Systemtap GUI: an IDE for systemtap

Jeff Briggs <jhbriggs@us.ibm.com>
Henry Hughes <whhughes@us.ibm.com>
Ryan Morse <rmorse@us.ibm.com>
Hien Nguyen <hien@us.ibm.com>

April 14, 2006

Table of Contents

1. Overview

Development with systemtap is currently text based and lacks ease of use. Additionally, there is no easy way to view predefined functions or tapsets. This presents a significant limitation for development efforts.

Systemtap developers wish to create a graphical IDE to facilitate the use and adoption of systemtap. Another goal is to make the development of functions and tapsets easier for developers.

This document describes the requirements for this system.

2. Requirements

2.1. Reliability

The GUI must be able to handle any errors generated by users. Also, any code created or modified must be semantically and syntactically correct such that it can be run using the stap command.

2.2. Ease of Use

The interface should be similar to Eclipse and as easy to navigate as possible. It should allow for the addition of existing functions and tapsets.

2.3. Performance

The interface should instantly respond to any user request. Loading and execution of scripts should also be accomplished in a timely manner. In situations where long response times are unavoidable (running stap with a large module), a visual cue will be provided to inform the user that something is happening.

2.4. Simplicity

The software, tests, and documentation undertaken by the Speed Team must be able to be developed in a short amount of time.

2.5. Flexibility

Features must be easily modifiable to accommodate different users preferences.

2.6. Scalability and Extensibility

As systemtap evolves, this interface must be able to easily scale with it. The code should be easily modifiable in order to accommodate additional requirements and features.

3. Script Development Environment

The central part of the application is the code development perspective. This is where all script design will take place. See example design below:

