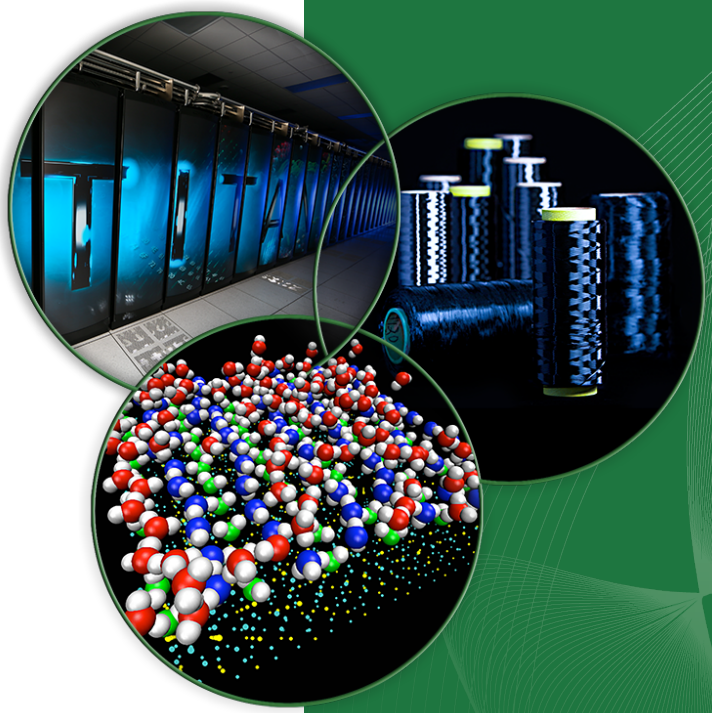
Modernizing Simulation Input Generation and Post-Simulation Data Visualization with Eclipse ICE



Alex McCaskey
Research Staff
Oak Ridge National Laboratory
mccaskeyaj@ornl.gov
@amccaskey2223

Taylor Patterson
Research Associate
Oak Ridge National Laboratory
pattersontc@ornl.gov
@TCPatt

Jay Jay Billings
Research Staff, Project Lead
Oak Ridge National Laboratory
billingsjj@ornl.gov
@jayjaybillings
+jayjaybillings

EclipseCon North America 2015
Burlingame, CA, USA
2015-03-12

Outline

- ICE and the Standard Model of Scientific Computing
- Streamlining XML-based scientific code integration in ICE
- Visualization through VisIt integration

Additional Resources: http://www.eclipse.org/ice

GitHub

github.com/eclipse/ice

Eclipse Wiki

wiki.eclipse.org/ICE

YouTube

youtube.com/user/jayjaybillings

Standard Model of Scientific Computing

All users must do these things...

Define the Problem

Write an input file in a format reminiscent of a dead language

Run the Simulator

Manually launch jobs on impressively terrifying machines

Analyze Output

```
01100010
01101001
01101110
01100001
01110010
01111001
```

Analyze simulation output in its most raw and unlimited form

Archive Output

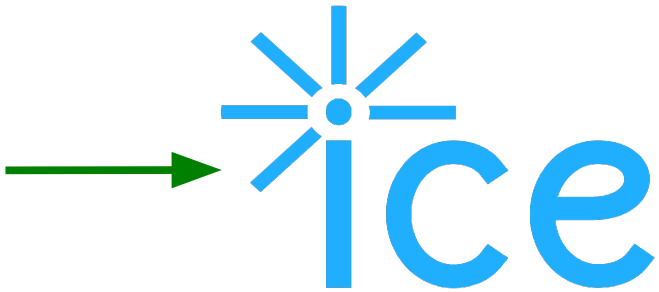
Store data... somewhere!

Super-users think these are easy tasks, but most users are overwhelmed!

A cooler model of Scientific Computing

It would be better to have a computer program handle all of that...

A. User



Define the Problem

Run the Simulator

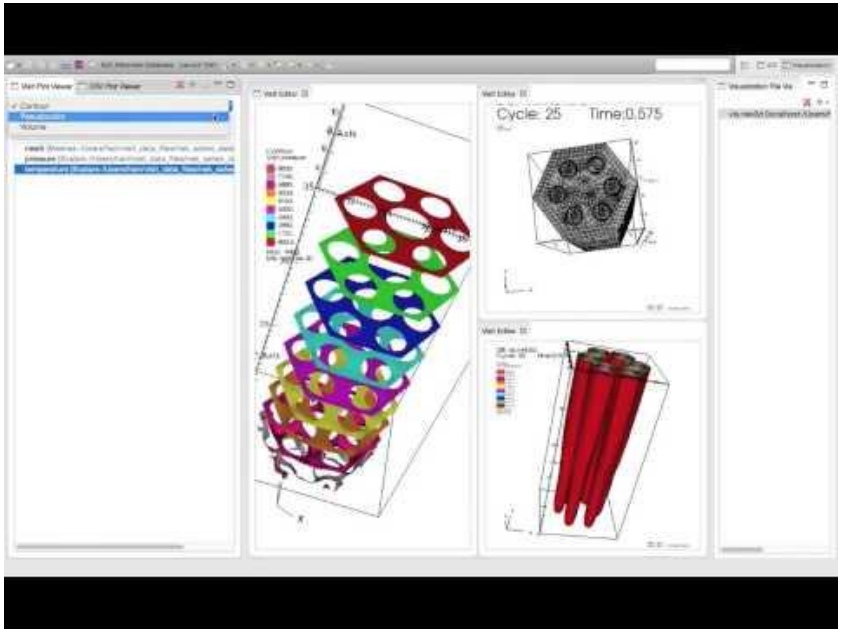
Analyze Output

01100010
01101001
01101110
01100001
01110010
01111001

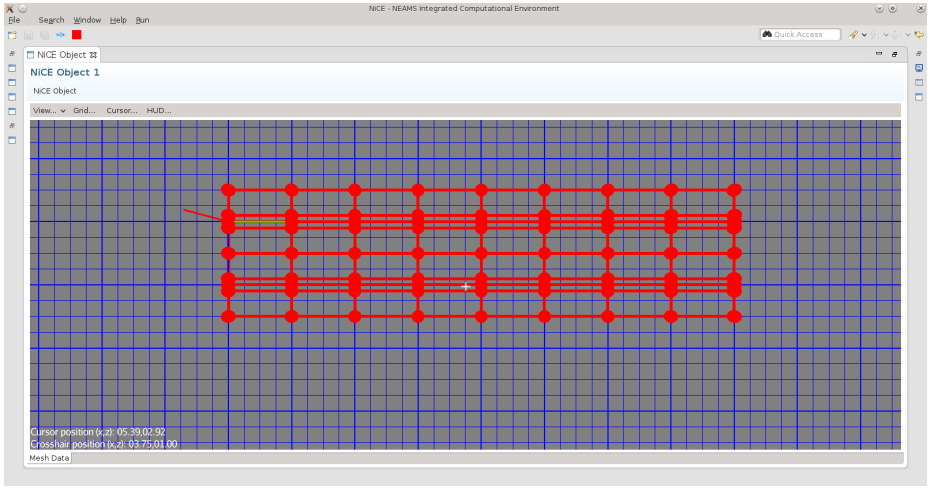
Archive Output

Most of the stuff we need to do can be encapsulated for ease of use and/or automated entirely with improvements.

