



Investigating Graphical User Interfaces for the SECAD Project

David Alexander

Distributed Technologies, Tech-X Corporation

alexanda@txcorp.com

*SECAD – a Schema-based Environment for Configuring,
Analyzing and Documenting Integrated Fusion*



What Functions Can A Graphical User Interface (GUI) Improve for SWIM Users?

General benefit: Users need less “framework” skill/knowledge, lower cognitive load for mundane needs, higher physics productivity

Feature Examples	Description
Context-Sensitive Help	Display of pertinent information
Typed-Fields	Only allows user to enter correct type
Pull-Down Menus, Radio buttons, selection lists, etc.	Users prompted with complete set of choices (even better than starting with existing input)
Enabling/Disabling	Guides user to know what appropriate actions are for any given state

Important Questions:

- Who are the users? (current & targeted)
- What are their goals?
- What are their skills and experience?
- What are their needs?



Tech-X is Building Abstract Studio Has been applied to FACETS input

Configuration
"Text" Editor



Assistance
Around Editor



Run
Capability



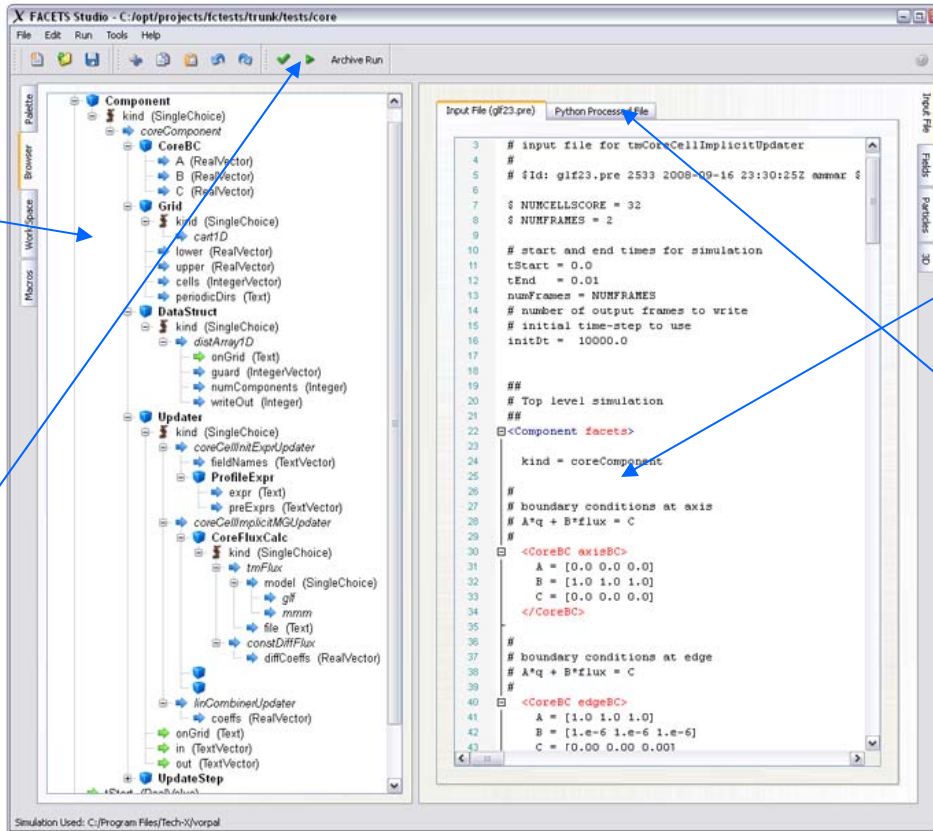
Visualization
Capability



Replace Editor
with Dialogs

Palette of
what can go
into
configuration
file at cursor

Local Run
control
working,
remote run
under
development



Color-coded
editor with some
validation
(parameter
name/type
checks)

Python Interpreter
used to add user-
defined variables
and "macro"
statements



XML Description of Configuration File Allows For Validation

Validation is Can Include:

- Absence of required parameters can be noted
- Unrecognized parameters can be flagged
- Default values can be filled in
- Parameters can be checked for section-by-section
- Type of parameter can be checked
- Parameters that reference system files can be checked that the file exists
- Parameters that are choices among specified list of values can be check that the value in the file is in the list
- Parameters that must reference other parameters can be checked for existence of that other parameter



Studio Flexibility Through Localization of Application Specifics & XML (1 of 4)

Making a New Studio Application...