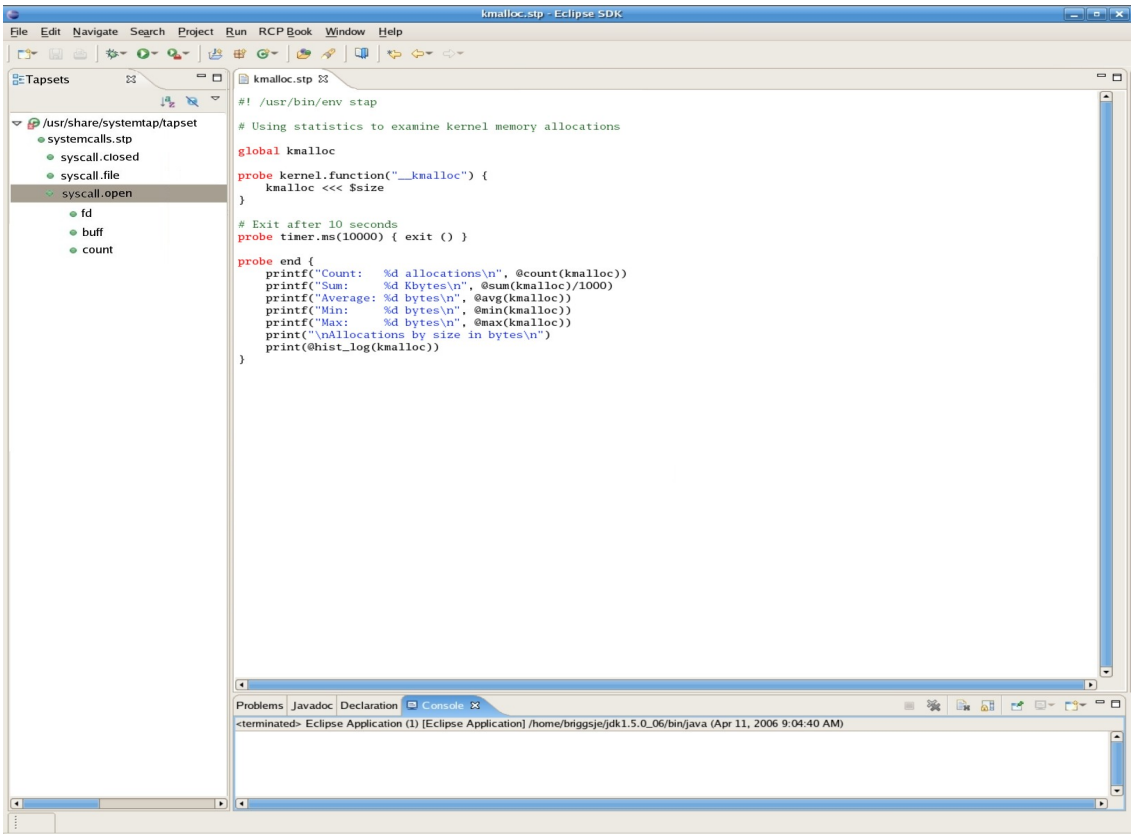

Figure 1: Script Development Environment

3.1. Script Language Editor

The main window in the perspective will be devoted to a script editor. This editor is where all the code will be placed and is the main focus of the GUI.

3.1.1 Syntax highlighting

Like other IDEs, the systemtap GUI will provide syntax highlighting for ease of development. Keywords , comments , and embedded C code will all be highlighted. Each of these will be colored differently and allow for user customization.

Syntax highlighting will follow the standard used in Eclipse. Keywords will be purple, strings will be blue, comments will be green, and embedded C code will be a dark gray.

In addition to being highlighted, embedded C code will also be checked for unsafe dereferencing of pointers. Since this can cause serious errors in the code if the developer is not careful, a small warning flag will mark each potentially unsafe occurrence. This warning flag is will be placed on the far left of the editor and is for reference only, it will not affect execution of the script.

3.1.2 IntelliSense

In order to reduce the amount of time required to develop scripts, as well as to reduce the number of errors in scripts, the editor will employ IntelliSense. Developers who do not like this feature will have the option to disable it. When enabled, developers see helpful dialogs with the available functions and variables that are relevant to what they are currently working on.

3.1.3 Code errors

The editor will employ just-in-time debugging in order to help the user reduce the amount of time they have to spend debugging their code. This is accomplished by periodically running translator, making sure the code is valid. Errors will be marked for attention but will not force the user to deal with them immediately.

3.1.4 Existing Scripts

In addition to being able to develop new scripts, the editor will also be able to open existing *.stp files. These files will be treated exactly the same as files created using the GUI to begin with.

3.2. Kernel Source Browser

Because looking at code and counting line numbers is prone to errors, the systemtap GUI will provide an easy Kernel Source Browser. This window will allow the developer to look through kernel code and simply double click on the line they wish to probe. Doing so will leave a marker beside the line they clicked on as well as insert a prototype probe at the bottom of the script code.

A commented list of available variables will also be added to the probe prototype. This saves the user from having to search to find what variables they have access to.

3.3. Library Browser

A window will be setup to list all predefined items in existing tapset files. By default files in the /usr/share/systemtap/tapset directory are included, however, the user can modify this default directory as well as import additional directories for use. In addition, all items in the current file will be added to the tree.