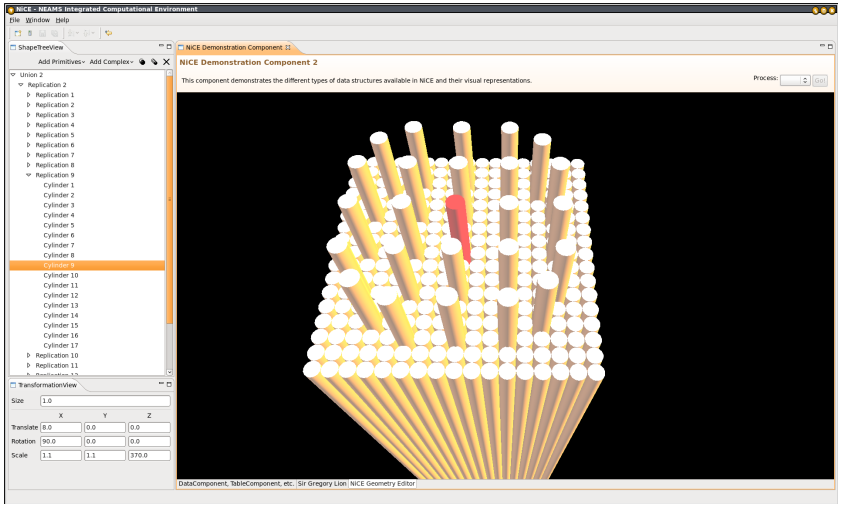
ICE in Action

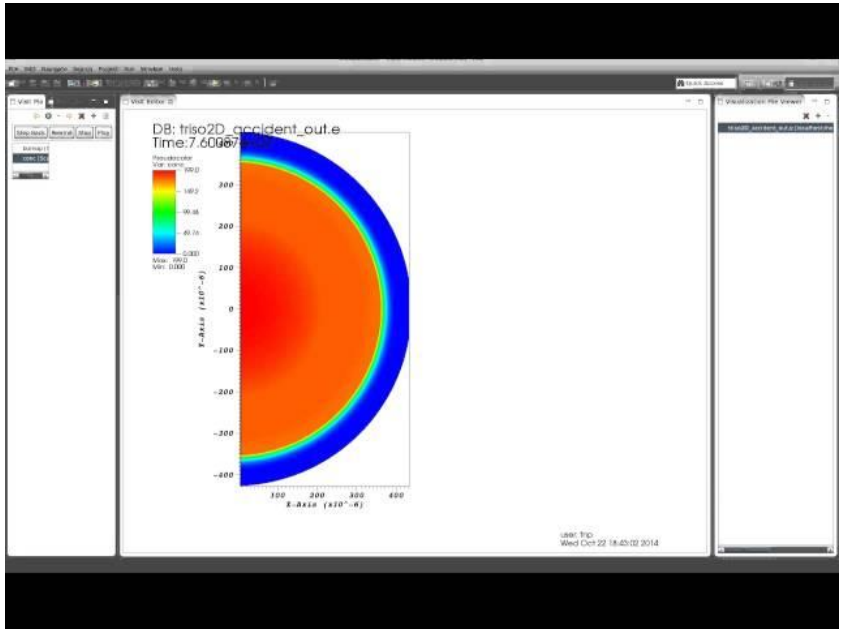


Fully interactive 3D support in VisIt



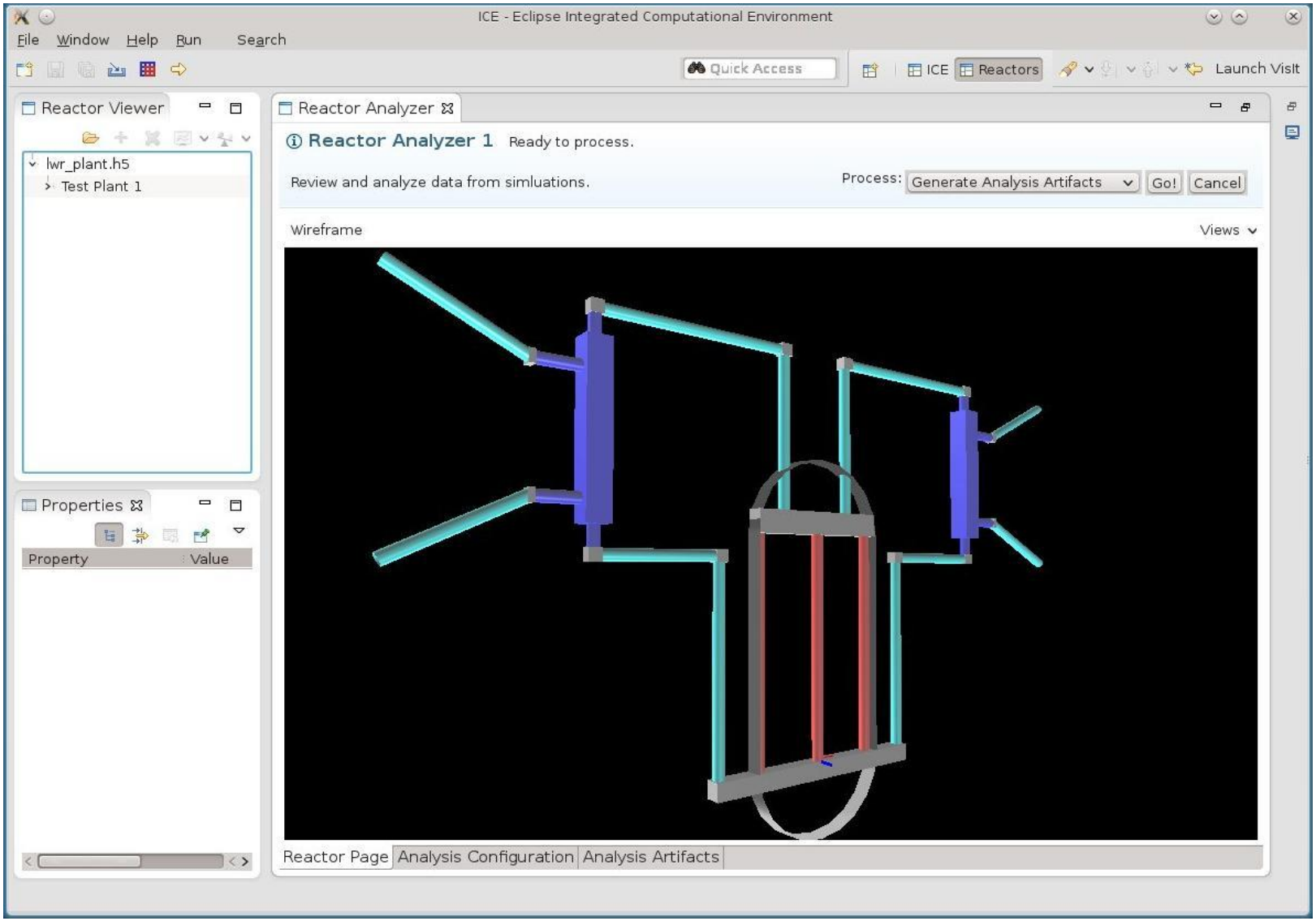
JMonkeyEngine for Mesh and Geometry





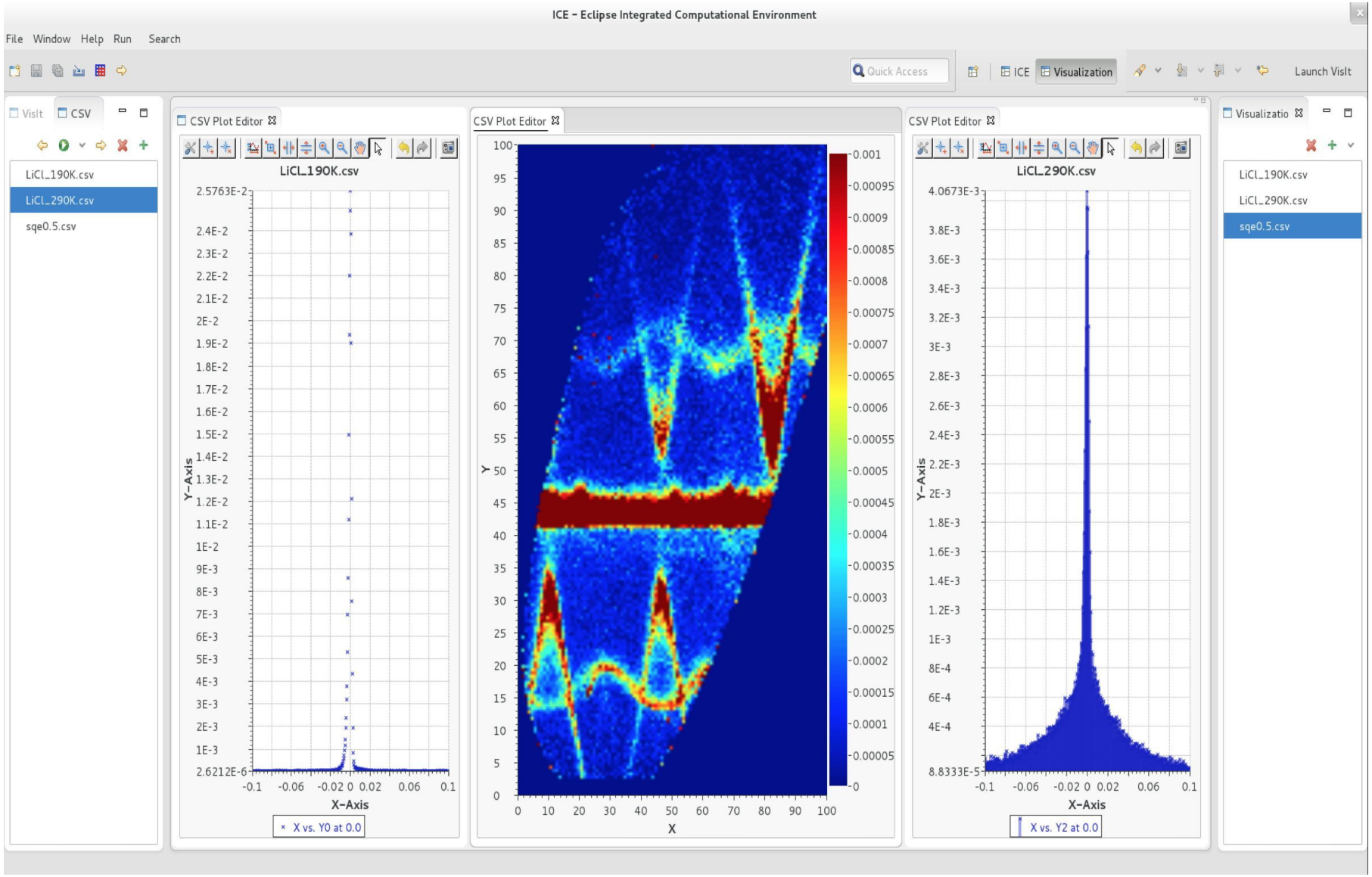
An early video of the triso results in ICE's visualization perspective (thus the bugs!)

ICE in Action



3D Model of a Nuclear Plant (TMI)

ICE in Action



SNS Phonon Scattering Data

Today's Focus

Streamlining the implementation of input generation tools for developers

A. User

Define the Problem

Run the Simulator

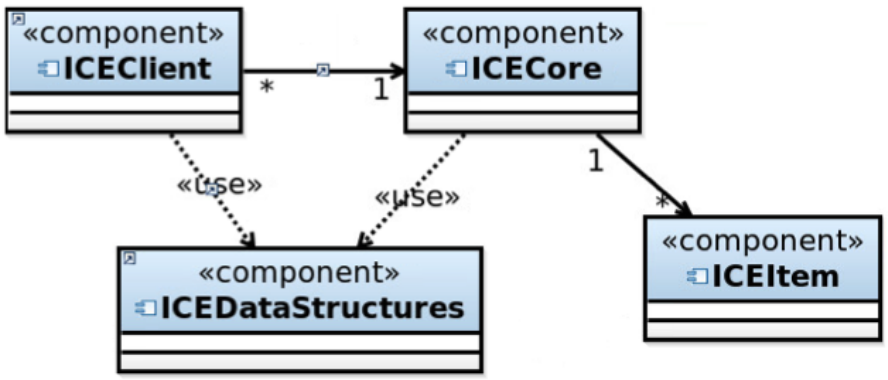
Analyze Output

01100010
01101001
01101110
01100001
01110010
01111001

Archive Output

VisIt provides the tools for generating 2D and 3D visualizations

ICE Architecture: How we do it...



The platform is easily modified:

- Developers write *plugins* for their scientific codes
- We handle data representations, processing, presentation, etc...

Plugin code goes here!

Plugins are:

- Dynamic Services - Completely reusable components!
- "Item" Subclasses - Most of the work is already done by the platform
- Self-contained, business logic - **ONLY** your code, not UI, etc.
- Tools - Reusable components, tools, or things other

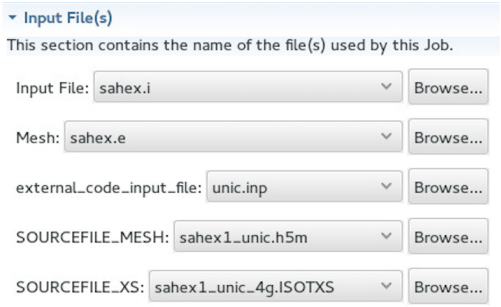
ICE Architecture: Data Structures

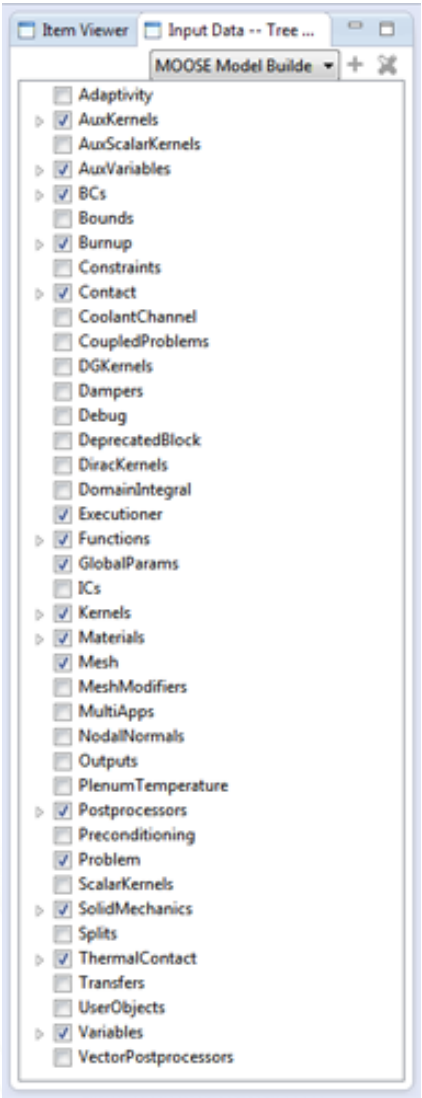
ICE parses each Form and presents each component graphically to the user

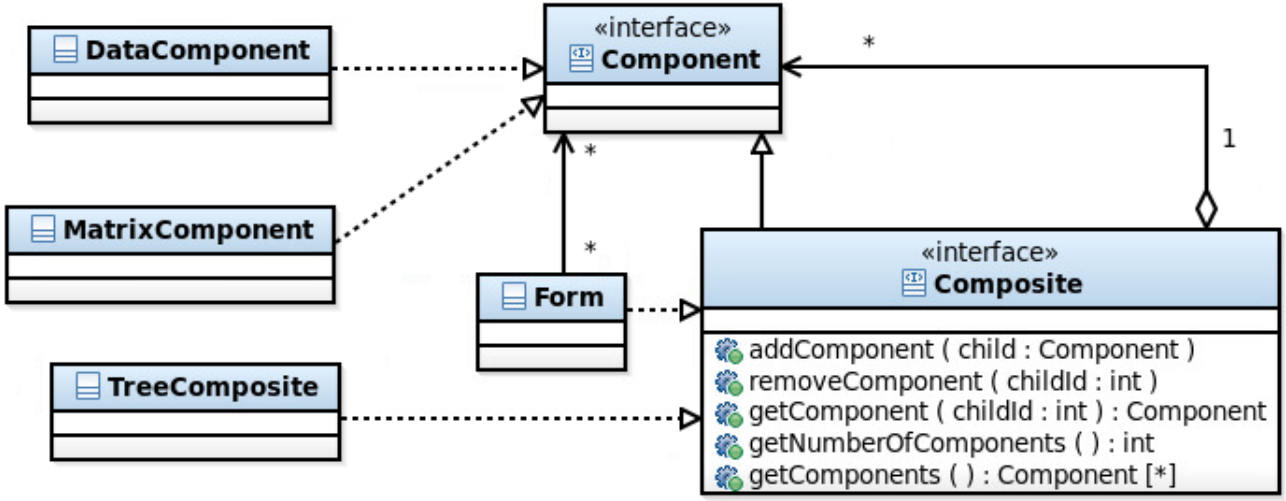
- TreeComposite, DataComponent UI View

Each Item provides a hierarchical Form

- Each Form contains Components
- Tree Pattern







XML and Scientific Community

Quantum Computing

```
<?xml version="1.0"?>
<Form itemID="1" id="1" name="Boxio2" description="" HashCode="1772275030">
    <DataComponent name="Physical Program Name and Description" id="1" description="Please provide the name and
        description for this Physical Program.">
        <Entry name="Physical Program Name" value="Boxio2" ready="true" secretFlag="false" changeState="false"
            defaultValue="" id="1" description="Provide a name for this Program, it will serve as a unique
            identifier as well as the name of the save file.">
            <AllowedValueType>Undefined</AllowedValueType>
        </Entry>
        <Entry name="Physical Program Description (Optional)" value="" ready="true" secretFlag="false"
            changeState="false" defaultValue="" id="1" description="Please provide a brief description of this
            Program">
            <AllowedValueType>Undefined</AllowedValueType>
        </Entry>
    </DataComponent>
    <DataComponent name="Processor Files" id="2" description="Please provide valid Entity files needed by this
        Program.">
        <Entry name="Processor File" value="${jadefolder}/FullyConnected8Qubit.xml" ready="true"
            secretFlag="false" changeState="false" defaultValue="UnitCell.xml" description="Please provide a Jade
            Processor to use with this Logical Program.">
            <AllowedValueType>Discrete</AllowedValueType>
        </Entry>
        <Entry name="Qubit Bias Annealing Schedule" value="x1/30" ready="true" secretFlag="false" changeState=
            "false" description="Set the annealing schedule for the qubit bias parameters.">
            <AllowedValueType>Undefined</AllowedValueType>
        </Entry>
        <Entry name="Qubit Coupling Annealing Schedule" value="x1/30" ready="true" secretFlag="false"
            changeState="false" description="Set the annealing schedule for the qubit coupling parameters.">
            <AllowedValueType>Undefined</AllowedValueType>
        </Entry>
        <Entry name="Qubit Tunneling Annealing Schedule" value="1-(x1/T)" ready="true" secretFlag="false"
            changeState="false" description="Set the annealing schedule for the qubit tunneling parameters.">
            <AllowedValueType>Undefined</AllowedValueType>
        </Entry>
        <Entry name="Final Annealing Time" value="30" ready="true" secretFlag="false" changeState="false"
            defaultValue="" id="1" description="The end time of the input annealing schedule.">
            <AllowedValueType>Undefined</AllowedValueType>
        </Entry>
        <MatrixComponent resizable="true" valueType="Undefined" nCols="8" nRows="8" isSquare="true" name="Final
            Ising Parameters" id="3" description="The Final Ising Parameters generated by JadeD for the given
            Problem and Processor configuration.">
            <elements>1</elements>
            <elements>1</elements>
            <elements>0</elements>
            <elements>1</elements>
            <elements>1</elements>
            <elements>0</elements>
            <elements>0</elements>
            <elements>0</elements>
            <elements>1</elements>
```

