VERSIONED LEXICAL SEARCH INTERFACE

Capstone Project

Alex Garcia
Candidate: Masters of Science in Computer Science
University of Wisconsin-Milwaukee
Fall 2012
Under the advising of Dr. Ethan Munson
In collaboration with Michael Haufe

Abstract

This paper is in partial fulfillment of the degree requirements for the Masters of Science in Computer Science at the University of Wisconsin-Milwaukee. It reports the processes taken to complete the capstone project, which was the development of a Graphical User Interface for a version Lexical Search.

The user interface requirements, prototypes, functionality report, and goals for future version releases will all be discussed.

Introduction

Semantic Designs software develop a variety of tools which allow for the analysis and modification of large scale software. Among these tools are quality analysis (e.g., metrics, test coverage, clone duplication) , problem analysis (e.g., dead or erroneous code), software migration (e.g., porting between platforms, and software language conversions) and lexical search engine. The latter is the focus of this capstone project.

The goal of a lexical search engine tool is to improve the accuracy of a search within a program. "Lexical search" means that the search knows about the syntax of the programming language, so users can search for all the occurrences of "foo", but only where it is the name of a function in a function call. These search engine tool is able to achieve a very high performance by first processing all of the files, and then creating a set of indices. Although this process is time-consuming, it only needs to be done once if the files are not changed.

The aim of this capstone project is twofold. First, it will produce a design for a practical user interface to do a versioned lexical search. Second, it will implement as much of this user interface as is practical, with the goal of creating a reasonable “proof of concept” for a versioned lexical search. Rather than requiring that the developer remember to create the indices, the system will automatically create the indices incrementally. Not only would this make it much easier to do searches on short notice, but it would also permit cross version searches, for example, "find the earliest version where the 'foo' function was present". This tool can be useful if a user wishes to identify a specific older version of the system that needs to be rebuilt, or to connect implementation changes with related historic documents about the system.

This document will take you through four phases of development. First, there is a discussion of the brainstorming process during the user interface requirements stage, which has a brief discussion of all of the requirements that needed to be taken into consideration before starting the development of the program. Next, is the prototype phase, in which the different models and mockups used in the beginning stages are discussed. Following the prototype section is a detailed functionality description of the current application. Here, pertinent information about the program, such as how users can perform searches and navigate through the main display windows is explained. Finally, there is a short discussion of improvements which can be made to future releases of this program.

USER INTERFACE REQUIREMENTS

Before beginning the development of the new user interface, a requirements phase was addressed. The goal of this requirements phase was to collect a list of improvements in which the functionality of the current UI could be enhanced. Below is the list of the requirements collected during this phase.

Improve the User Interface

The Semantics Designs lexical search engine runs by using the command prompt. The tool comes with a user interface that is not user friendly. For instance, the query and results screens are not intuitive-most likely a user would have to run several tutorials in order to figure out where to enter data and how to read the results. The new application should present the user with an improved interface that focuses on the intuitive decision process when working with repositories and lexical searches. Ideally, the interface should include a way to graphically represent the search results, and an easier way to access the files, for instance, a file browser.

Automated Configuration

In order to do a lexical search, the user needs to follow a series of steps to generate multiple files which are required to properly run the search engine. The application should do this as an automated process, and generate all the required files.

Repository Communication

The application should be able to manage the transactions required to interface with the repository. The interaction from the user should be at a minimum.

Repository Search Capability

The lexical search engine was built to only run local searches. However, the application should be able to run searches locally, not just in the current project files but also in all the files associated with the repository.

Text Editor Utility

The text editor utility should have the ability to highlight the syntax and the search results. This would allow the users to quickly identify the differences between searches. Also, it should include features that are commonly used such as line numbers, and a way to copy, and cut the selected text.

File Comparison Utility

After doing a search over the repository the user should be able to do a comparison between files from different revisions.

PROTOTYPE AND DESIGN DESCRIPTION

Initially, the project started as a standalone application to be developed in Visual C#. However, promptly after discussing the design approach, Dr. Munson and I agreed that the best way to implement this project was by writing a plugging for the Eclipse IDE. This decision was made based off of the thought that it would be easier to later integrate the program into Eclipse IDE. Moreover, after doing some research and looking into the details on how to develop a plugging, this approach was then replaced by developing the application as a standalone version using the Java language and the SWT libraries for the GUI design. The change to writing the program in Java was made in order to smooth the progress of future improvements given that Java is more commonly used that C#. Furthermore, the change to Java was beneficial given that it would ease the future transition to the plugging platform used in the Eclipse IDE.

