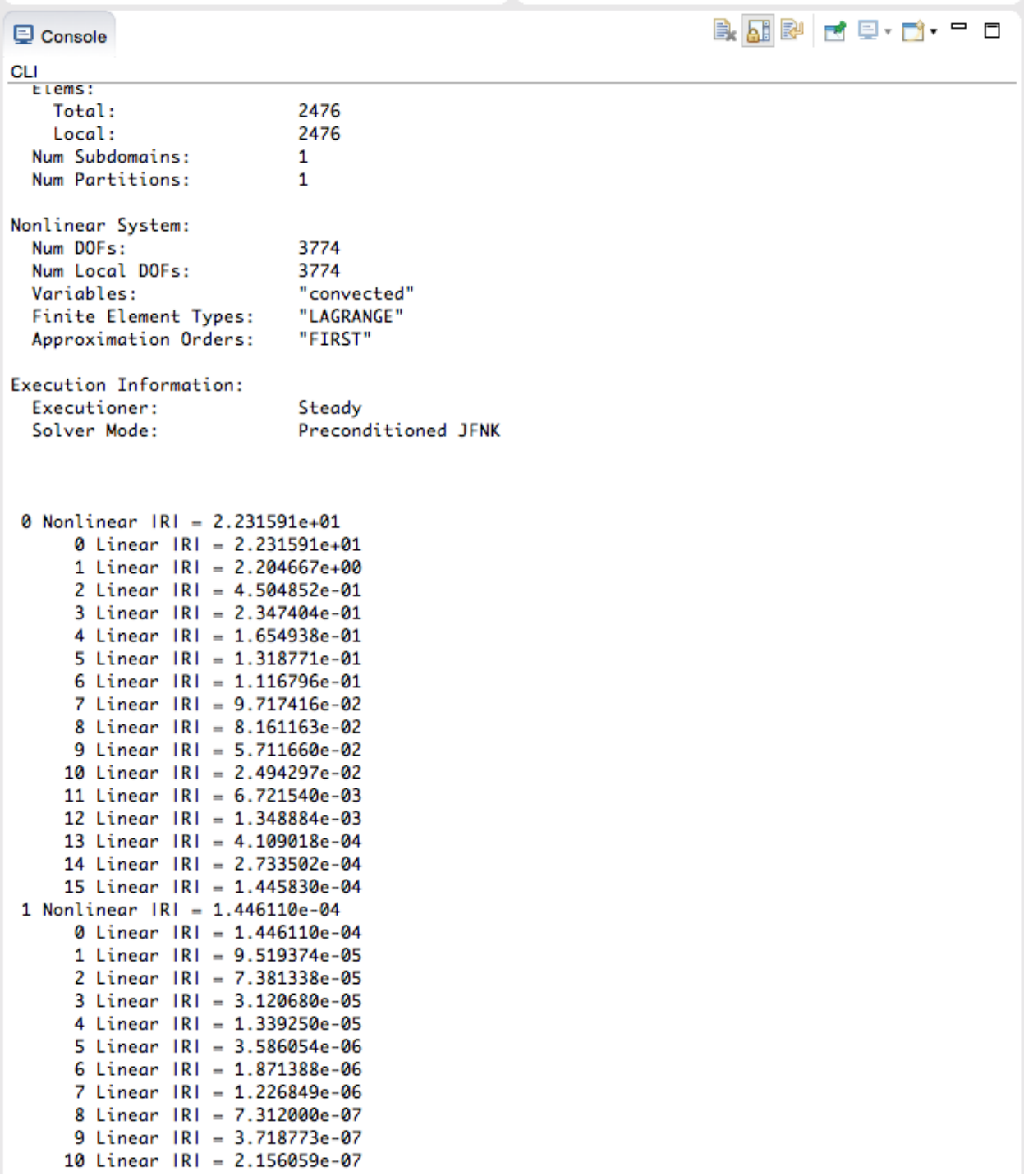

Fig. 18. The Console view showing the output for a MOOSE simulation.

double-click the file. For the output Exodus file, you can right click on the Plot to navigate a context menu of all possible plots to view. Simply select the solution field you're interested in viewing on the mesh, as shown in Figure 19.