XML widely used in scientific community:

- Standardized
- Readers/Writers
- Hierarchical

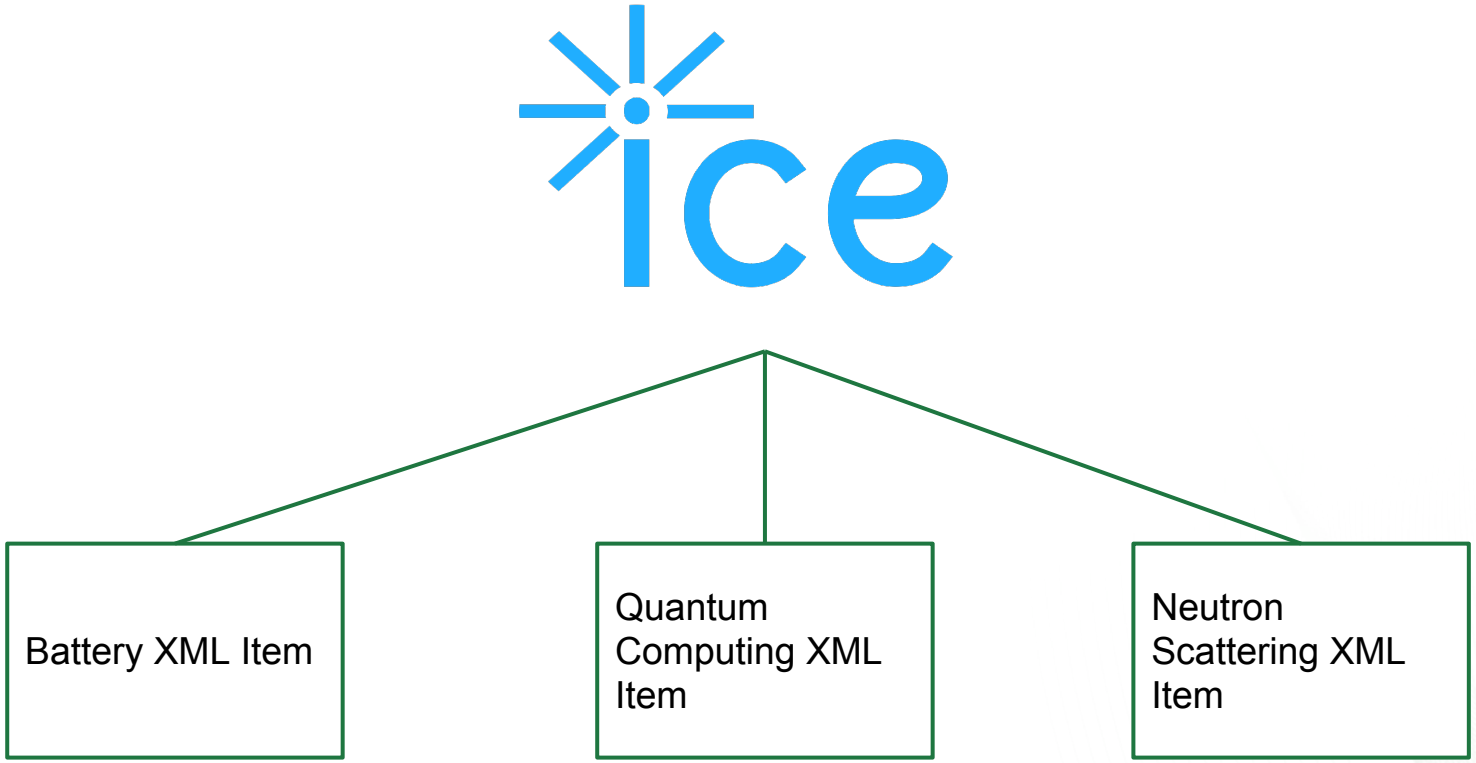
Neutron Scattering

```
<root>
    <sample>
      <structure>
        <file>toppar/crd.md18_vmd_autopsf.pdb</file>
        <format>pdb</format>
      </structure>
      <framesets>
        <frameset>
          <file>production_single.dcd</file>
          <format>dcd</format>
    </frameset>
      </framesets>
    </sample>
    <stager>
      <target>system</target>
```

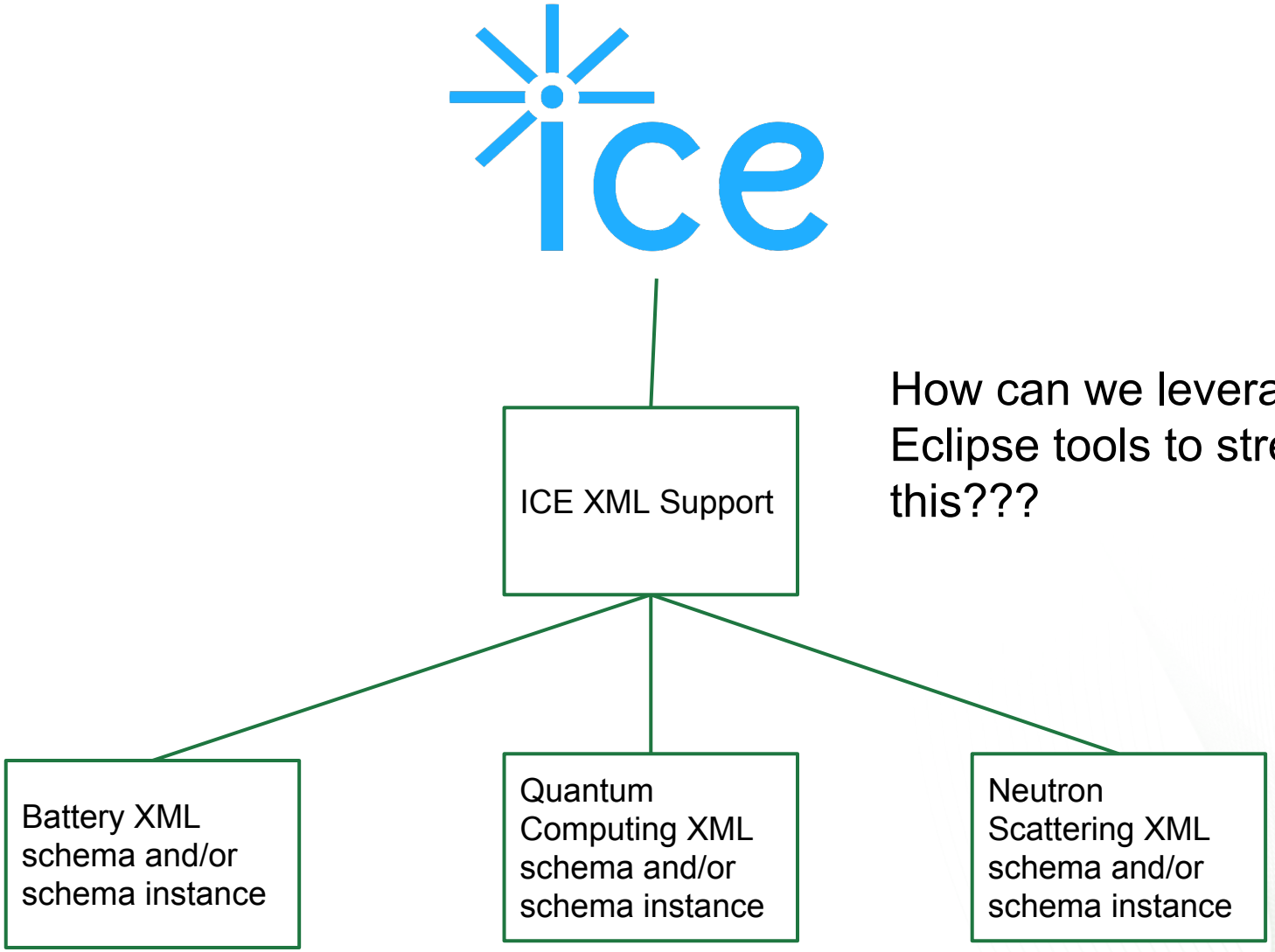
Advanced Batteries

```
<?xml version="1.0" encoding="utf-8"?>
<BatteryML_Doc xmlns:xsi="http://www.w3.org/2001/XMLSchema-instanc
    "BatteryML:2.0 batteryml:2.0" name="input_Electricity">
  <ModelDB>
    <Model id="input_Electricity">
      <Definition>
        <Category id="electrical">
          <Parameters>
            <Parameter id="NumberOfMeshes">
              <Scalar>
                <String>1</String>
              </Scalar>
              <Unit></Unit>
            </Parameter>
            <Parameter id="DoD">
              <Scalar>
                <String>0.1</String>
              </Scalar>
              <Unit></Unit>
            </Parameter>
```

For each of these, and others, we have to write a new ICE Item plugin…



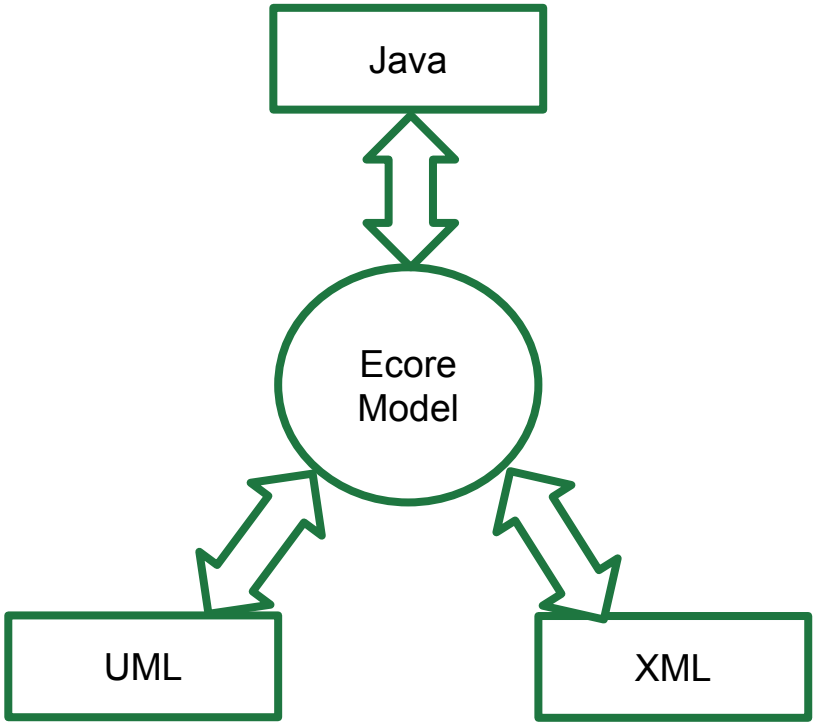
We want something like this...

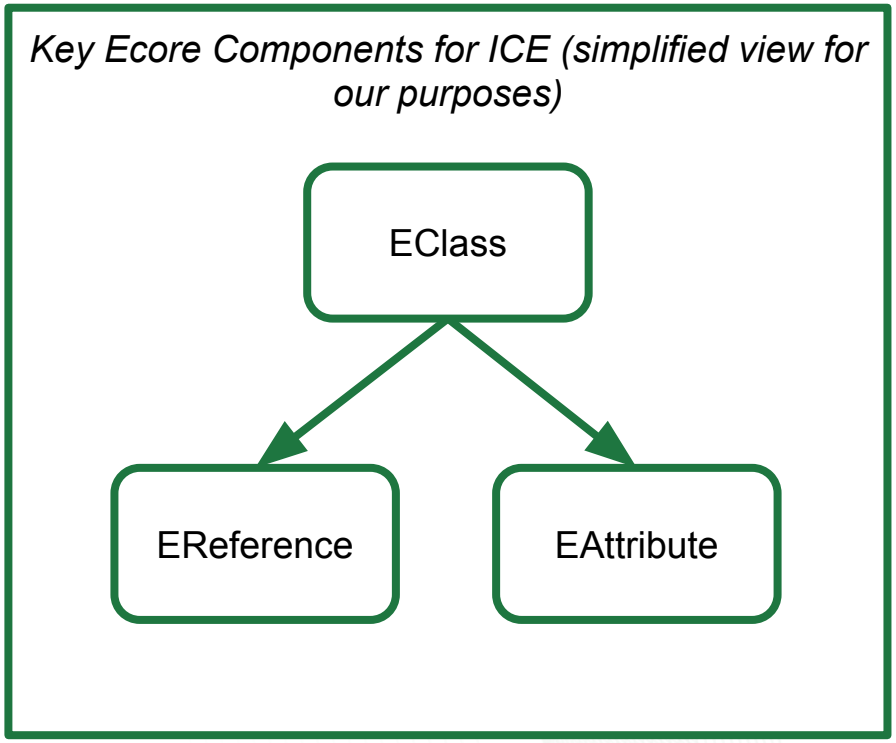


How can we leverage existing Eclipse tools to streamline this???