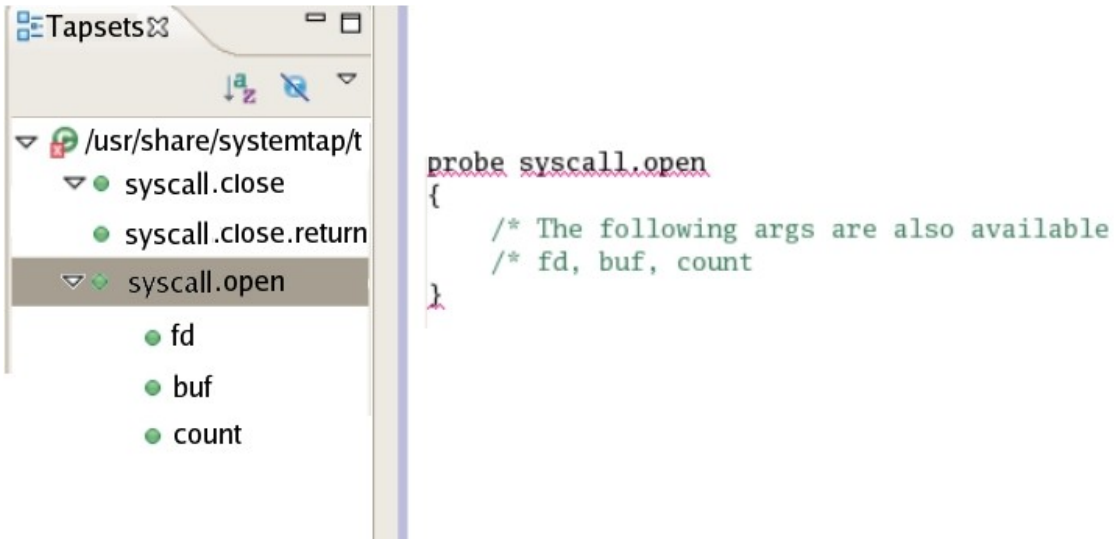

Figure 2: Library browser example

3.3.1. Function Browser

A tab will be in the library browser window for users to scan through predefined functions. All functions that are available will be listed in a tree view. Items will be organized based on Folder->File->Function. Developers can double click on any function to insert a call to that function at the current cursor location. In addition to the tree view, a search feature will be developed for users to quickly locate functions.

3.3.2. Tapset Browser

Like the Function Browser, a tab will be in the library browser window for users to scan through predefined probe aliases. All probe aliases that are available will be listed in a tree view. Items will be organized based on Folder->File->Alias. Developers can double click on any tapset to insert a call. New references will be placed at the bottom of the code editor. In addition to the tree view, a search feature will be developed for users to quickly locate probe aliases.

Each probe alias in the tree will also include all of the available local variables that can be accessed with that probe. When a developer double clicks a probe to include a call the available variables will be added in a comment block. Additionally, variables can be double clicked and added as a reference at the current cursor location.

```
probe syscall.read {
        /* The following args are available for this probe    */
        /* fd, buf, count                                     */
}
```

Figure 3: Example inserted probe

3.4. Conditional Filter Selector

This window will provide a list of commonly used conditional filters. Developers can easily double click on a conditional statement to have it added to the script at the cursor location.

3.5. Traces

Traces are simple scripts with print statements that are to run and to display output to the user without having to do any additional configuration.

3.5.1. Trace Builder

This feature will allow developers to bundle actions that occur often into a simple package that can be run at a later time. This greatly reduces the time required to run common tests. Developers create or use a simple script that inserts print statements at desired points to see what is happening. They can then select how they would like to have the output displayed. These preferences can then be saved for reuse later.

3.5.2. Trace Browser

This browser allows developers to look through traces that have been defined in the past and easily select and run one. This removes the time needed to load up a script and select the desired output conditions.

4. Output Display

When a script runs, the output has traditionally been printed out to the terminal window. However, with the development of the systemtap GUI, developers will now have multiple options on how they wish to view the output. These display methods are available only when scripts are run using the systemtap GUI.

4.1. Raw Data Display

This display method is similar to the terminal window. It is still in raw text format. The only difference is that instead of a terminal output, text is displayed in the systemtap GUI instead. This display can be used to show details for both a previously run probe as well as one still in progress.

4.2 Chart/Graph View

The Chart/Graph View option allows developers to see a chart or graph of the statistics their script generated. This can be much easier for seeing how variables and functions behaved than the raw text view. Graphical elements are created using the TPTP charting service. Any desired functionality not available in the API will be developed.

A new tapset will be developed that specifies functions that will format data in such a way as to allow the systemtap GUI to recognize and plot it. This allows developers to specify what they want to have charted in the script itself. This also allows for multiple different charts to be created in the same script.