- **Must create a new directory with a TxAppSpecifer.h file**
- **Must write set of XML description files (NOT Schema, since configuration files not XML)**
- Requires that developer has a complete set of example configuration files or that they must dig into code
- Validator assumes “block” structure, but python pre-parser is run first, so there is a chance to convert from any syntax before validation and users can interface through editor with native syntax (although highlighter must be customized)
- Has much needed benefit for complex hierarchically related configurations, but still has good value for flat structures (such as FORTRAN name lists)



Studio Flexibility Through Localization of Application Specifics & XML (2 of 4)

```
class TxsAppSpecifier {
public:
static QString name() { return "VorpalsStudio"; }
static QString organization() { return "txcorp"; }
static QString configDir() {
return QDir::homePath() + "/" + organization() + "/" + name() + "/";
}
static QString version() {
return "VORPAL-4.0.0rc5";
}
static QString configFile() { return configDir() + "/" + version() + ".conf"; }
static QString logFile() { return configDir() + "/" + version() + ".log"; }
static QString appDirPath() { return QApplication::applicationDirPath(); }
static QString baseSimDescriptionPath() { return ":/resources/inputdesc/vorpals/"; }
static QString absSimExecutable() { return "vorpals"; }
static QString absSimExecutableWin32() { return absSimExecutable() + ".exe"; }
static QString defaultSimPath() { return "/usr/local/vorpals/vorpals"; }
static QString defaultWin32SimPath() { return "C:\\Program Files\\vorpals\\vorpals.exe"; }
static QString simProcessName() { return "vorpals"; }
static QString testWindowFileName() { return "VORPAL Input Files (*.html)"; }
static QString allCapsName() { return "VORPAL"; }
static QString simDisplayName() { return "VORPAL"; }
static QString simConfigTemplatePreFile() {
return ":/resources/inputdesc/vorpals/VorpalsSimConfigTemplate.pre"; }
static QString descriptionsDir() { return ":/resources/inputdesc/vorpals/"; }
static QString aboutDialogTitle() { return "About VORPAL Studio 4.0"; }
static QString aboutDialogText() { return "<p><b>VORPAL Studio</b> helps users configure and " \
"run VORPAL Electromagnetic Simulations " \
"by assisting in input file validation.</p> " \
"<p>For more information see: " \
```



Studio Flexibility Through Localization of Application Specifics & XML (3 of 4)

```
<Component facets>
kind = coreComponent
# BCs: A*field + B*flux = C
<CoreBC axisBC>
A = [0. 0. 0. ]
B = [1. 1. 1. ]
C = [0. 0. 0. ]
</CoreBC>
<CoreBC edgeBC>
A = [1. 1. 1. ]
B = [1.e-6 1.e-6 1.e-6]
C = [0. 0. 0. ]
</CoreBC>
<Grid grd>
lower = [0.0]
upper = [1.0]
cells = [32]
</Grid>
...
```

```
<Block name="Component">
<!--description-->
<Description>Component Block</Description>

<!--kind definition-->
<SingleChoice name="kind" default="coreComponent"
minOccurs="0">
<Description>kinds</Description>

<Choice name="coreComponent">
<Description>Component to model the
plasma core.</Description>
<Block ref="CoreBC" />
<Block ref="Grid" />
<Block ref="DataStruct" />
<Block ref="Updater" />
<Block ref="UpdateStep" />
</Choice>
</SingleChoice>
</Block>
```

Component.xml

```
<Block name="Grid">
<!--description-->
<Description>Grid Block</Description>

<!--kind definition-->
<SingleChoice name="kind" default="cart1D" minOccurs="0">
<Description>kinds</Description>
<Choice name="cart1D"/>
</SingleChoice>

<!--Contained Blocks/Attributes -->
<RealVector name="lower" default="[0.0 0.0 0.0]">
<Description>lower</Description>
</RealVector>
<RealVector name="upper" default="[0.0 0.0 0.0]">
<Description>upper</Description>
</RealVector>
<IntegerVector name="cells" default="[0 0 0]">
<Description>cells</Description>
</IntegerVector>
<Text name="periodicDirs" default="[]">
<Description>periodDirs</Description>
</Text>
</Block>
```

Grid.xml



Studio Flexibility Through Localization of Application Specifics & XML (4 of 4)