Eclipse Modeling Framework

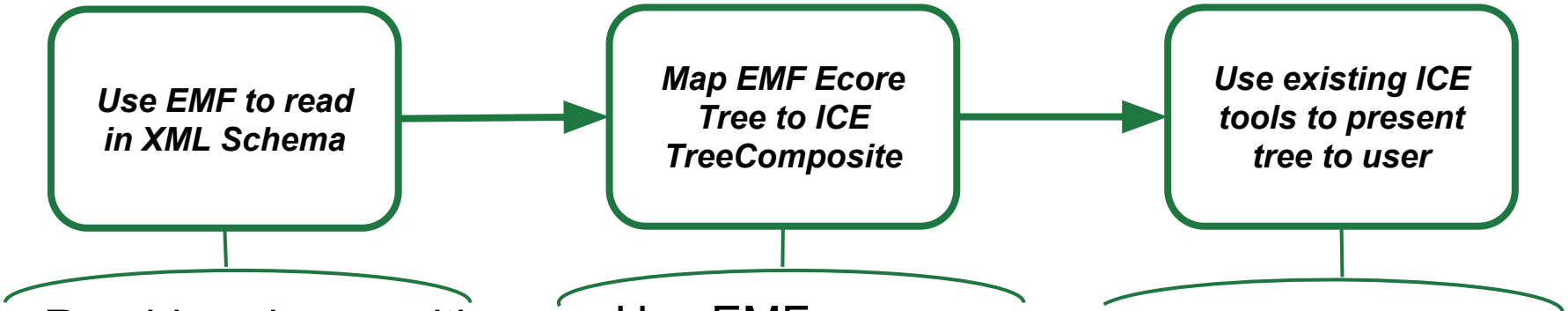
- XML Schemas define input ***models*** for these scientific codes…
- The EMF already handles XML well
- Reflection API





ICE Leveraging the EMF - High Level View

- **Goal**: Instead ***N*** Item plugins for ***N*** XML-input based codes, write just ***1***
- We can do this by leveraging the EMF ***and*** existing ICE data structures - TreeComposite



Read-in schema with existing EMF XMLProcessor

Let EMF do the heavy lifting!

Use EMF Reflection API to create ICE TreeComposite instances that correspond to EClass nodes in the Ecore Tree

Present with JFace TreeViewer with Properties Tab provider for data input

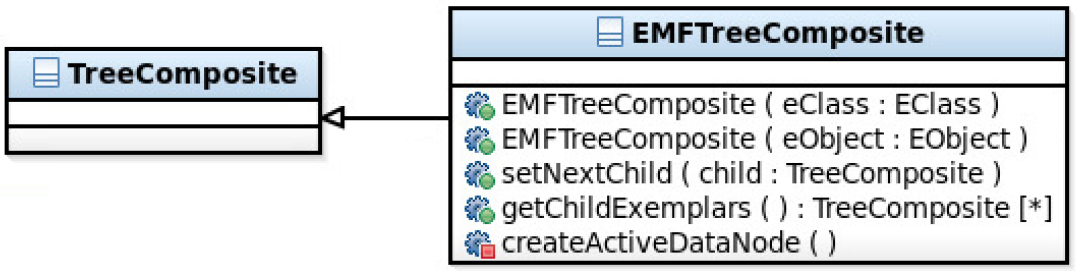
ICE Leveraging the EMF - Low Level View

```
// Create a new XMLProcessor to be used in creating
// and persisting XML Resources
try {
    xmlProcessor = new XMLProcessor(URI.createFileURI(file
            .getAbsolutePath()));
} catch (SAXException e) {
    e.printStackTrace();
}

// Get the package containing the model
EPackage ePackage = (EPackage) xmlProcessor.getEPackageRegistry()
        .values().toArray()[0];

// Get the TreeIterator to walk over the elements
TreeIterator<EObject> tree = ePackage.eAllContents();
```

Creating the Ecore Tree - Easy with the tools provided by EMF!

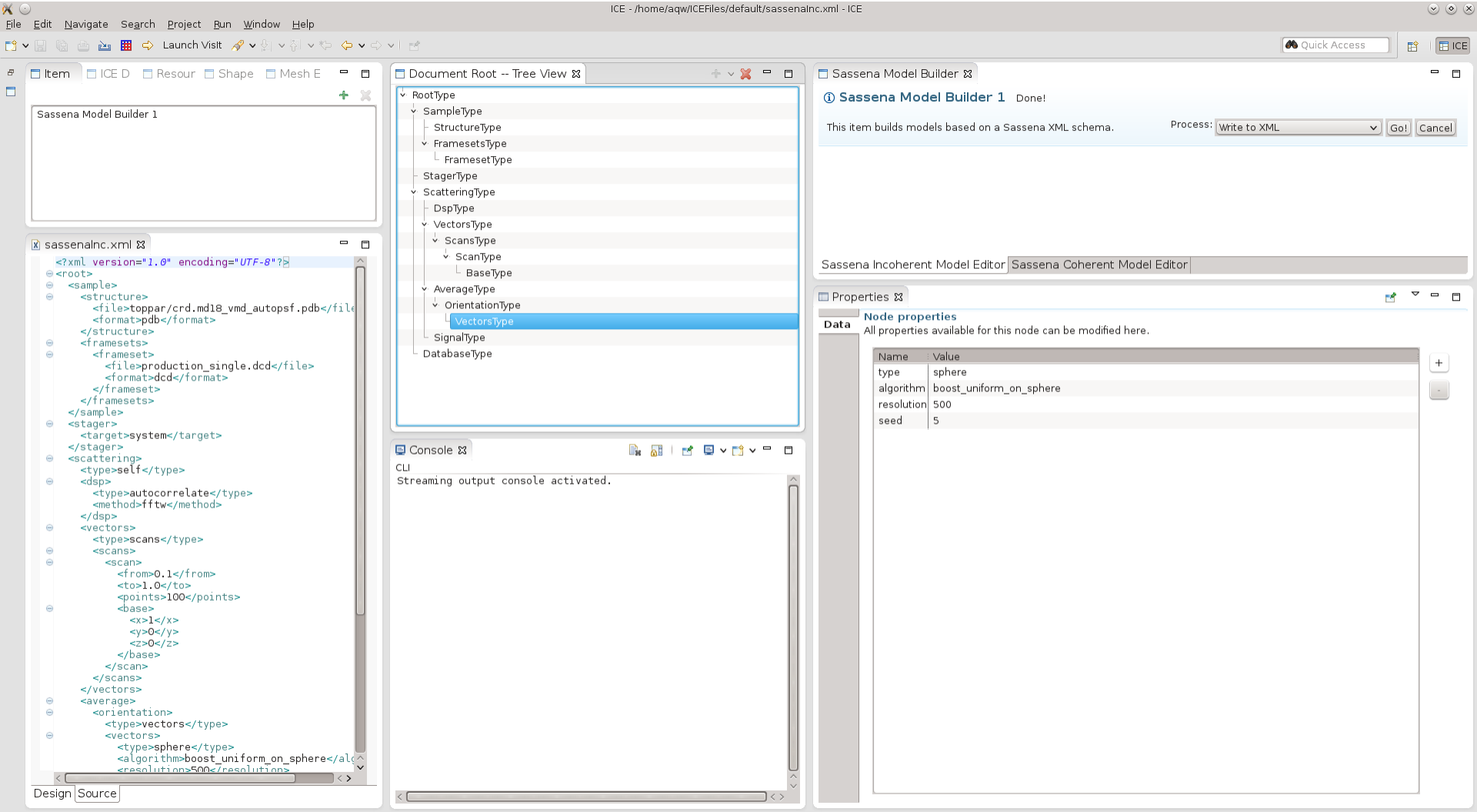
Map to ICE tree structure

Specialized behavior for adding children and querying ability to add children



Results!

Sassena - Neutron Scattering Simulation for the ORNL Spallation Neutron Source.



Post-Simulation Data Visualization in ICE through VisIt

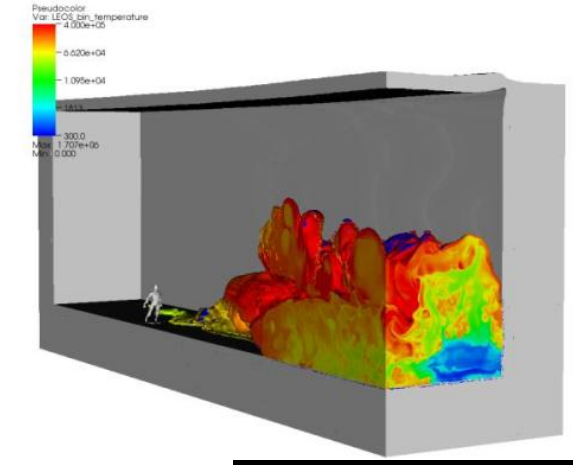
What is VisIt?

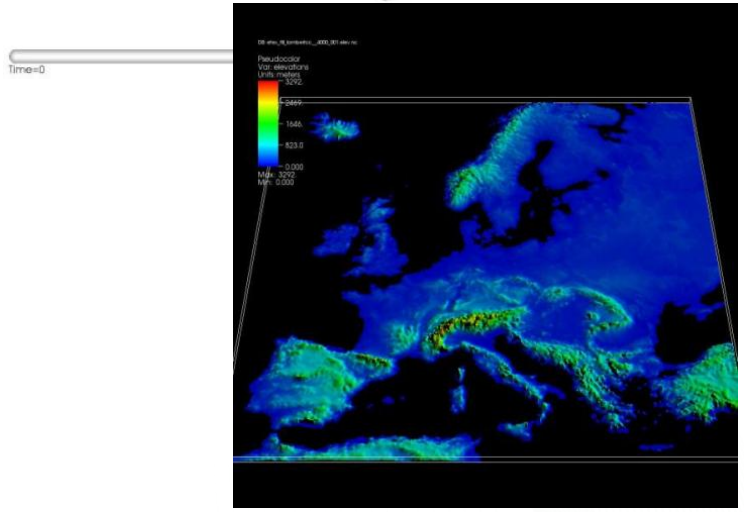
*"… an **open source**, interactive, scalable, visualization, animation and analysis tool."*

*"… a distributed, **parallel** visualization and graphical analysis tool for data defined on **2D and 3D** meshes."*

*"… originally developed… to visualize and analyze the results of **terascale** simulations… [but] has also proven to be well suited for visualizing smaller scale data..."*

*"… capable of visualizing data from over **120 different scientific data formats**."*





Credit: Quotes and images from VisIt Development Team website (https://wci.llnl.gov/simulation/computer-codes/visit)

Why include VisIt in ICE?