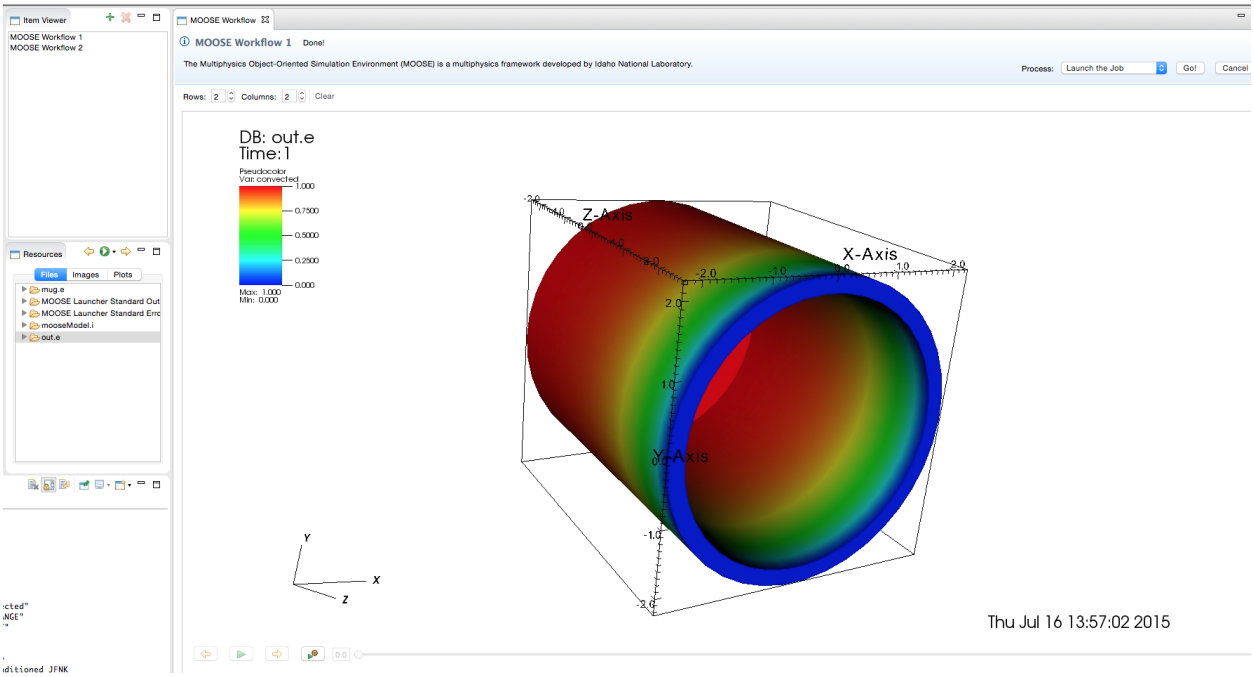

Fig. 19. A view of the embedded solution mesh visualization.

3.7 Real-time Updating for Postprocessor Values

NiCE and BISON can now communicate in real-time during a simulation execution over a web connection. Now, as a simulation is running, BISON can communicate Postprocessor values in real-time so that NiCE can display a dynamically updating plot of the Postprocessor during simulation execution. When the user selects the Launch the Job process, NiCE adds an *ICEUpdater* block to the Outputs block in the MOOSE tree that directs BISON and the underylying MOOSE framework to communicate with NiCE over a web connection during simulation execution. So, as the simulation is running, you can double-click a Postprocessor CSV file in the Resources view and view it populate in real-time. Or, if you'd like your plots to open immediately, you can select them in the Show Postprocessors component on the MOOSE form (Figure 20). If these Postprocessors are enabled, NiCE will immediately open up the Postprocessor plot when the simulation is executed (Figure 21).

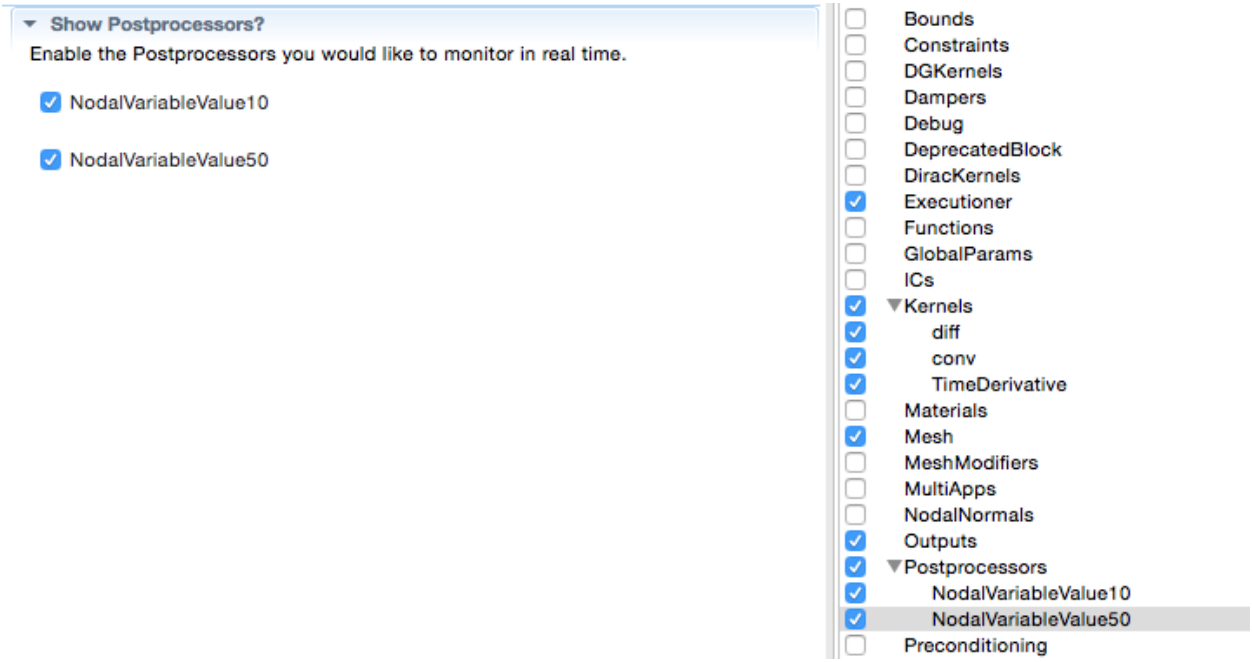

Fig. 20. A view of real-time Postprocessor plot display configuration.