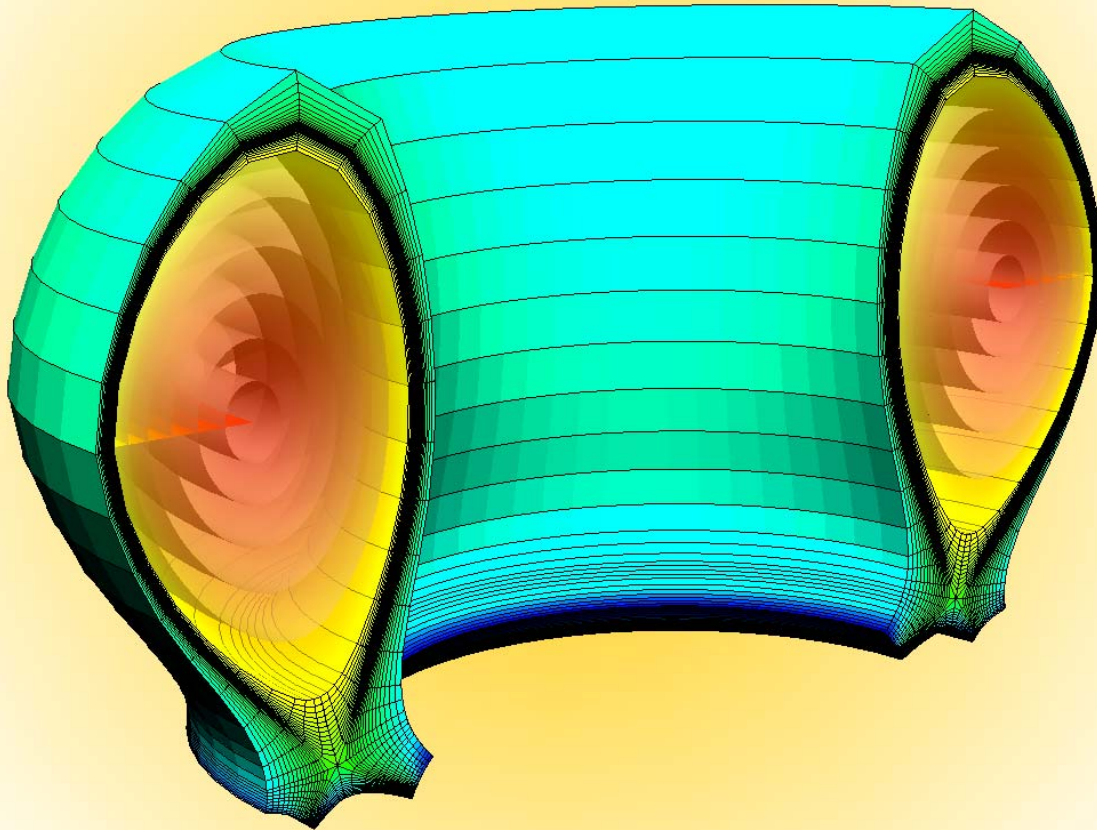
Making a New Studio Application...

- Must create a new directory with a TxAppSpecifer.h file
- Must write set of XML description files (NOT Schema, since configuration files not XML)
- **Requires that developer has a complete set of example configuration files or that they must dig into code**
- **Validator assumes “block” structure, but python pre-parser is run first, so there is a chance to convert from any syntax before validation and users can interface through editor with native syntax (although highlighter must be customized)**
- **Has much needed benefit for complex hierarchically related configurations, but still has good value for flat structures (such as FORTRAN name lists)**

Current Visualization for FACETS done with VisIt. Takes Many Steps. The Goal is to Automatically Produce Default View in Studio