Prototyping

In the beginning of the prototyping stage, low fidelity prototypes were used., This decision was made based off of the flexibility and the simplicity that the low fidelity prototypes offer in representing the ideas in the UI development process. The software used to design this prototypes was Balsamiq. This software comes with a variety of controls and tools, such as combo boxes, buttons, labels, etc. which were needed during the initial stages of this project. The following sections describe some of the prototypes created during this project stage:

The first design included the concept of multiple windows which the user could switch between by selecting their desired screen from the drop-down menu "Windows". Figure 1 represents the prototype used for the main screen. In this prototype, some of the fundamental controls for the software can be seen, for instance, the different labels which display the information about the current project. The *feedback principal* was considered when designing this control. The idea behind this was to display as much information about the state of the search and the system, without overloading it with too many buttons and text. There are also controls to access the search result and the configuration of the application.

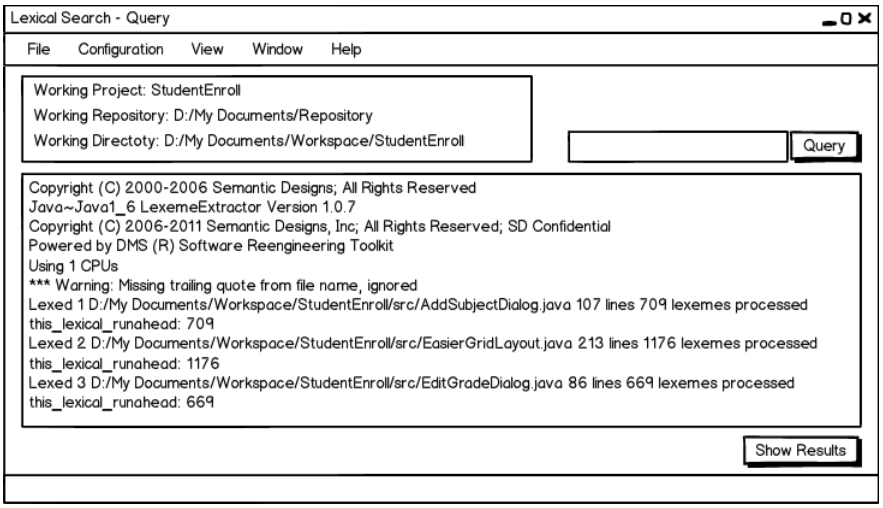

Figure 1: Main Screen: Original prototype

This second prototype presents a screen which displays the search results as a revision graph. There are three levels which were used to place the nodes, depending on the repository content: Tags, Trunk, Branches. The last level displays the time associated with the revision creation.
This control was designed with the *simplicity principle* in mind. The information is shown in a meaningful and user-friendly manner. Each node in the graph has a tooltip listing the files containing a match for the current query. Figure 2 displays a mockup for the second prototype.