Simulation requires

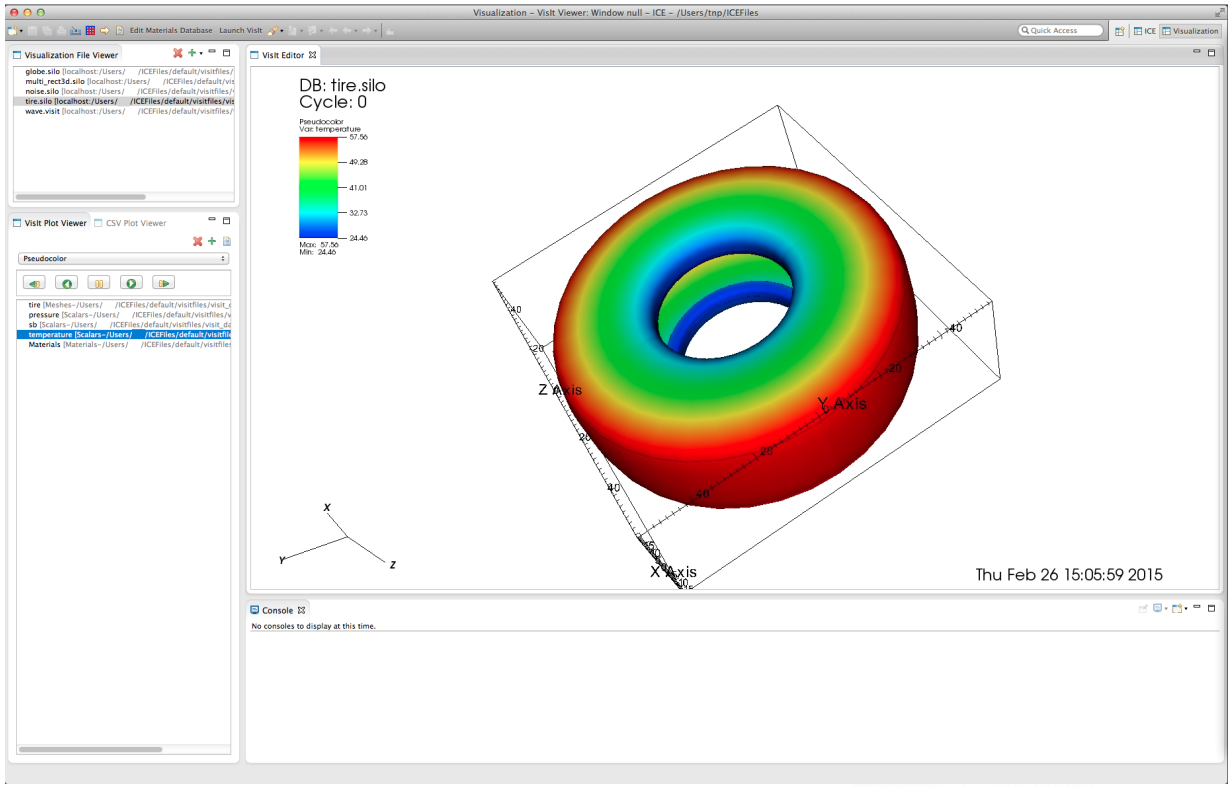
3D visualization

All-in-one approach

Familiar UI

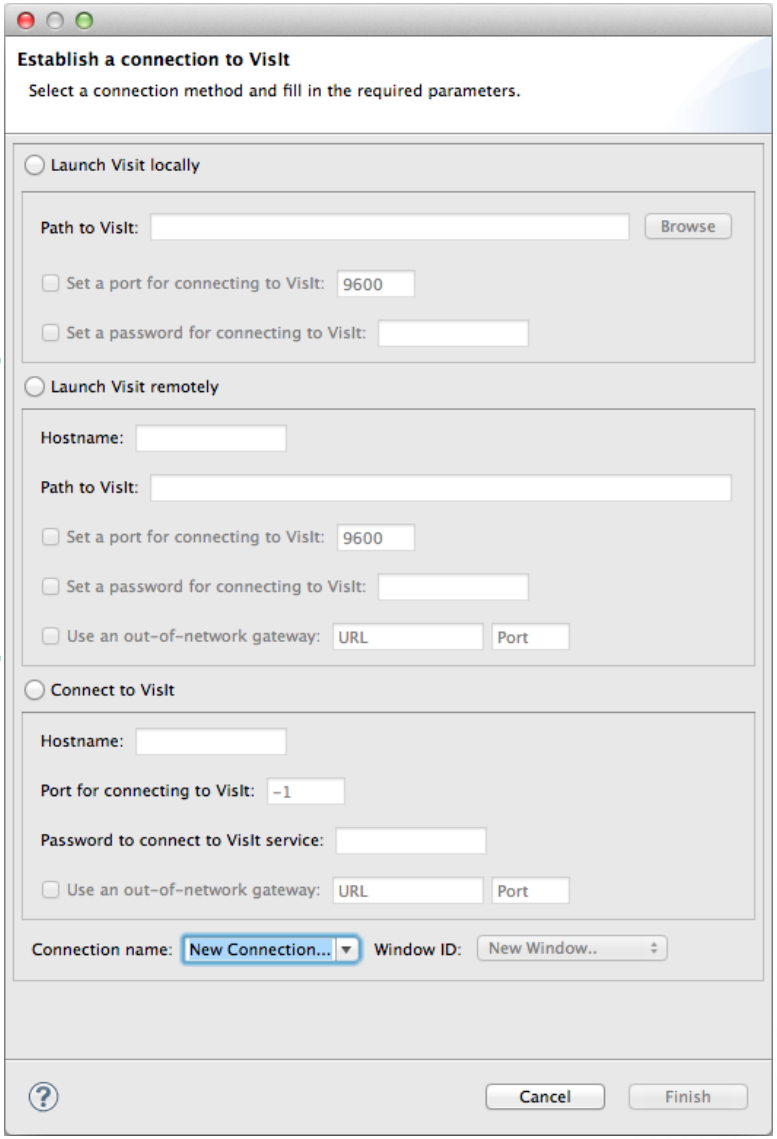
Users of ICE also

using VisIt

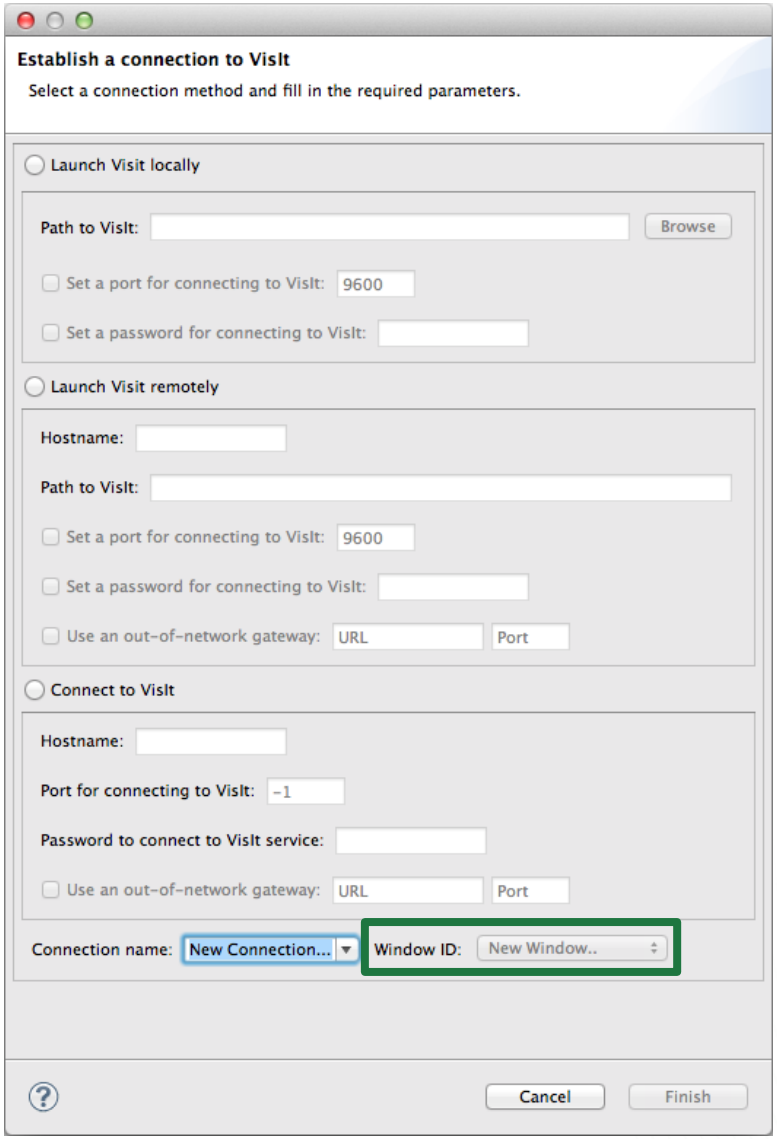


Features – Connections

Remote launch and rendering