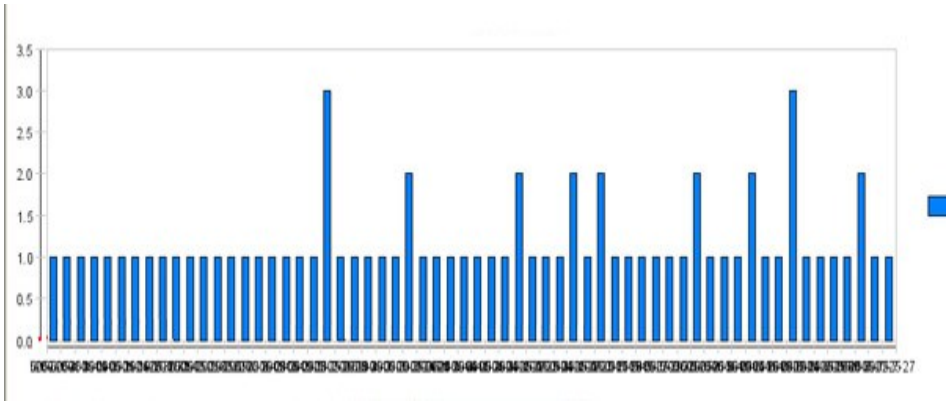

Figure 4: Example display of running program

4.3. Dynamic Line View

The Dynamic Line View is a way for developers to view what is currently happening to values that they are tracking in their script. The view will update itself at a frequency specified by the user. This is displayed as the script is currently running as opposed to the Chart/Graph view that shows data from a previously run script. The Dynamic Line View will also be implemented using the TPTP charting service.

4.4. Event Notification

This option is used to notify the developer upon specific event occurrences. When a predefined even occurs, a message is sent informing the developer. These events are specified in the systemtap GUI, not in the script itself.

5. Documentation

All features of the systemtap GUI are clearly documented in the help section of the application. Any information a developer is interested in should be provided here. All documentation will be compatible with the Eclipse Help engine.