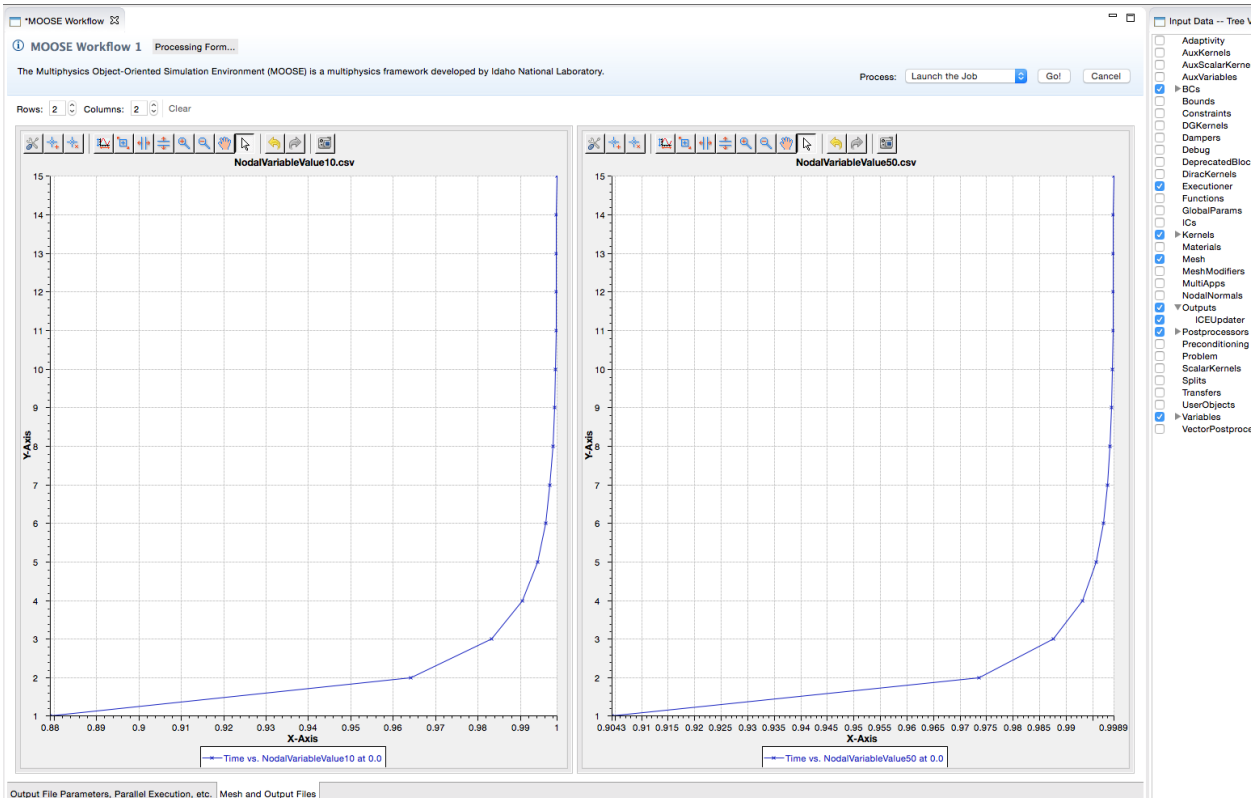

Fig. 21. Real-time plots for selected Postprocessors.

4. MOOSE-based Application Development in NiCE

Since NiCE is built on top of the Eclipse platform, a large variety of sophisticated software development tools and technologies for developing scientific software can be integrated into the NiCE platform. Version control, code editing, code completion, code building, and code generation are just a few of the various technologies now available to all MOOSE-based application developers using NiCE. BISON developers can take advantage of this support to improve their scientific computing workflow, and leverage the aforementioned MOOSE Workflow tools for interacting with the developed simulation code, all from the same application.

4.1 Cloning MOOSE

To clone MOOSE, simply switch to the Git Perspective in the top right corner of NiCE. You will be presented with the view in Figure 22. Now click the 'Clone a Git Repository' button in the toolbar of the

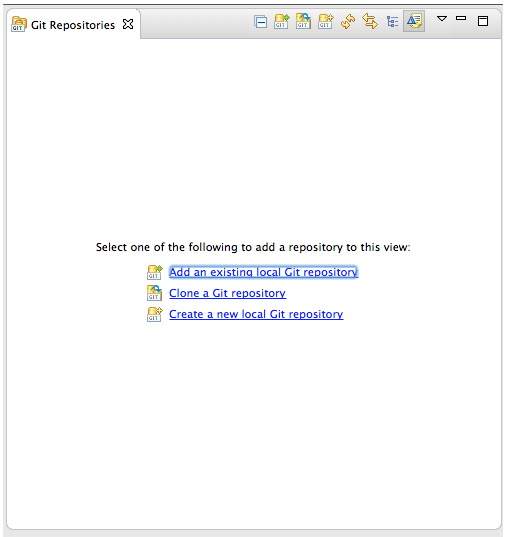

Fig. 22. A view of the NiCE Git Perspective.