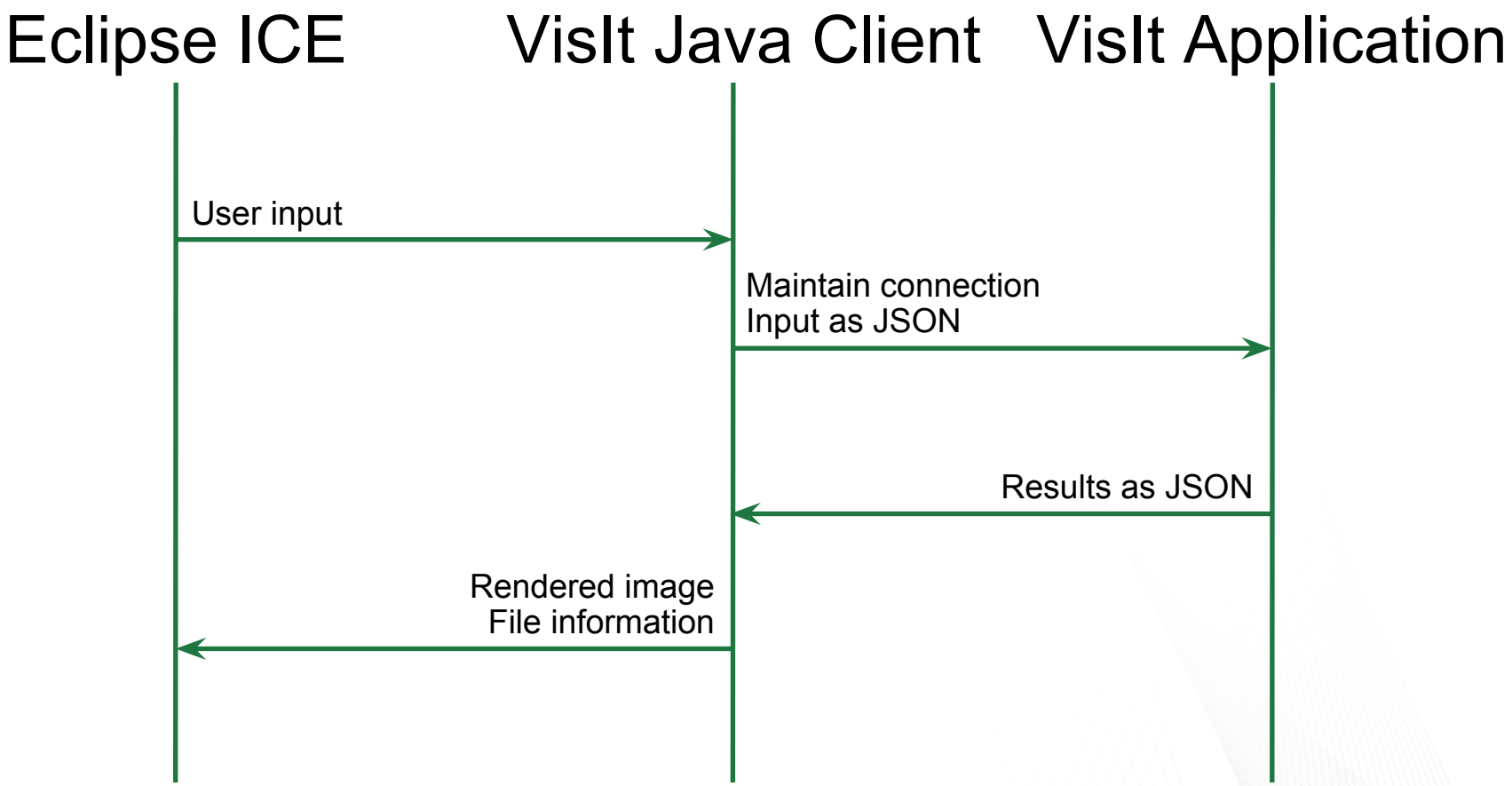


Features – Connections

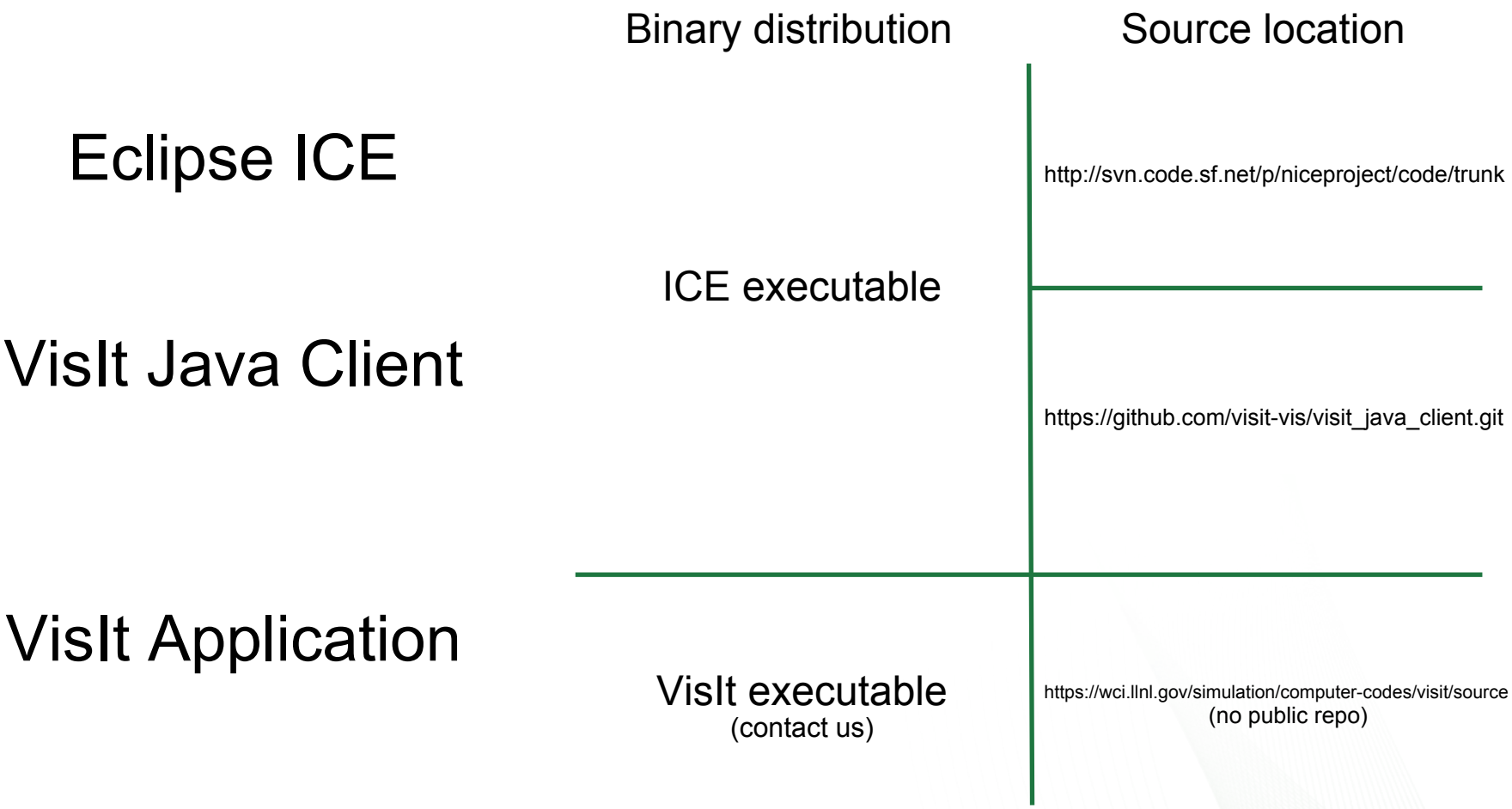


Single install capability

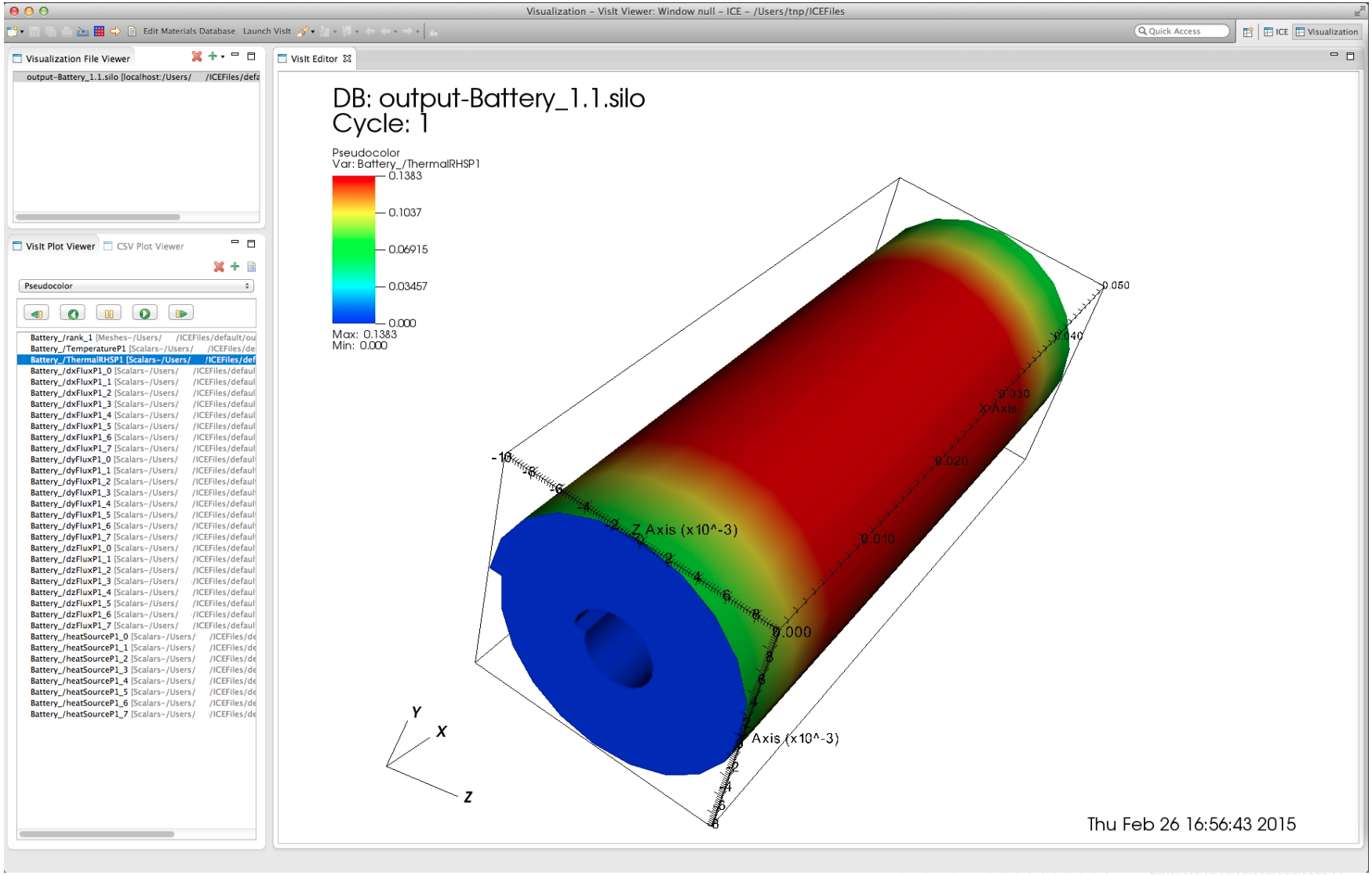
System Components



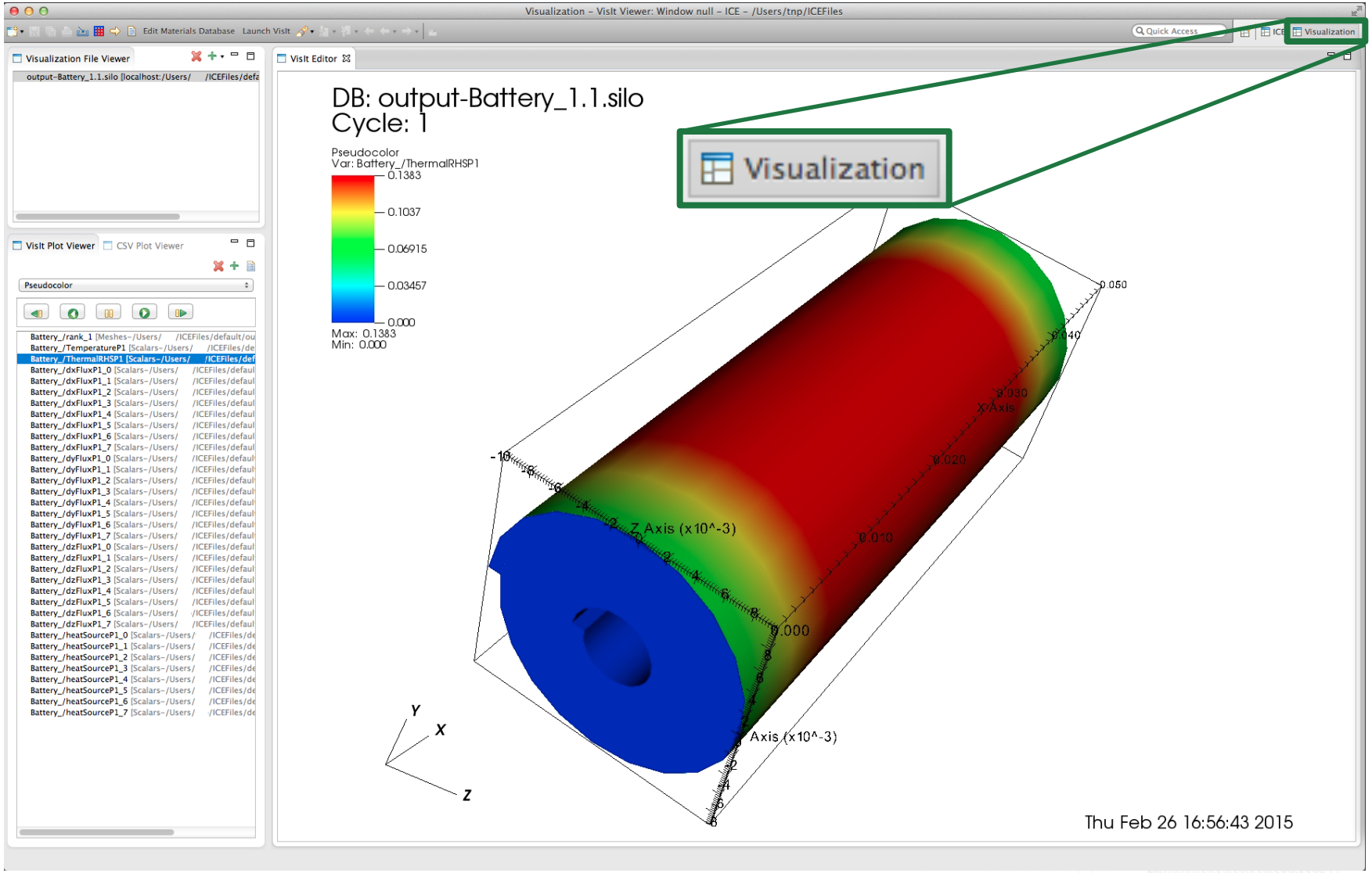
System Components



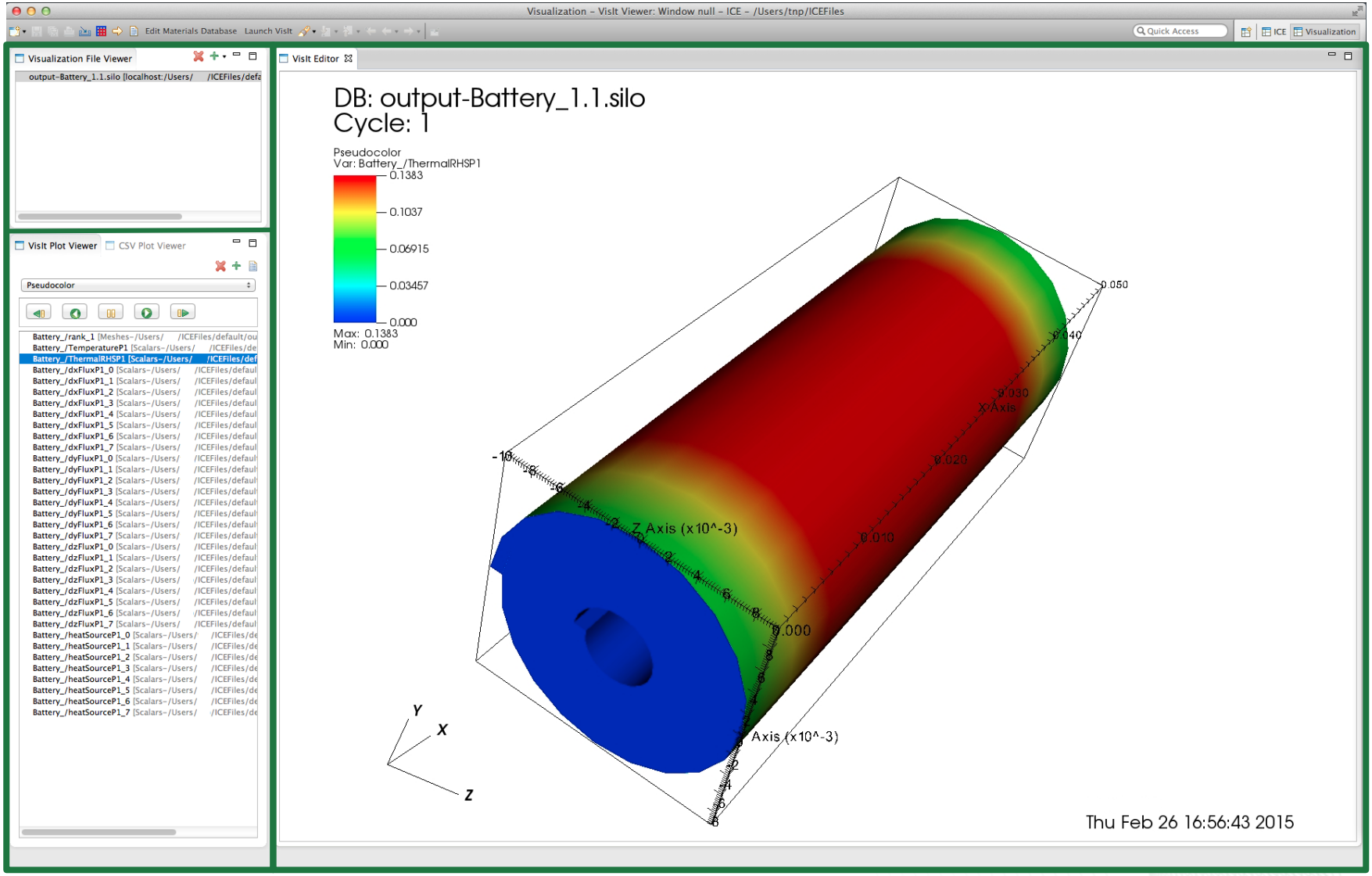
ICE Components – Perspective