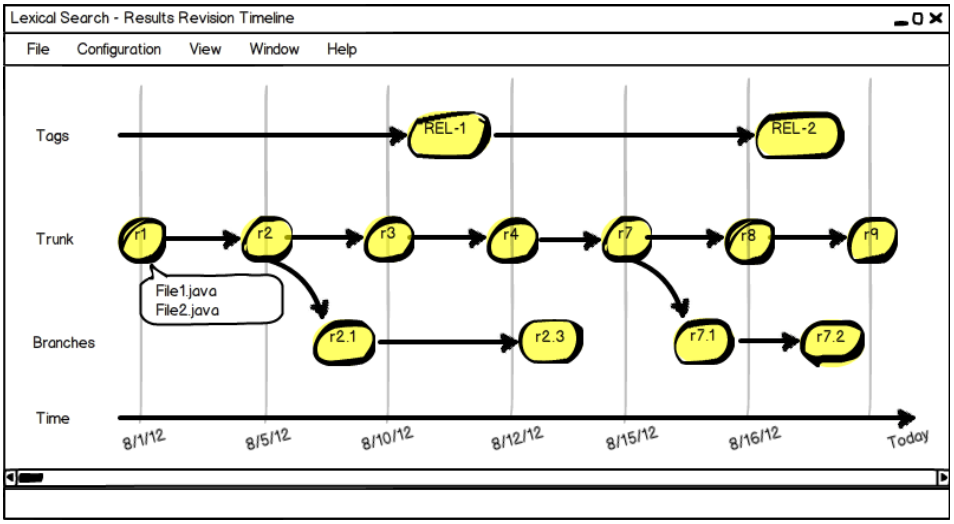

Figure 2 Revision Graph Window

The screen shown in Figure 3 represents the mockup designed to display the information captured by the search, used in the third prototype. Some of the main components are the file viewer, and the file browser. This section of the user interface was designed with the *structure principle* in mind. By using the file browser style to display the information related to the search, the user is able to access the files in a way that is commonly used in operating systems such as Windows and Linux. The text viewer is used to display the files which the user had selected from the file browser. The file viewer also provides access to tools such as *Copy*, *Cut* and *Select All*, frequently used in any word processor.

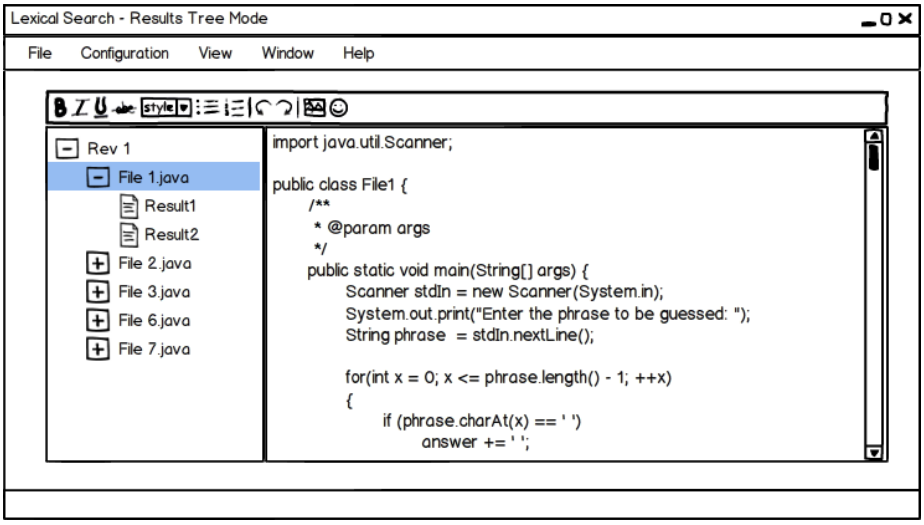

Figure 3 Text Viewer Window

The mockups previously described were part of the first design. In this design, each window was separately opened, and in order to change from one view to another, the user had to use the "Windows" drop-down menu. As mentioned before, this design was replaced by the plugging approach. The next sections show the final low-fidelity prototypes used in the development of this application.

The plugging approach combines all the previously described prototypes into one simplified user interface, facilitating the access to all the windows. By using a tabulated control, the user can change between views without switching windows. These changes were made in order to follow the *reuse design principle*. See figures 4-5.