'Git Repository' view (or the hyperlink in the middle of the view if you have not repositories). You will be presented with the wizard in Figure 23. Enter `https://github.com/idaholab/moose` into the URI entry and select next. This will present you with the branch selection wizard page (Figure 24. Select which branches you'd like to import in this clone and click Next. The last page will let you specify the clone location on your local filesystem (Figure 25.

To import MOOSE into the NiCE Project Explorer, simply right click the created moose repository in the Git Repository view and select 'Import Projects'. On the first wizard page, select *Import as New Project* and click finish. This will present you with the NiCE New Project wizard (Figure 26). In this wizard, open the C/C++ tree node and select *Makefile project with Existing Code*. Provide a valid project name and toolchain and click finish. You should see MOOSE in your Project Explorer.

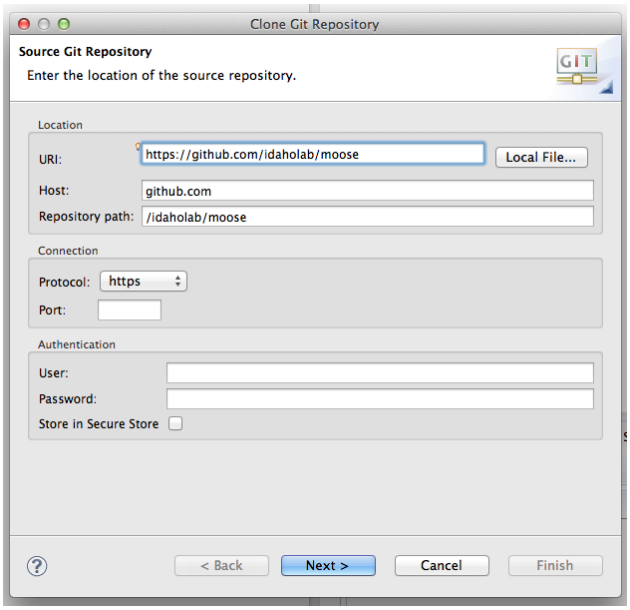

Fig. 23. The first page of the Clone Repository Wizard requesting information about the remote Git repository URL.

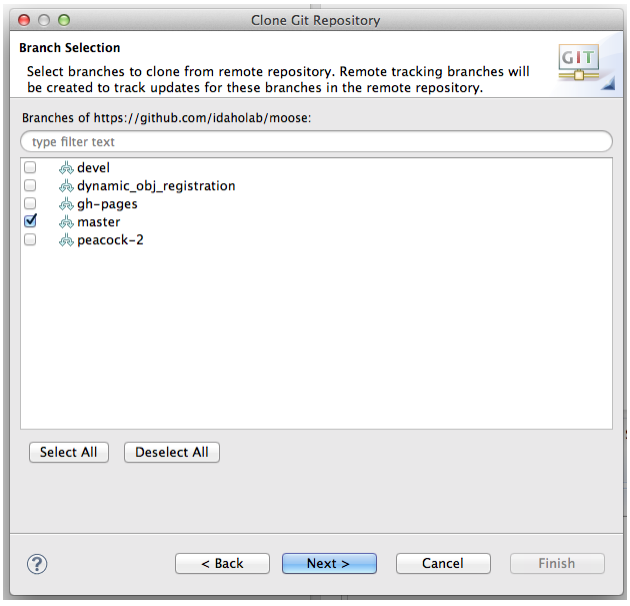

Fig. 24. The second page of the Clone Repository Wizard requesting information about which branches to pull down.

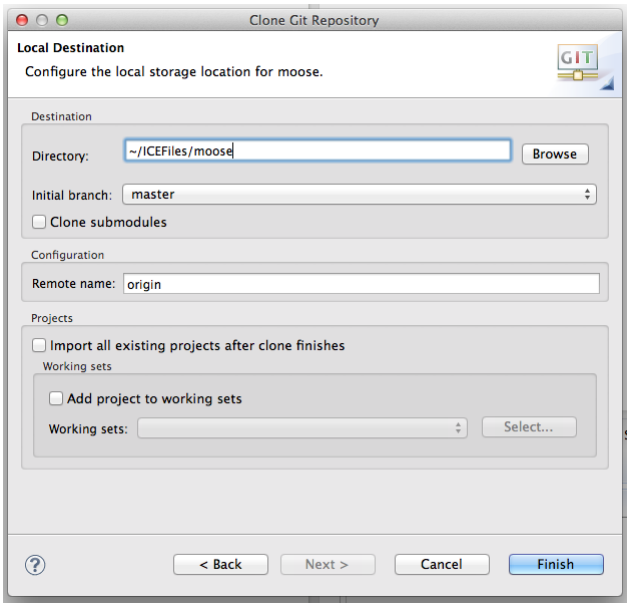

Fig. 25. The final page of the Clone Repository Wizard requesting information about the local location of the repository.