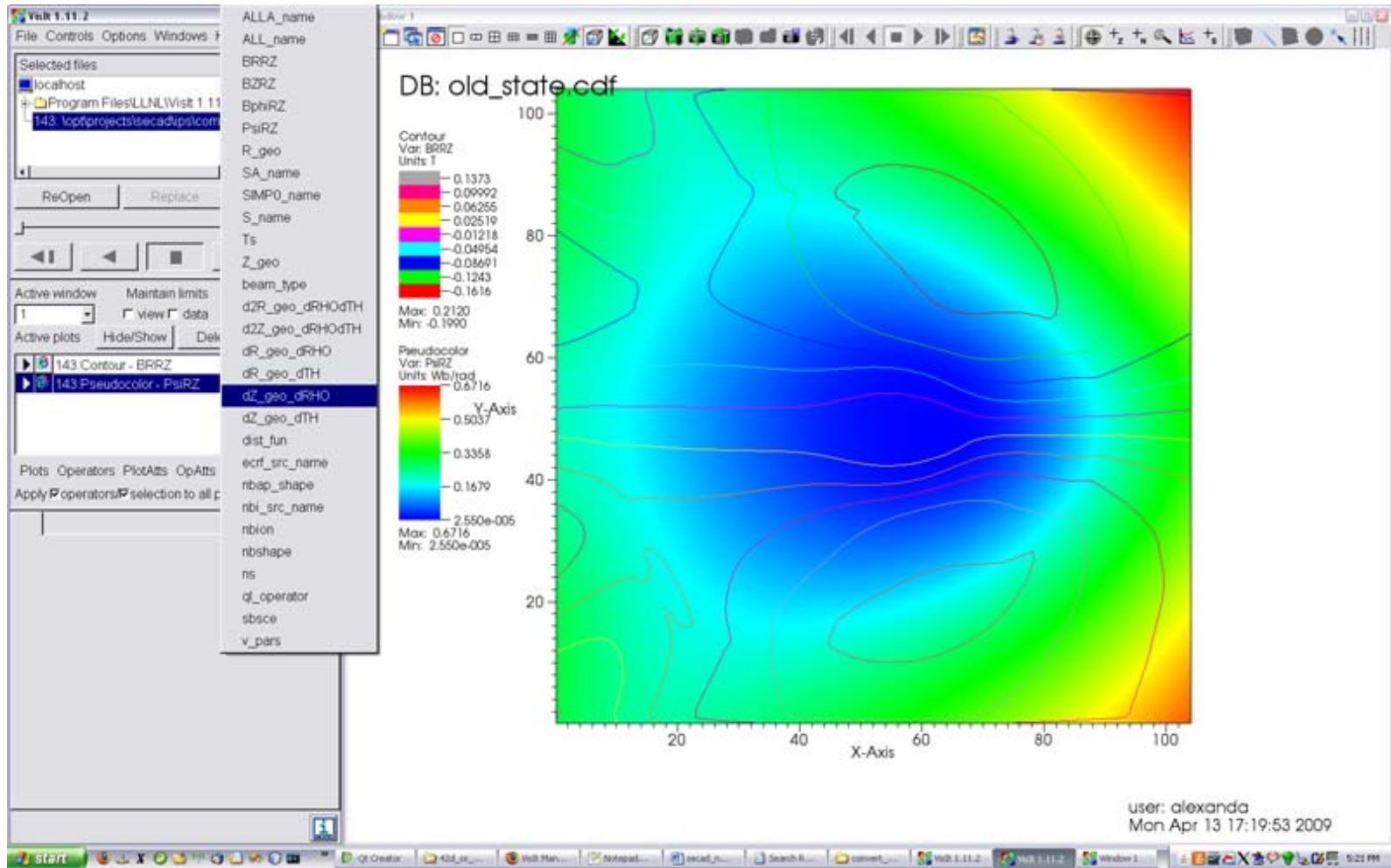
Actions to produce this plot

- Open several files
- Choose correct types of plots and variables.
- Choose operators (revolve, iso-surface, contours, etc.)
- Adjust plot parameters (pseudo-color, axis limits, etc.)



Made with the VisIt Visualization Tool, see <https://wci.llnl.gov/codes/visit/>

We Started Investigating Whether VisIt can Visualization SWIM Plasma State



user: alexanda
Mon Apr 13 17:19:53 2009



Default Visualization Gives Efficiency and Low-Threshold for New Users

POWER NOT ALWAYS GOOD...

- Visualization tools can be very powerful, but as we have seen there are often many action steps to useful views. High learning curve for new users.

NO BLANK SCREEN...

- Often there are common visualizations that users will want/need to do rudimentary assessment of simulation output. These views make good defaults.

SAVE THE USERS TIME...

- Initially can save users time even if they later want to explore using more of the richer capability of the visualization tool.

GOOD OPPORTUNITIES ARE AVAILABLE...

- Studio written in Qt to be able to (if desired) embed VisIt display widgets. If VisIt is compiled, then user will experience one application (i.e. the Studio) and not need to download or install VisIt separately. Remote visualization would require VisIt installed on server co-located with visualization cluster.
- Markup of data file (possible with self-describing formats) allows for expression of data structures by simulation or framework developer, which the studio can use to make default plots (users ultimately benefit from developer's experience).