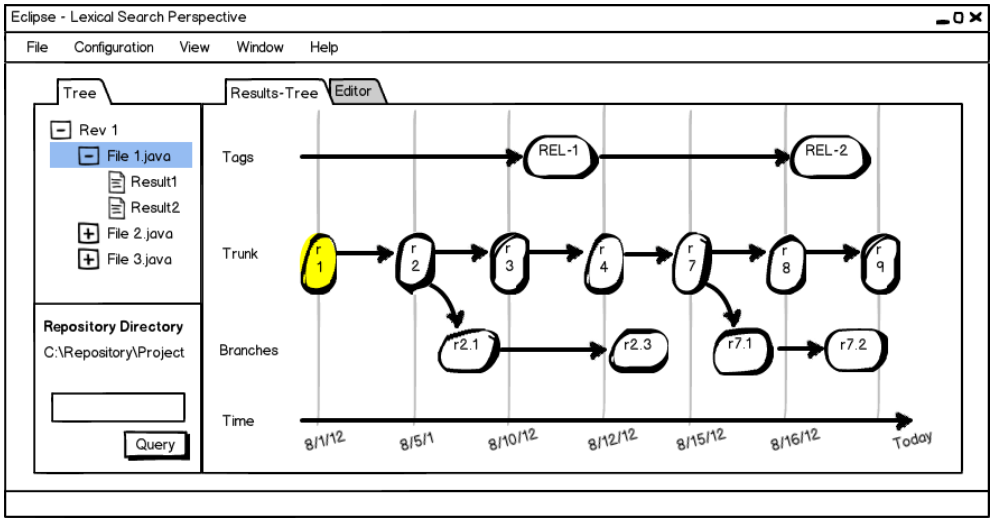

Figure 4 Tabulated Main Screen

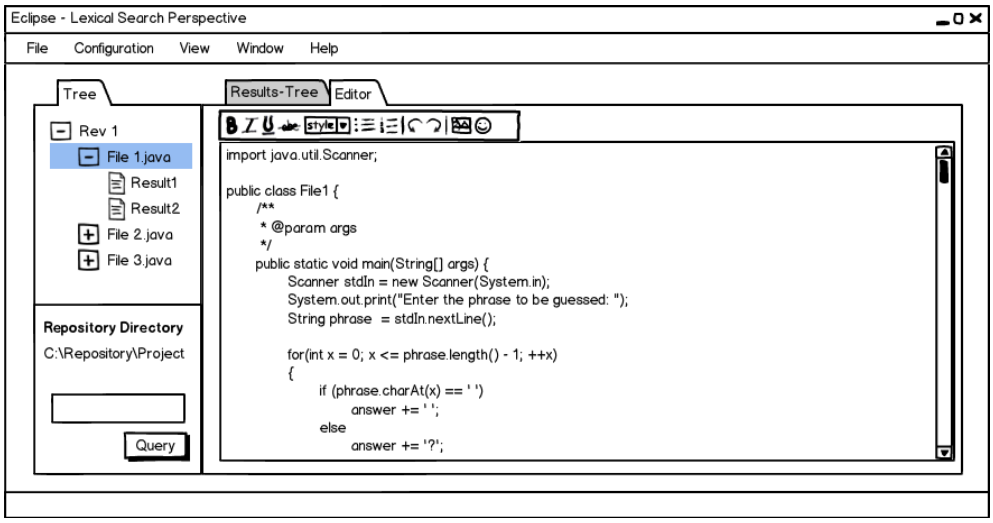

Figure 5 Tabulated Text Editor

FUNCTIONALITY DESCRIPTION

The following sections will provide a detailed description of the lexical search program created for this capstone project including how it was created, the different steps a users must take in order to run the program, and possible error message that a user may encounter.

Backend Development

When writing the program, two different patterns were followed: observer pattern and MVC pattern. The revision control software used was subversion (SVN), and the tool used to perform the searches was the search engine tool developed by semantic designs (DSM).

Repository Prompt Window

Upon opening the program, the user will be shown a repository prompt window. Within this window is a text field where the user must write the repository URL which they are planning to search. After doing so, the user must click on the "validate" button. This will retrieve the head revision number. Next, the user can use this information in order to decide the range of revisions they would like to search. For instance, if a user would like to access the first twenty revisions of the program, they would write the number "1" in the first revision text box, and the number "20" in the last revision text box, and then click "OK". If the user decides to cancel the search, the default URL will be "http://". Figure 6 displays the repository prompt window presented to the users.

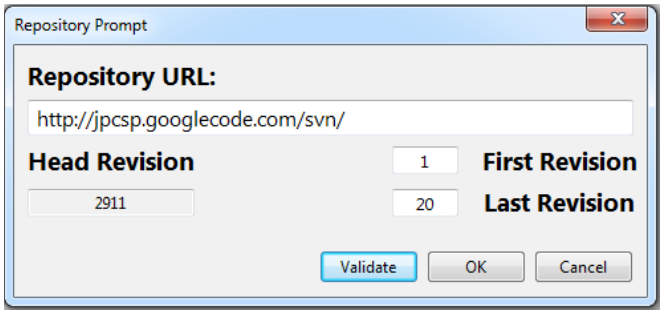

Figure 6 Repository Prompt Window

Application's Main Window

After entering the revisions to be search in the repository prompt window, the user will be shown the application's main window. There are three pertinent aspects to the main window: 1) the query search composite, 2) the logging output tab window, and 3) the revision graph. The main screen is shown in figure 7.