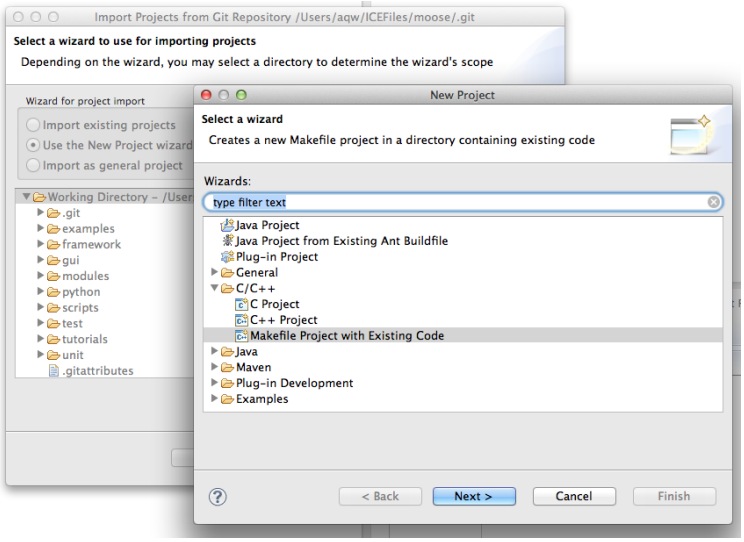

Fig. 26. The import projects wizard for the NiCE Git repositories view.

4.2 Building MOOSE

To build MOOSE/Libmesh within NiCE, open the Make Target view by going to *Window > Show View > Other* and search and select Make Target. With MOOSE imported into your Project Explorer, you should see the MOOSE project in the Make Target view. Right click on that project and select New. A dialog will pop up prompting you for the Make Target name, target name, and build command. Set the name as 'Build Libmesh', uncheck 'Same as target name' and leave the Make target blank, uncheck 'Use builder settings' and set the command as 'sh scripts/update_and_rebuild_libmesh.sh', then click 'Ok'. Now you should see a 'Build Libmesh' target, which upon double-clicking will execute the update_and_rebuild_libmesh.sh script with the output streaming in the Console view.

Once that is done, you can create another Make Target in the same manner, this time setting the target as all, but setting the build command 'make -C framework ' (feel free to add -j N to this command, where N is the number of make threads). Now, double-clicking this make target will execute the MOOSE build, and you should see the output streaming in the Console.

4.3 Forking the Stork

The internal MOOSE development team provides another GitHub repository called *stork* located at `https://github.com/idaholab/stork` that represents the base structure needed to create a new MOOSE application. So *Forking the Stork* implies forking this repository, changing its name to whatever you've decided to call your MOOSE application, and cloning that locally to begin work.

NiCE now provides this functionality in an easy-to-use toolbar button using the tools provided by the Eclipse EGit plugins. To *Fork the Stork* in NiCE, simply click the *MOOSE Fork the Stork* button in the toolbar (Figure 27).

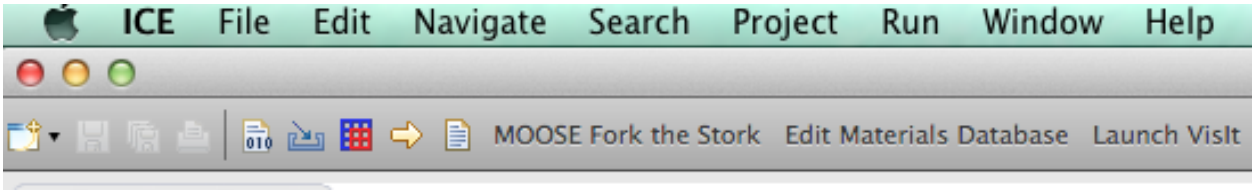

Fig. 27. The NiCE Fork the Stork button in the toolbar.

This will present a new dialog asking for the name of your new MOOSE application, as well as your GitHub username and password (Figure 28). Upon providing this information and clicking 'Ok', NiCE will fork the `https://github.com/idaholab/stork` repository for you, rename it to your provided application name, clone it to your workspace, and import it into NiCE as a new C++ project in the C/C++ perspective's Project Explorer view (Figure 29).

Additionally, the import generates a fully configured Make Target in the Make Target view (Figure 30), and sets up the C++ Indexer to point to your NiCE MOOSE project's include files. This is essential for providing code completion and MOOSE code search while your developing your MOOSE application. To look at a MOOSE class that you've referenced in one of your application's source files, simply click the class name or the header file and click F3. NiCE will take you directly to the declaration for that MOOSE class so that you can peruse and look up its method definitions.