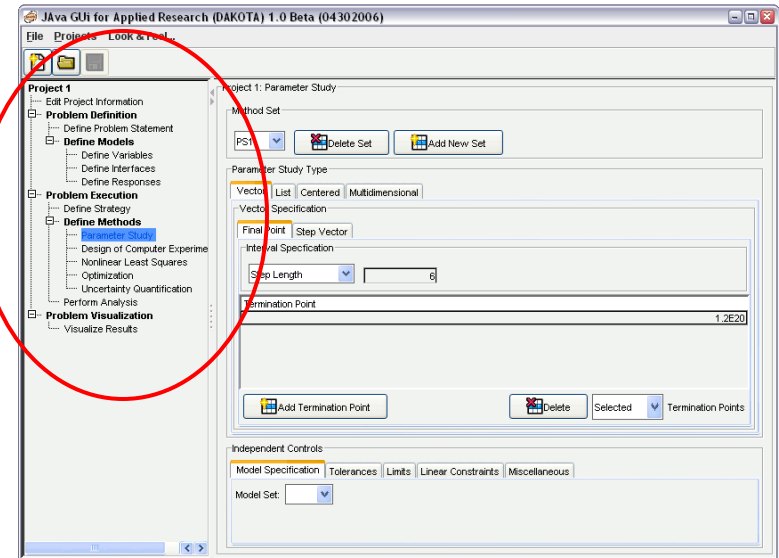
DAKOTA GUI

(Design Analysis Kit for Optimization and Terascale Applications)

```
1 # DAKOTA - configuration file for a parameter study on IPS
2
3 strategy,
4   single_method
5   tabular_graphics_data
6
7 method,
8   vector_parameter_study
9   final_point = 1.2e20
10  num_steps = 6
11
12 model,
13   single
14
15 variables,
16   continuous_design = 1
17   initial_point 0.8e20
18   descriptors 'ne0'
19
20 interface,
21   system
22   analysis_driver = './ips_wrapper.sh'
23   parameters_file = 'params.in'
24   results_file = 'results.out'
25   file_tag
26   aprepro
27
28 responses,
29   num_objective_functions = 1
30   no_gradients
31   no_hessians
32
```

strategy
method
model
variables
interfaces
responses

Creates dakota.in file



GUI organized like dakota configuration file
Very similar in concept to LCML GUI



Concluding Questions....

- Assuming SWIM wants a GUI, do you envision running GUI in client/server mode? Or local (including over X). SWIM Portal is working good for monitoring. Does it lack any features?
- Do you want VisIt visualization? Integrated with GUI?
- Will SSH work as communication mechanism?
- Currently ElVis monitors simulation in 2D. Is 3D visualization useful?
- Data formats? We know netCDF for plasma state. Others?
- What are the visualization that are needed? Are the ElVis templates pretty good? If not, what are good default visualizations?
- IPS Framework has users writing drivers in Python (with errors not caught until runtime) – how can these be caught before simulation is submitted?