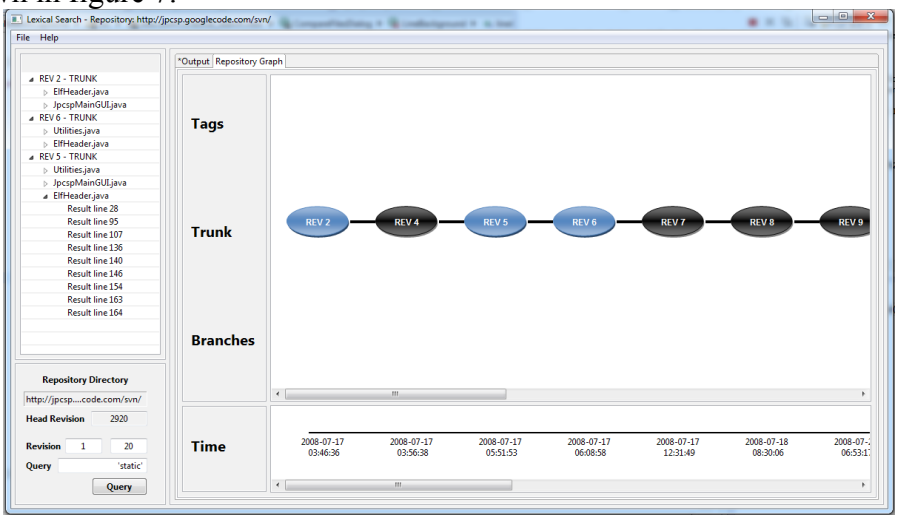

Figure 7 Application's Main Screen

The Query Composite

The query composite is displayed in the lower left hand corner of the main application screen. Here, the user will be able to see the repository URL, the head revision of the project, the revision range being searched, and the query text box.

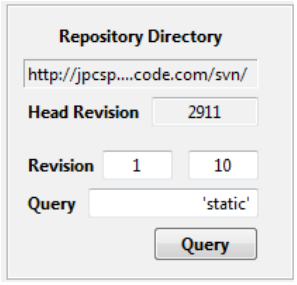

Figure 8 Query Composite

The repository URL, head revision and the range being searched are populated after the repository prompt window is closed. In this section, the user is able to write what they would like to search for in the query text box. After doing so, they will click the "query" button to start the search. If the user wishes to search other revisions, they can do so by changing the revisions range and clicking the "query" button again. However, in order to change the repository URL, the user must select "change repository" from the "file" drop-down menu. See Figure 8.

The Logging Output Tab Window

Upon beginning the search, the logging output tab will begin to populate with the information being returned by the semantic designs tool set. Additionally, any errors or warning encountered during the search will also be displayed.

The information displayed in the logging output tab will be color coded in order to differentiate the system messages, ultimately making it easier for the user to quickly find what they need. Below is a list of what each color represents:

- Beige: Information related to the semantic design tool and the SVN commands.
- Green: Output returned from running the semantic design tool and the SVN commands.

- Grey: information associated with the processes not related to the semantic design tool on the SVN commands
- Red: Error messages
- Yellow: Warning messages
- Blue: "Done" once the search has been completed.

In the upper most part of the output window, the user has several tools available to them such as "copy", "cut", "stop".

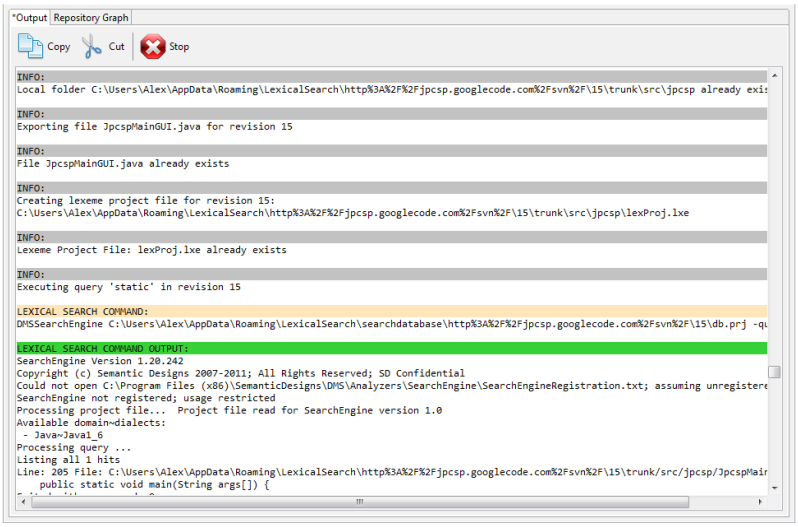

Figure 9 Log Output Tab

The Revision Graph

After the revision search is complete, the revision graph will appear on the users' main screen. Here, the user is presented with a series of nodes which graphically display the search results. Depending on where the revision was made, the node will appear in either the "Tags", "Trunk" or "Branches" line in the graph. Each node represents a different revision in the repository, and the files which are associated with each revision. Furthermore, there is a timeline which matches the date and time in which the revision was created in the repository.