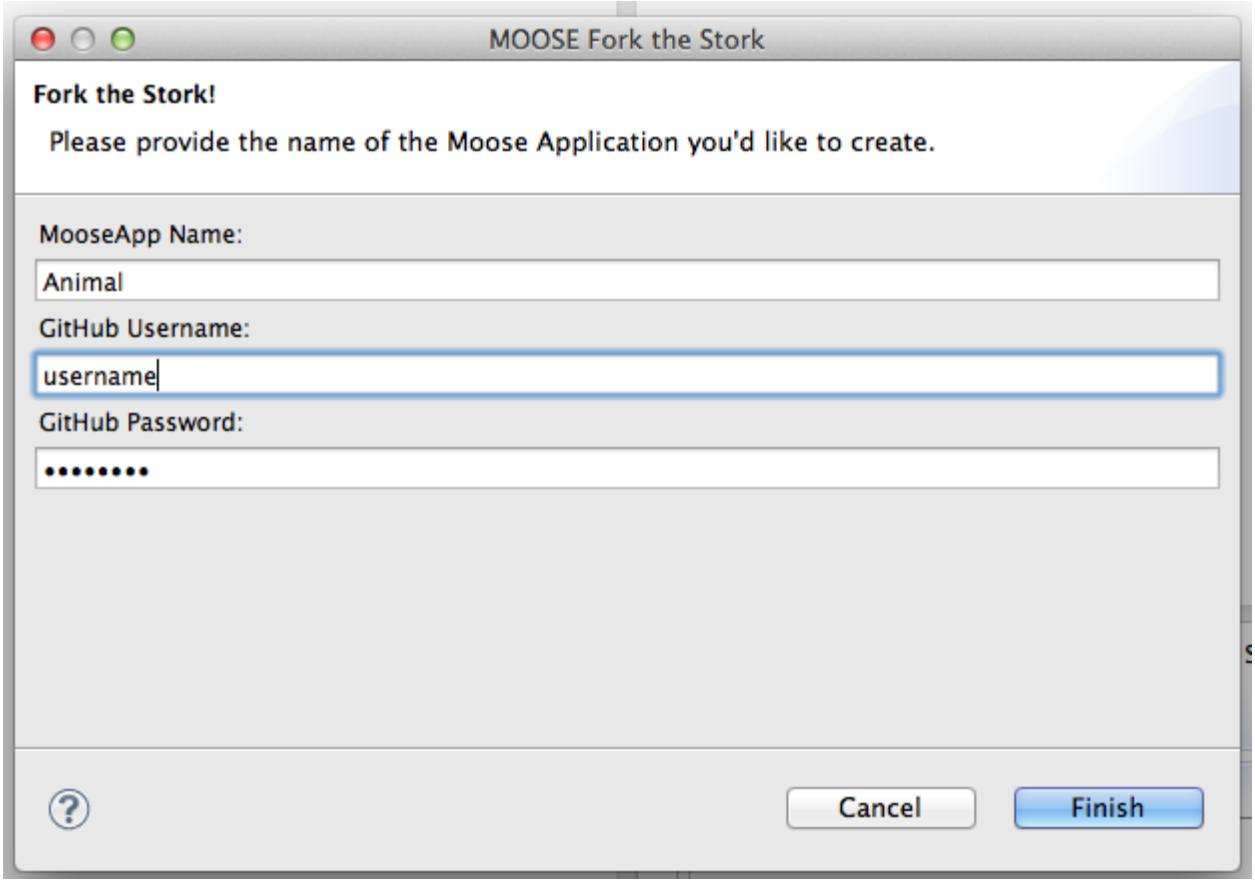

Fig. 28. The Fork the Stork Wizard for entering your new MOOSE application name and your GitHub credentials.

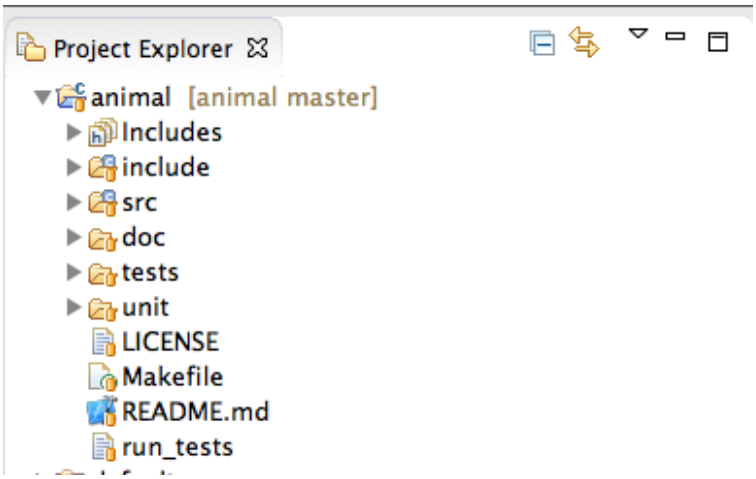

Fig. 29. NiCE creates the newly forked MOOSE application as a C/C++ project in the Project Explorer.