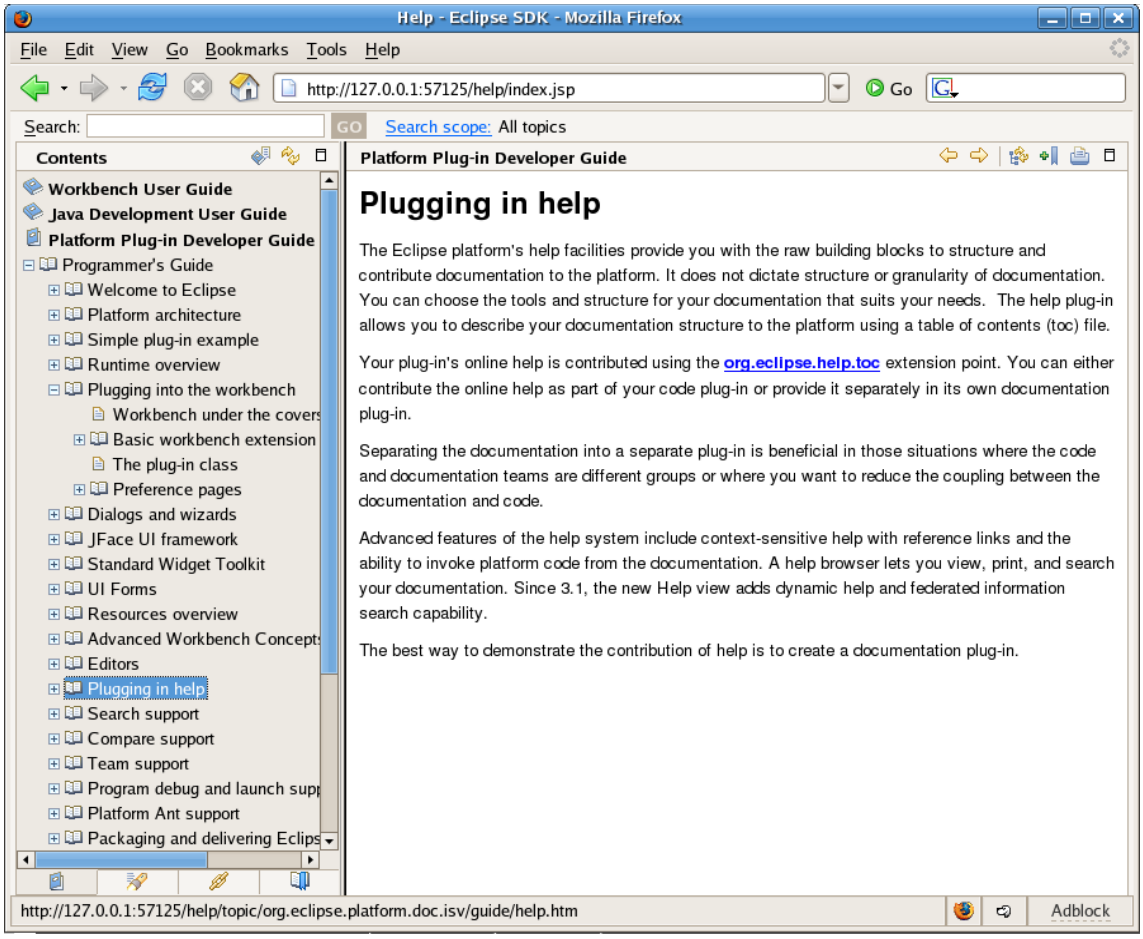

Figure 5: Help display using Eclipse Help engine

5.1. Help

In order to provided information to the developer, a sophisticated help document will accompany the systemtap GUI. Frequently asked questions should be answered by this document in an orderly manner. All features and functionality should be explained. It is assumed that readers will have a **basic** understanding of both systemtap and kprobes before working with the systemtap GUI.

5.2. Tutorial

In order to get developers comfortable with the GUI as fast as possible, tutorials will be provided to walk them through each of the features. By the end of the tutorial, users should be comfortable using the GUI for all of their development and analysis.

6. Testing

The entire Systemtap GUI will be extensively tested for correctness. Testing will be done at three different levels.

6.1. JUnit

After each component is developed JUnit tests will be created for it. These tests will ensure that the newly created component is working as desired. Each method inside the component will be tested multiple times. All standard, and boundary conditions will be tested.

6.2. Rational

Once the application has been developed, we will begin using Rational to test the entire application. This testing will help detect issues with component integration and performance.

6.3. TPTP

Also, upon finishing development further testing will be done with TPTP (Tracing and Profiling Tools Project). Like Rational, this testing suite will help locate and resolve performance bottle necks and issues with component integration.

7. Stretch Goals

These tasks are enhancement features that will be developed if time permits. Items here, while not necessary for a working GUI, will provide the user with advanced features.

7.1. Performance Analysis

The Performance Analysis tool will be similar to Sun's Performance Analysis tool. Its purpose is to allow developers to easily see how resources are being consumed. This can provide significant help in determining what can be changed in order to reap the most benefit. Performance can be based on CPU usage, memory usage, or the amount certain things are accessed.

This tool is not built into the systemtap GUI itself, but will be a separate application. The data will be a collection of output from the kernel over some time period. It can act as a summary tool, grouping together results of a number of different probes.

7.2. User Customization

In order to make the IDE more user friendly, coding will be done such that users will be able to adjust settings based on their preferences. Settings such as syntax highlighting, window arrangement, and display properties will be available for customization.

7.3. Export Charts

Once a chart has been generated and displayed to the user a button will be provided to enable them to export all the chart data. The exported data will be stored in XML format. This will allow users to store just the data needed to generate the graphs.