Each node has a tool-tip. Upon hovering over the node, the user will be able to see the files that are associated with that revision node. To open the revision node, the user will double click on the node. This action will be displayed by changing the color of the node from black to blue. The file browser will also be populated. More than on node may be selected at the same time.

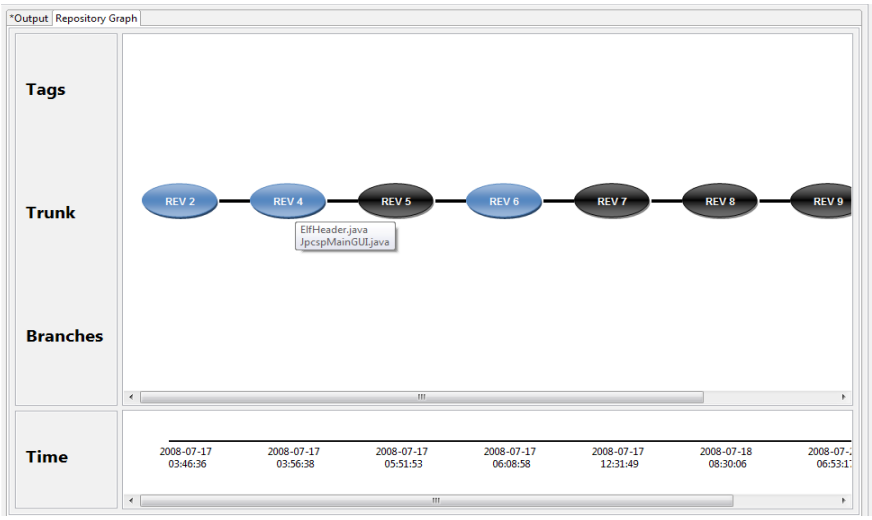

Figure 10 Revision Graph

File Browser

The file browse appears in the upper left hand corner of the application's main window. Here, the user is able to see the information that was extracted from double clicking on the nodes in the revision graph. The file browser will be automatically populated each time the user double clicks on one of the nodes. The data in the file browser is broken into sub-headings. The first level is labeled with the revision number, and the division from which it comes (trunk, tag or branch). The next level lists the file name. The user is able to expand or collapse these files depending on the amount of information they would like to view. Directly under the file names are the lines in which the user is able to see the results from the search.

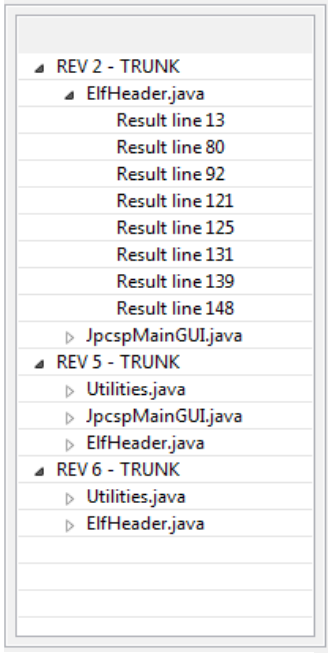

Figure 11 File Browser

To open one of the files, the user must double click on one of the result lines. This will open up the results in a new text editor. Additionally, the user has the option to select two files at the same time in order to compare their contents. To do so, they simply select both of the file names, right click to open the context menu, and select the "compare" option. The user can also remove any of the entries listed, the header file, or the entire revision by right clicking, and then selecting "remove" from the context menu. See figure 11 for a screen shot of the file browser.

Text Editor Utility

The editor utility, seen in figure 12, appears after the user selects the file they wish to chose from the file browser. The editor utility is organized with line numbers. The line number which corresponds to the results are highlighted in yellow for easy access for the user. Also, the text within the editor is formatted according to Java syntax.

If the user selects a result line from the file browser which is under a different file name from the same revision, or the same file name from a different revision, another text editor will appear

with the new results. However, if the user selects a different result within the same file, the program will automatically scroll down to the line in which that result is displayed.