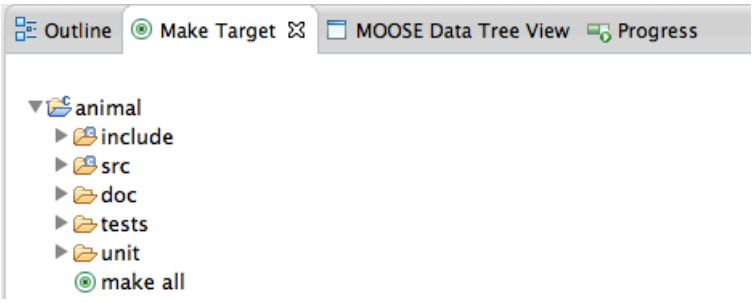

Fig. 30. NiCE adds a new Make Target to the Make Targets View for building your new application.

4.4 Adding a New Kernel

Once you've cloned and built MOOSE, and Forked the Stork to produce a new MOOSE application ready for development, you can easily create custom Kernels with NiCE. To create a new Kernel, right click on your new MOOSE-based application project and select *New > MOOSE Object > Kernel* (Figure 31). This action will display an input prompt asking for the name of your new Kernel subclass. Simply enter the name and push 'Ok'. Then NiCE will automatically generate a new include and source file in include/kernel and source/kernel, respectively. The new files are the stubbed out, base implementation of a subclassed Kernel that you can then add to and modify (Figure 32).

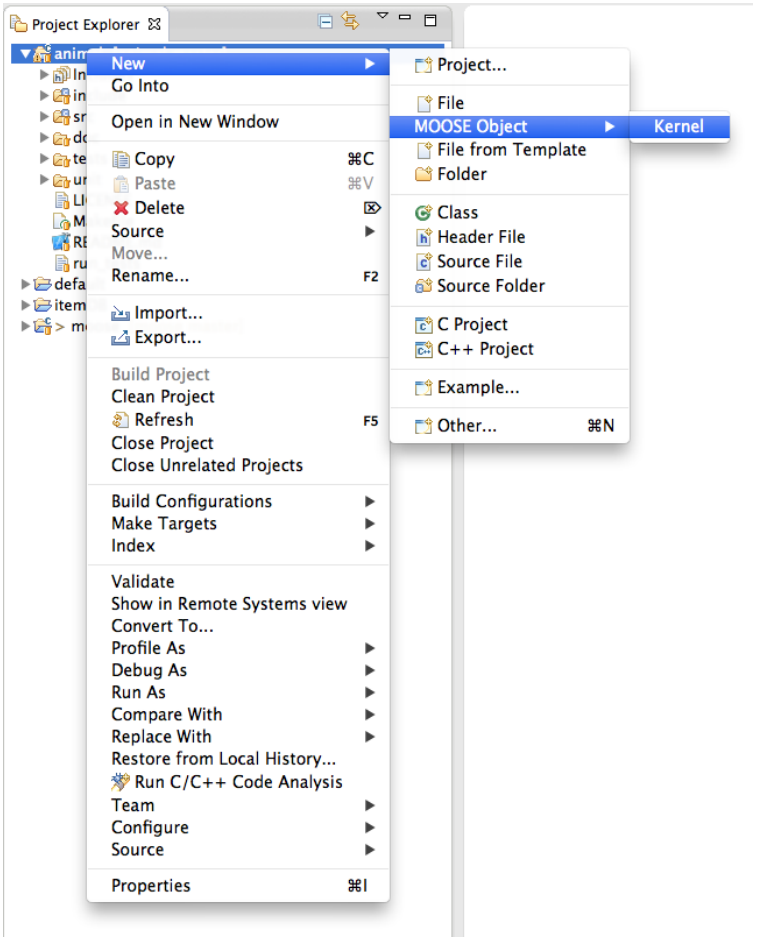

Fig. 31. Creating a new Kernel in NiCE is easy, just right click on your project and select New - MOOSE Object - Kernel.

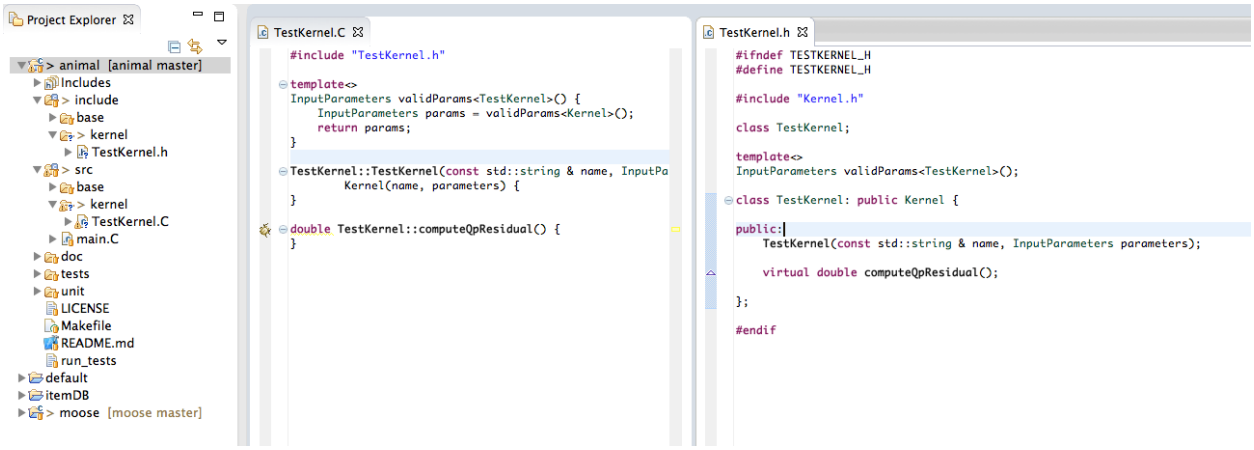

Fig. 32. The created source code for the Add Kernel context menu action.