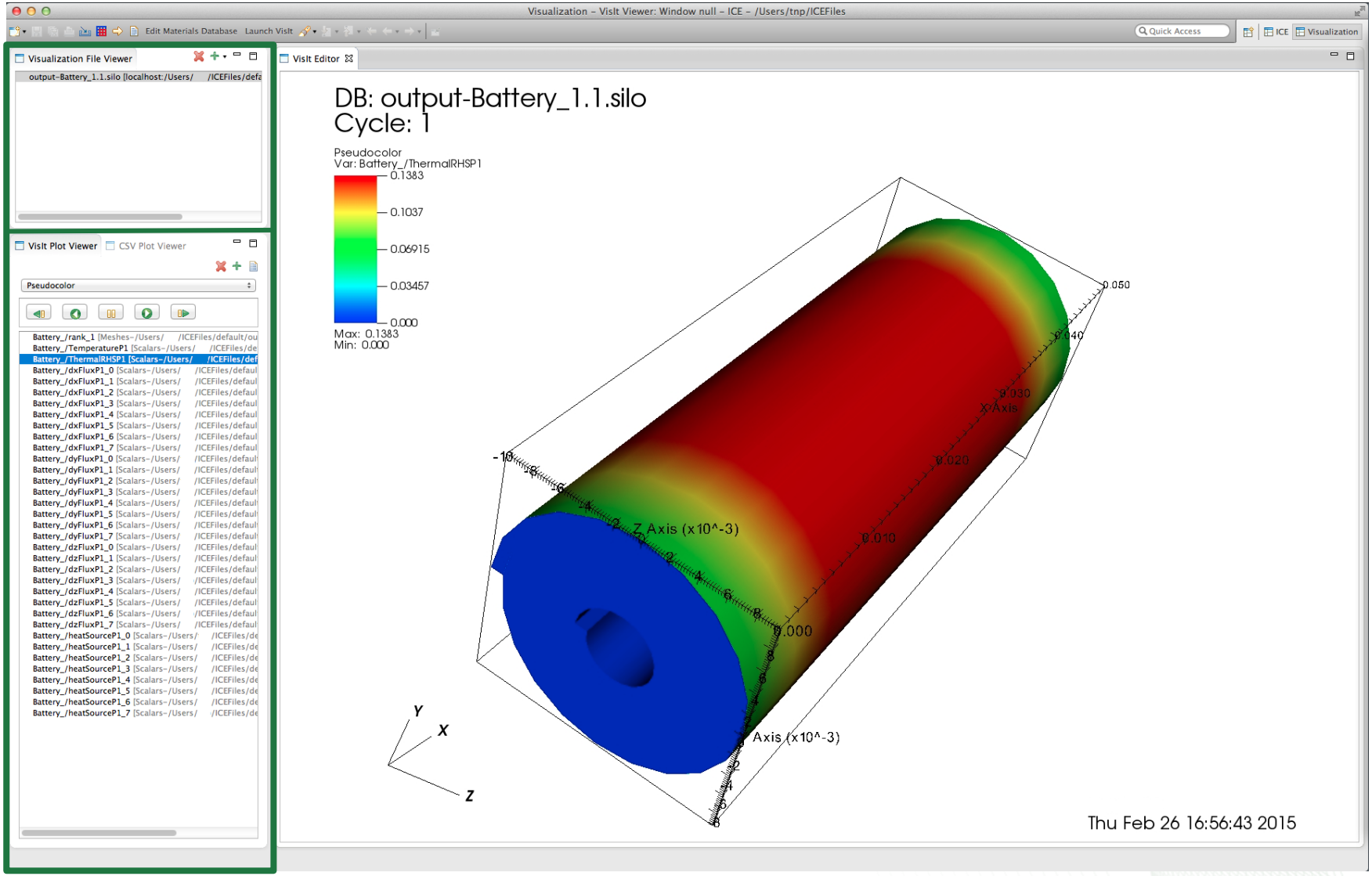
ICE Components – Perspective



ICE Components – Perspective

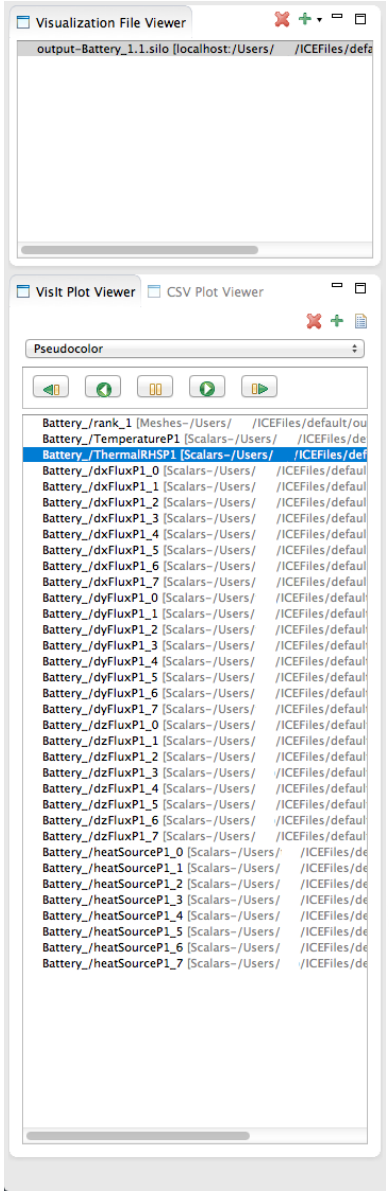


ICE Components – Viewers



ICE Components – Viewers

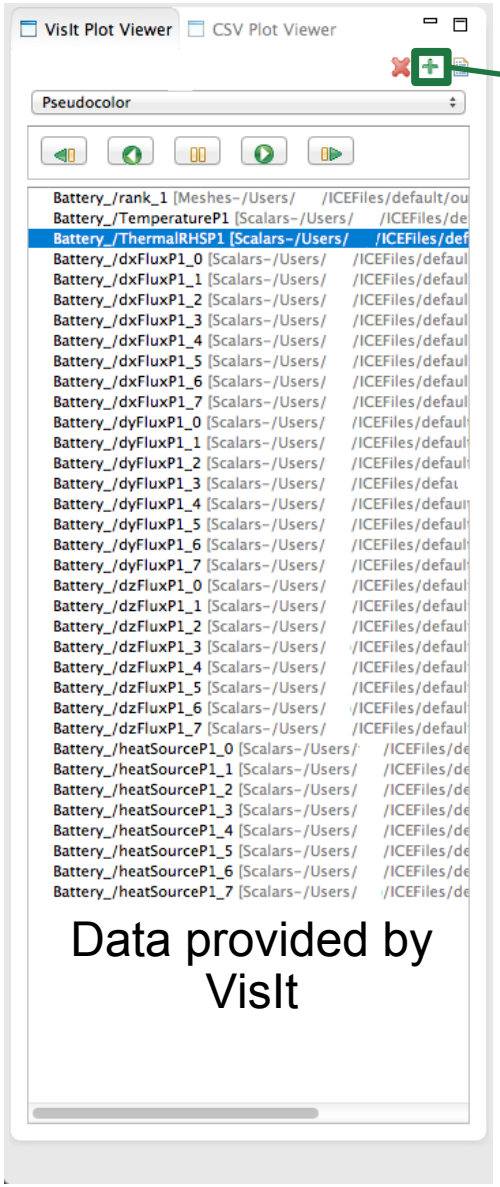
ViewPart subclasses with JFace TreeViewers