There are a few tools available to the user on the editor utility screen. They can "copy" or "cut" either by clicking on the corresponding button on the top of the screen, or by right-clicking on selected text, and choosing the appropriate function from the context menu. The context menu also provides a "select all" option.

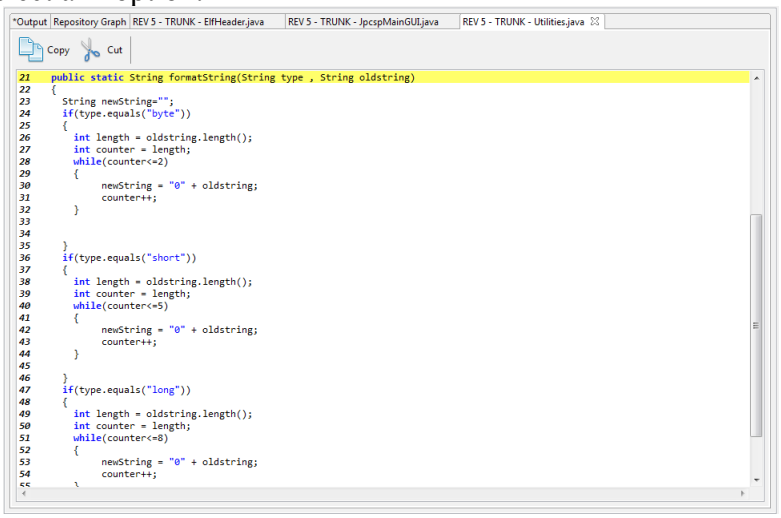

Figure 12 Text Editor

File Comparison Utility

The file comparison utility, seen in figure 13, pops up when the user selects two different file names from the file browser they] would like to compare. The files are shown side by side for easy viewing. The line numbers coordinate with one another, and when the user scrolls through the lines of one file, the other side automatically scrolls to the same point. There are several important features of this comparison utility which aide the user is comparing the two files. These features are as follows:

- If there is a change in the lines between the two files, the line is highlighted in blue, and an exclamation mark (!) is concatenated in the beginning of the line.
- If a new line has been added to the file, that line will be highlighted in grey, and an plus sign (+) is concatenated.
- If a line has been deleted from the program, it is highlighted in yellow and a minus (-) is concatenated.
- If there is no change between the two files, the lines are not highlighted.

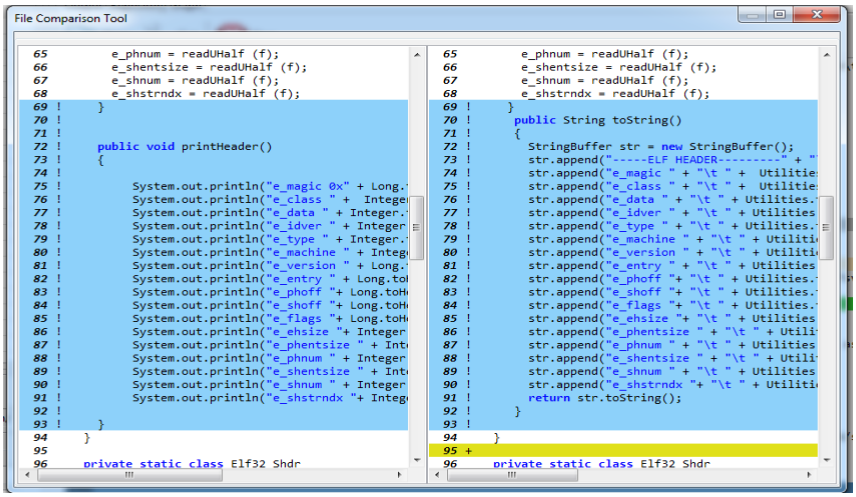

Figure 13 File Comparison Utility

Help File System

The help file system can be accessed by choosing "help" on the drop down menu. Here, the user can receive information on how to use the Semantic Designs search engine tool, as well as view examples of other searchers. A screen shot of the help file system is shown in Figure 14.