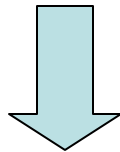


APPENDIX

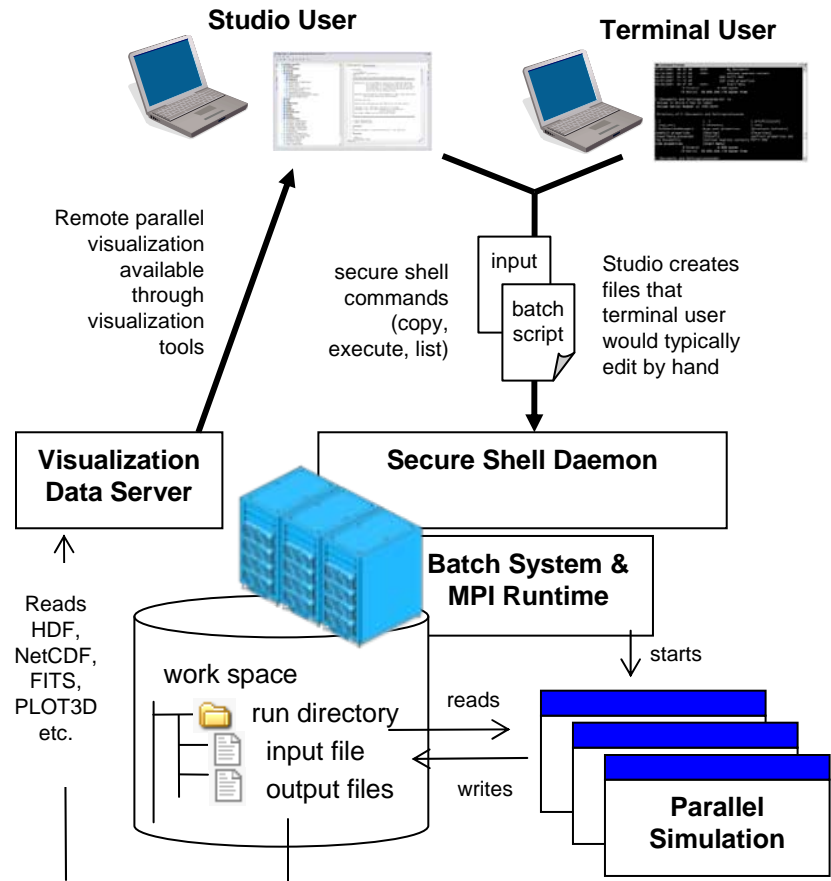


Concept of Remote Work Space & Following Power User Workflow

1. Simulations are run remotely and are often organized manually into directories (or work space)
2. GUI is helpful for creating configuration files
3. GUI can be run remotely, but SLOW!



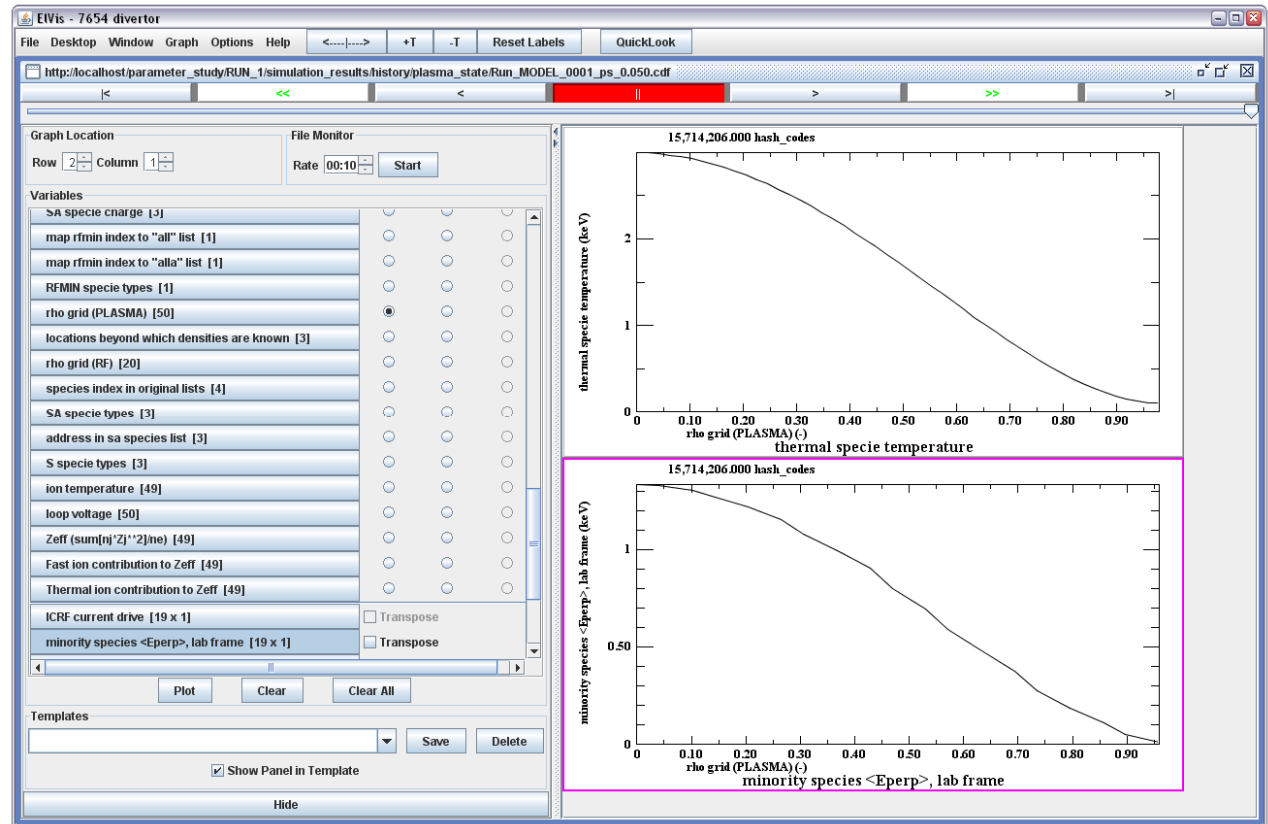
Distributed Solution Is Appropriate





ElVis Visualization Tool

- Plots of netCDF files
- HTTP access to remote files



DAKOTA Framework