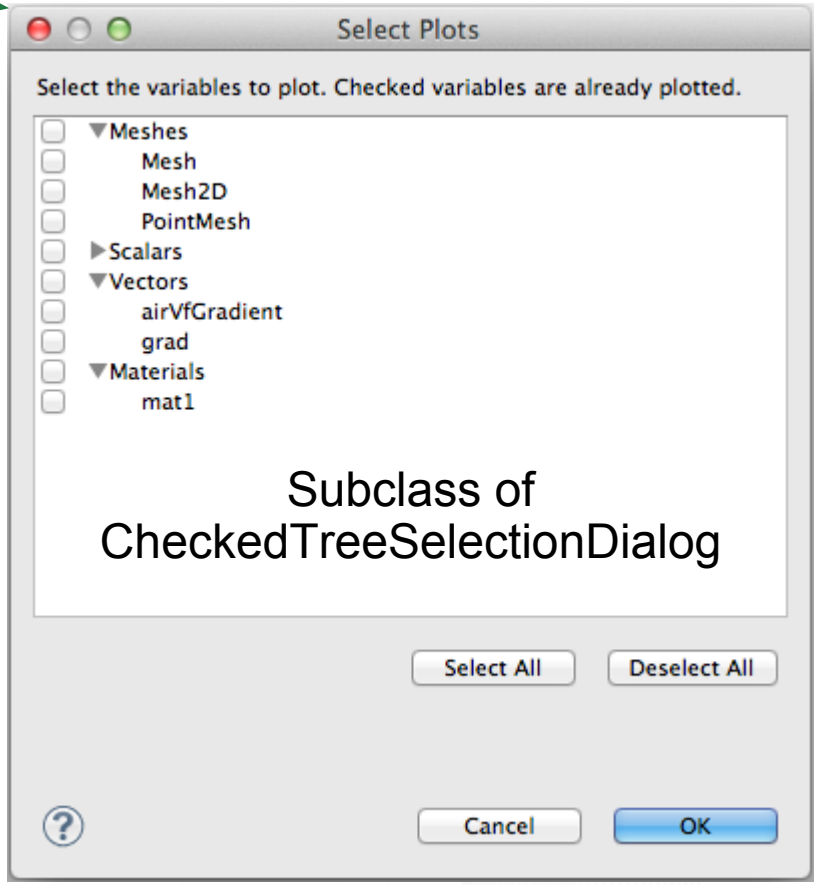


Data provided by ICE

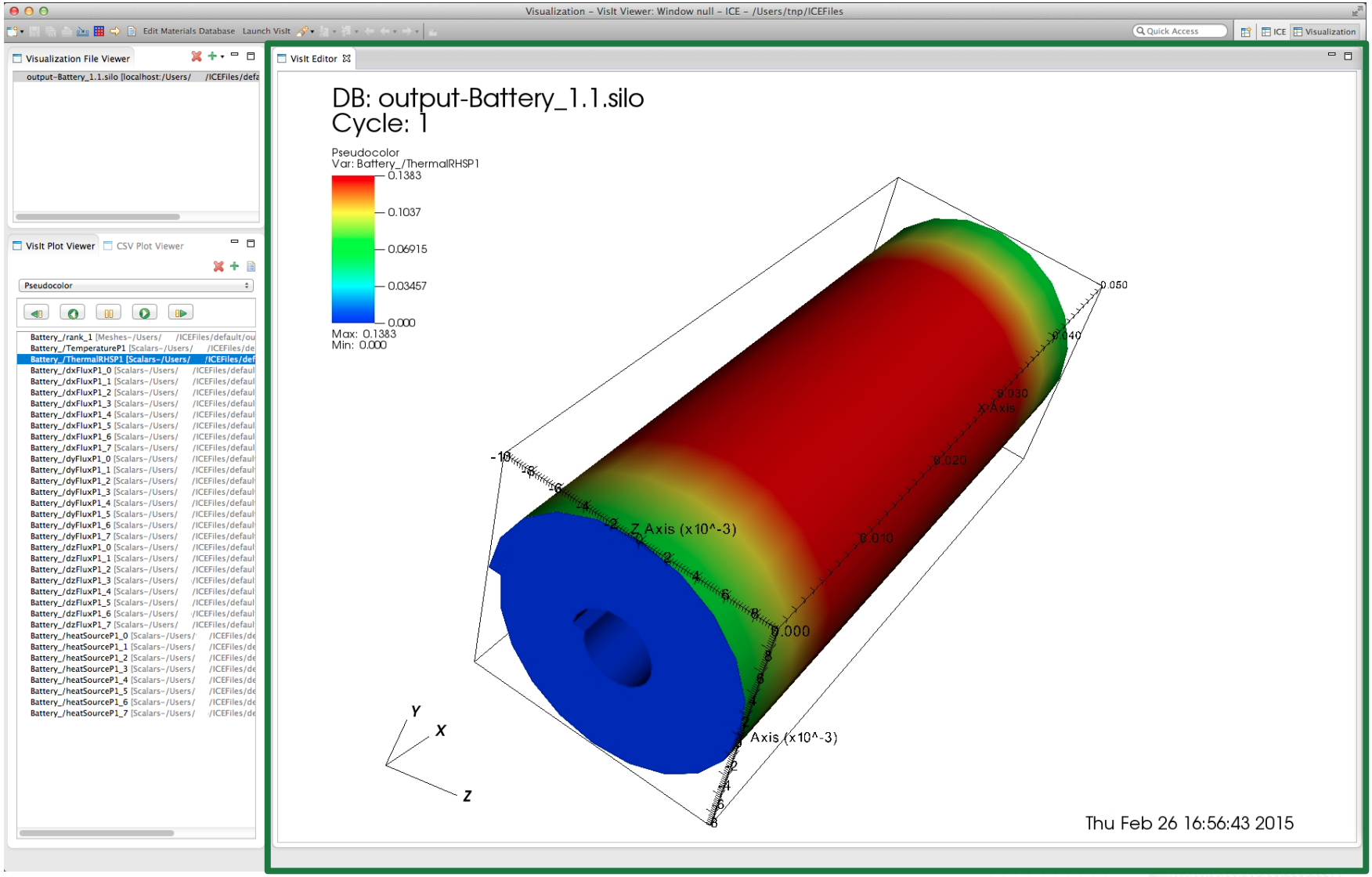
Data provided by VisIt

ICE Components – Viewers





ICE Components – Editor

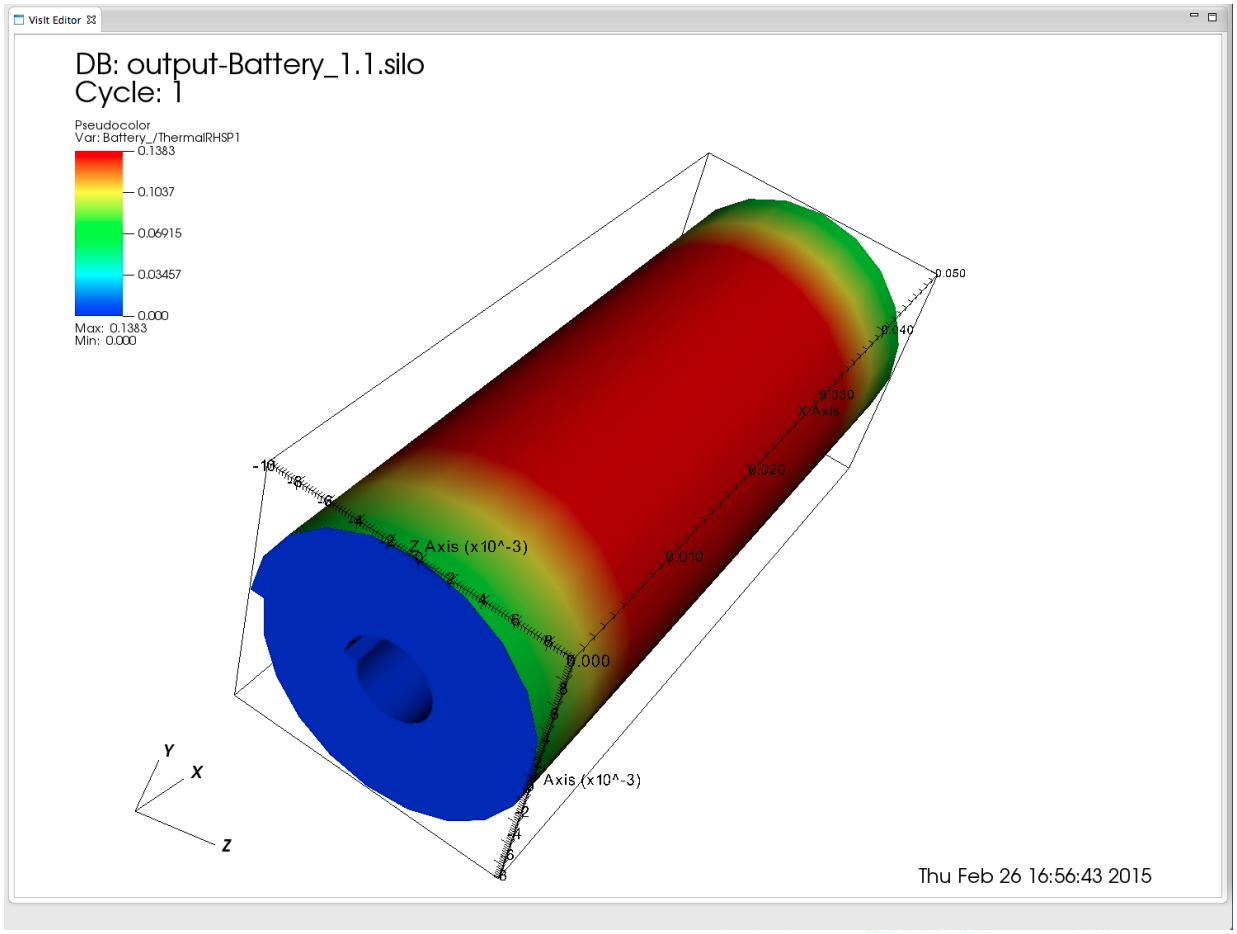


ICE Components – Editor

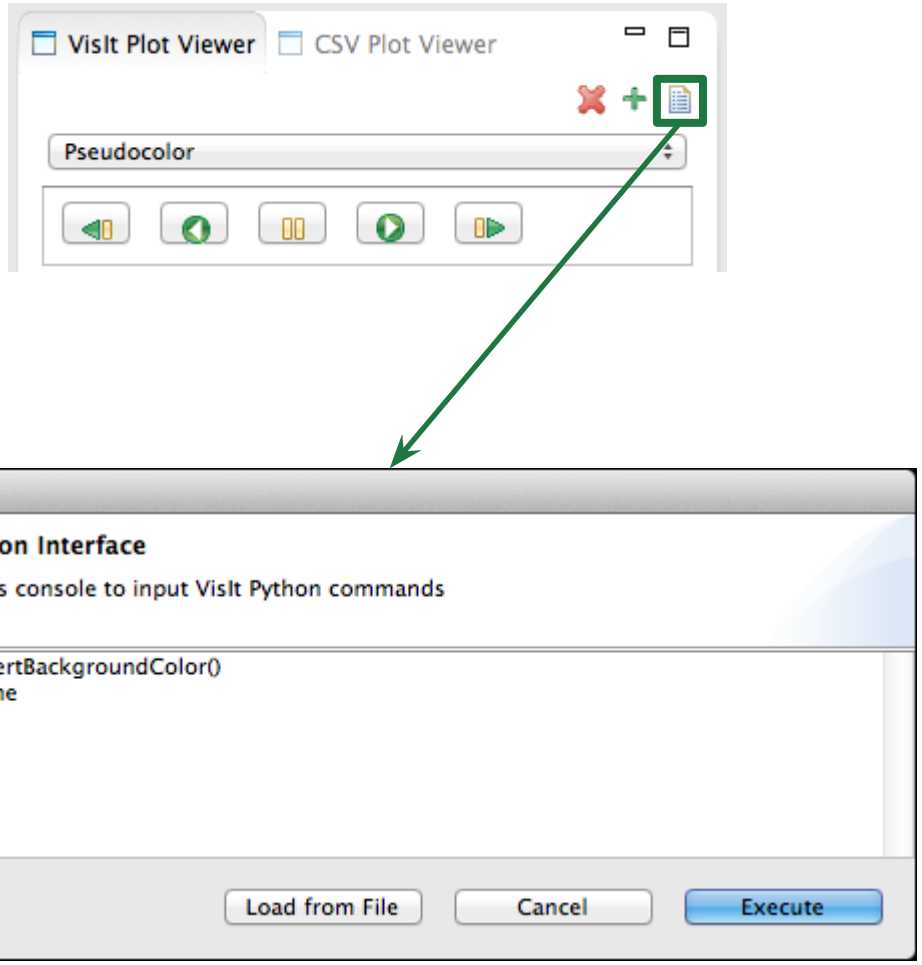
SWT Canvas subclass in an EditorPart subclass

SWT Image created from a ByteArrayInputStream

Canvas mouse listeners use atomic positions for daemon thread



ICE Components – Python Interface



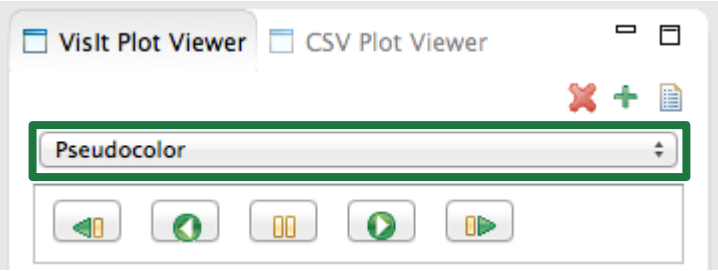
Immediate exposure of VisIt's rich Python API

Save a collection of operations and execute with only a few clicks

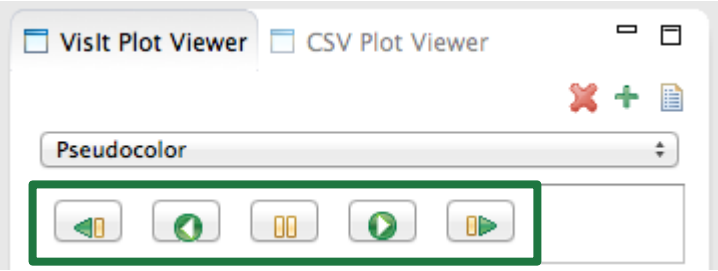
Automate frequently executed analysis workflows

Temporary dialog, moving to Console

ICE Components – Additional Features



Support for the full range of plot types provided by VisIt

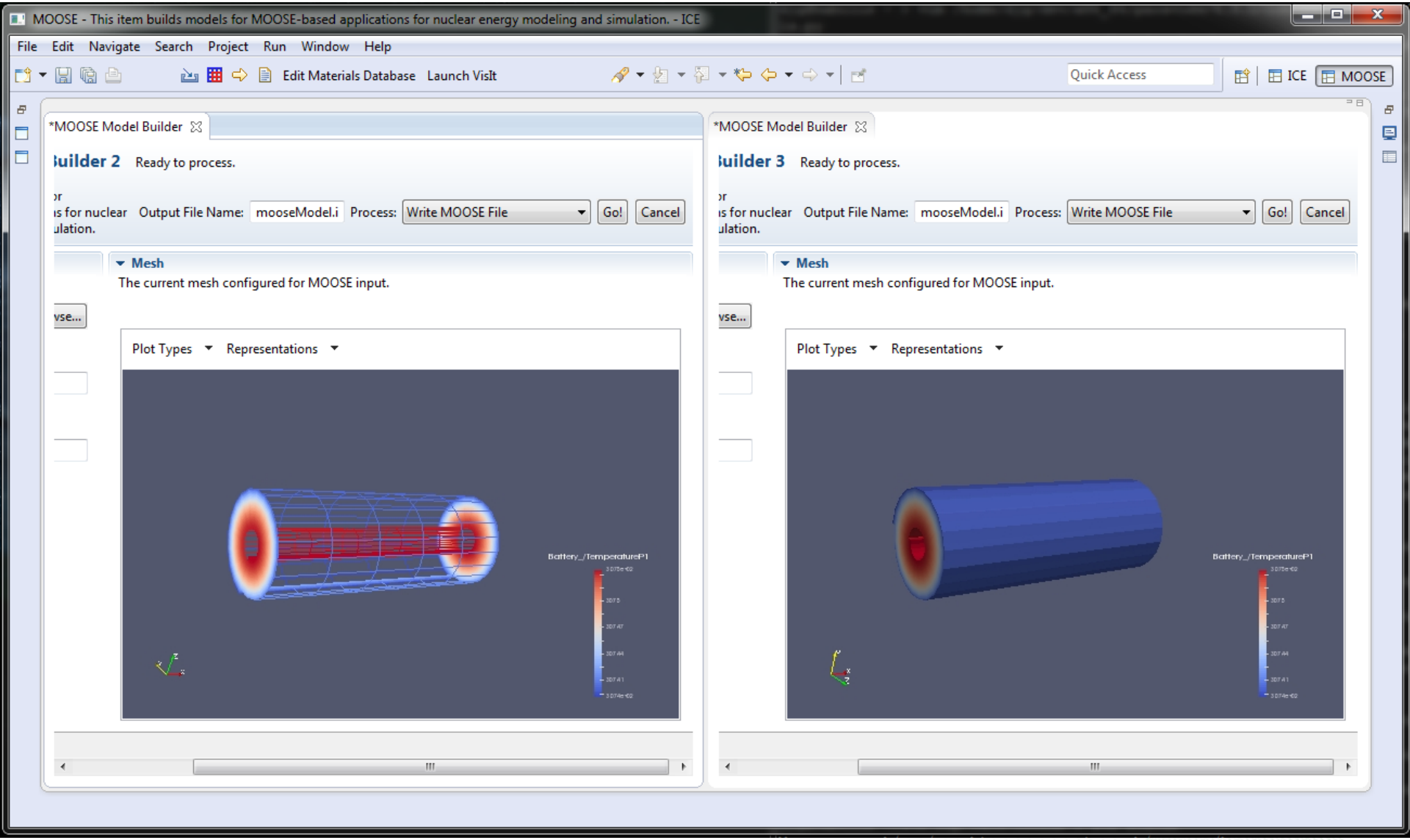


Animate time series data

Future Work

- Capabilities added to workbench
- Integrate other visualization tools - ParaView
- Extract interfaces and create an OSGi service
- Potentially spin-off a separate project
- Increase users and further collaboration with Science IWG (among others)

Future Work



Questions?

GitHub

github.com/eclipse/ice

Eclipse Wiki

wiki.eclipse.org/ICE

YouTube

youtube.com/user/jayjaybillings

Additional Authors: Andrew Bennett, Jordan Deyton, Hari Krishnan, Anna Wojtowicz

Author Email: mccaskeyaj@ornl.gov, pattersontc@ornl.gov, billingsjj@ornl.gov

eclipsecon 2015

Evaluate the sessions

Sign in: www.eclipsecon.org