4.5 Pushing Changes Back to GitHub

To push changes to the remote GitHub repository at `https://github.com/username/animal`, switch back to the Git perspective and click your applications git repository in the Git Repositories view. On the bottom right of the screen, you should see another set of tabbed views, one of them being the Git Staging view (Figure 33).

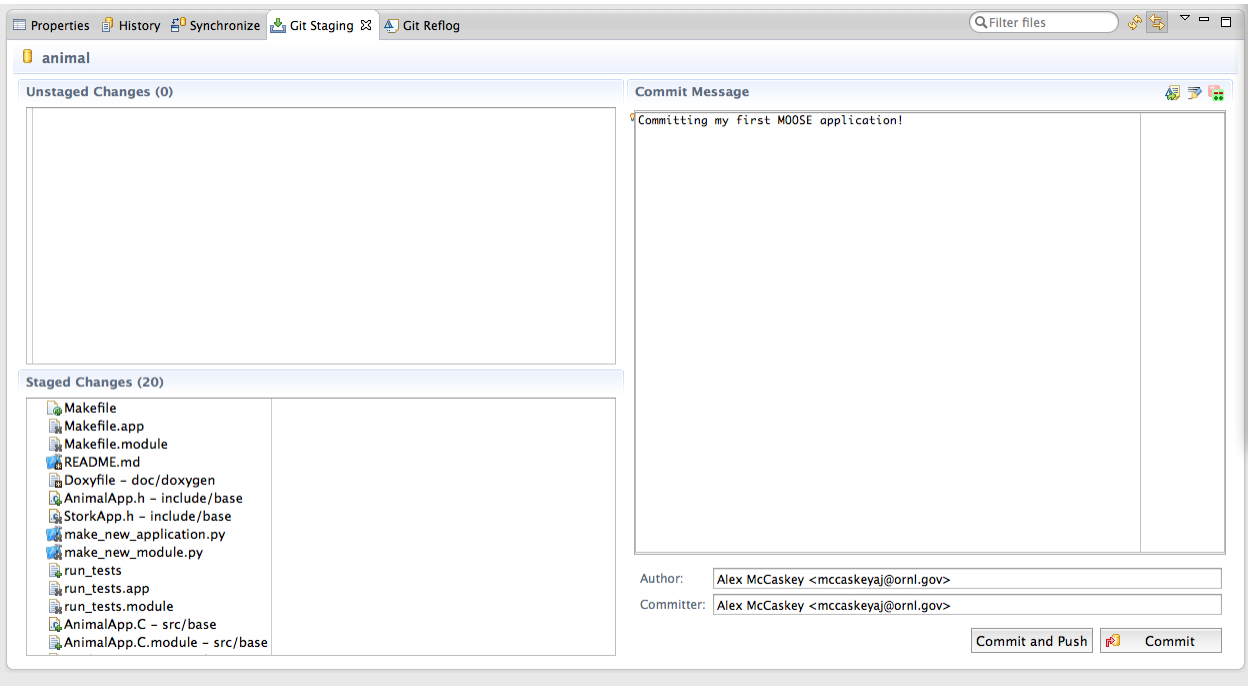

Fig. 33. Committing your new MOOSE application is easy, just stage it in the Git Staging view of the Git Repositories Perspective.

Click the Git Staging View and drag any Unstaged Changes to the Staged Changes section. Now provide a brief commit message and click *Commit and Push*, enter your GitHub credentials, and watch as your files are committed to the remote repository!

5. Acknowledgements

The NiCE team would like to acknowledge the BISON and MOOSE teams at Idaho National Laboratory, specifically Rich Williamson, Danielle Perez, Cody Permann, and Andrew Slaughter, who served as subject matter experts in making this work possible. Their input and advice greatly advanced this work, and will continue to do so in the future. Additionally, the NiCE team would like to acknowledge the funding agency for this work, the U.S. Department of Energy Office of Nuclear Energy's Nuclear Energy Advanced Modeling and Simulation (NEAMS) program and the Advanced Modeling and Simulation Office (AMSO) within DOE-NE.

REFERENCES

[1] Derek Gaston, Chris Newman, Glen Hansen, and Damien Lebrun-Grandié. MOOSE: A Parallel Computational Framework for Coupled Systems of Nonlinear Equations. *Nuclear Engineering and Design*, 239(10):1768–1778, 2009.

[2] R.L. Williamson, J.D. Hales, S.R. Novascone, M.R. Tonks, D.R. Gaston, C.J. Permann, D. Andrs, and R.C. Martineau. Multidimensional multiphysics simulation of nuclear fuel behavior. *J. Nuclear Materials*, 423(1-3):149–163, 2012.