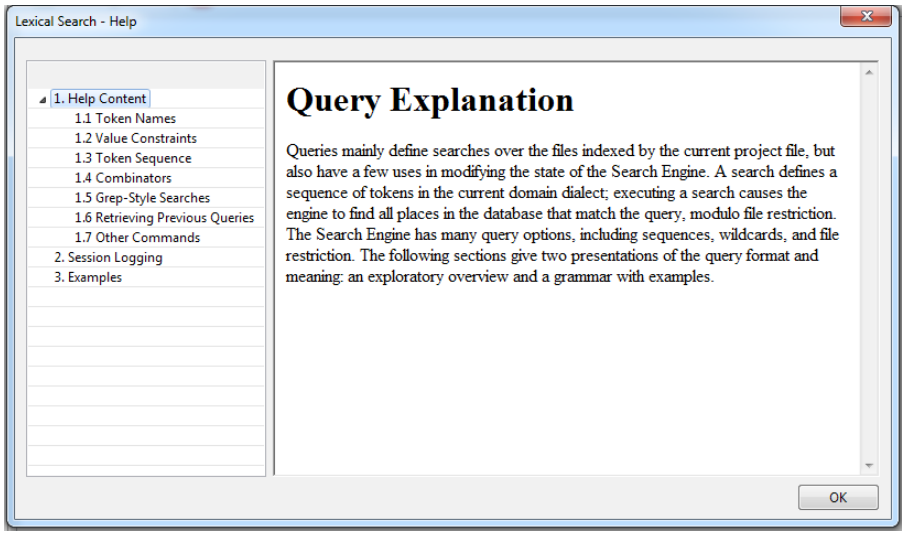

Figure 14 Help System

Error report

An application-modal error window will pop up if any of the following situations were to occur: 1) The user attempts to search for an invalid revision range. This could occur if the starting revision is greater than the ending revision, or if the revision number surpasses that of the head revision. 2) The user enters a non-numerical text into one of the revision ranges. 3) The command line returns with and exit error after runnnig. The message which appears in the error window will change depending on the type of error committed. This error will also be reported in the log output window. See Figure 15.

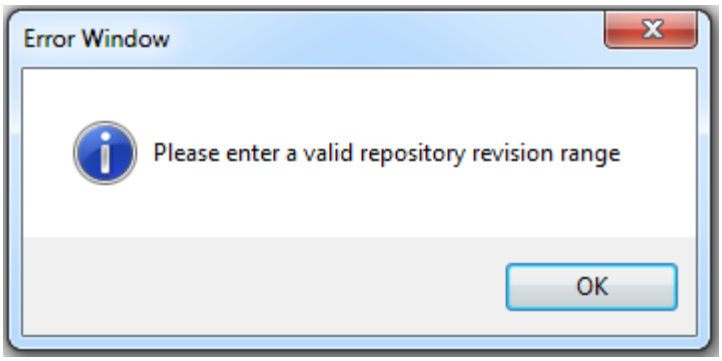

Figure 15 Error Message Window

FUTURE WORK

The Versioned Lexical Search application, in its current state, is able to perform many commands that a user may need when doing repository searches. However, there is still room for improvement. The issues that will be addressed in future releases are listed below.

Help System

The help system in general needs to be expanded. As of now, it provides the user with brief descriptions on how to run the program, as well as a few examples which they can follow. However, the detail in these descriptions is insufficient. First, more detailed accounts of the functionality will be added. Furthermore, a step by step tutorial can be implemented to walk the users through the example programs, further eliminating any confusion or questions that the user may have.

Search Engine Performance

If the user wishes to begin a new search in the repository, and the required files by the search engine have not yet been created, the program does not run as efficiently as expected. For instance, a search of 20 revisions may take up to 120 seconds to complete. The issue will be addressed in future releases, hopefully reducing the search time by at least half.

Support to More Revision Control Software

As previously mentioned, the current revision control software used to implement this application was Subversion (SVN). Future implementations should be able to support other revision control software such as GIT, Mercurial, and Veracity among others.

Develop the Plugging Version

The current stage of the application is a standalone version. However, future releases will migrate to the plugging platform. This will allow for added flexibility. Migrating to a plugging platform may also allow the development of additional features that can be acquired by integrating the tool into the Eclipse IDE.

CONCLUSIONS

Over the course of this semester, I was able to gain a detailed insight on how to develop graphical user interfaces. This experience gained from this project is invaluable not only since it allowed me to develop a unique functional program that could be useful across many different fields, but also because it gave me the knowledge I may need for my future work projects. Some of the most valuable aspects that I will take from this project are the use of the SWT libraries for the user interface development, the concept of multithreading used in the command prompt execution, and the design patterns used for the backend.