Developing a Graphical User Interface and Java Server Using Google Web Toolkit

Charlotte Smail, Joel Detweiler
Computer Science Department
Pomona College
185 E. Sixth Street
Claremont, California 91711
smail.charlotte@gmail.com, joel.detweiler@gmail.com

ABSTRACT

The leading search engines in the field of information retrieval, namely Google and Bing, have simple interfaces. A common thread between these is the way in which they display the results of a query. The results are displayed as a ranked, descending list of documents each with a corresponding snippet summary specific to that query. This paper provides an explanation of a simple online user interface and Java server built for the search engine *Bursti*. The user interface is modeled after the interfaces of the aforementioned search engines and is built to maximize ease of use while maintaining full functionality. In addition, this paper describes the process of creating such an interface with Google Web Toolkit (GWT), a software development framework useful for writing AJAX online applications with the Java programming language.

1. INTRODUCTION

The design and functionality of a user interface is a crucial part of a piece of software. A well designed graphical user interface (GUI) reduces the learning curve of software and frees the user from learning complicated commands. With the addition of an interface, a user searching with *Bursti* no longer has to enter queries into the console. Before the addition of the GUI, the system displayed the retrieved results in an unaccessible format. It is easy for computers to represent a series of results as a list of document IDs with their corresponding blocks of text. This representation is not as easily understood by humans. The interface of *Bursti* acts as a buffer for users that do not have a deep enough understanding of the underlying system to use it. Rather, it displays results in a readable and accessible format. The goal when building an interface is to maximize usability for the end-user, regardless of their amount of technical knowledge. Thus, the interface of a search engine can be evaluated by its ability to present relevant results in a way such that the user can find them faster.

2. BACKGROUND

As of late, the Web has been on a trajectory towards highly interactive online applications. GWT is a tool that is capable of creating such applications. Many new web applications, like Google Docs for example, act more like desktop applications than they have in the past. They are comprised of "widgets" that are easily manipulated by the user. A widget can be defined as a portable chunk of code that can stand alone. Widgets can be embedded into a larger chunk of code to form an entire program. Within our GUI, an example of a widget is the search box along with the search button. GWT enables the programmer to use widgets to create a client/server application in which the browser interacts with a javascript application. The server acts as the "the cloud" that stores necessary information for the system.

Incidentally, the technologies used to create a webpage with GWT have been around for many years. These include cascading style sheets (CSS), Javascript, HTML, JSON, etc. GWT utilizes browser technologies that are commonplace in the world of web development. Furthermore, GWT is unique in the sense that it combines all these commonly used technologies into one tool. It allows the programmer to build applications solely with the Java programming language. Thus, many programmers could use it without learning browser technologies. When using this tool, it is usually necessary to edit the CSS to achieve a particular aesthetic quality and do detailed formatting within the pages. The learning curve for such tools, like CSS and HTML for example, is not particularly steep. In this way, GWT is able to combine the necessary browser technologies and utilizes the power of javascript to create web applications.

3. GUI AND SERVER DESIGN

3.1 An Ideal Interface Design For Bursti

There are many different methodologies for creating effective graphical user interfaces. On one end of the spectrum lies maximizing user control. This is achieved by providing many options for the user. An example would be a search engine that has the capability to search by various categories, in a variety of languages, or with multiple types of media. While this approach benefits the kind of user with search engine skills, it also runs the risk of overloading the

user with too many options. This can decrease efficiency. In some cases, it is better to create a sleek, simple interface. The opposite end of the spectrum of GUI design is to minimize user control. The interface of *Bursti* is simple in its design but mains the functionality of the underlying system.

3.2 Using GWT to Create the Interface

GWT allowed us to easily complete our design goals using the object-oriented programming language Java. Google's widgets, like search boxes and search buttons, were used to design our ideal search engine interface. Each "panel", which can be thought of as a section of one page, encapsulates particular functions of the interface. This encapsulation allows us to move panels around according to how the user interacts with the system. For example, when a query is entered, the logo and search box portion of the webpage is moved up to the top of the page and the results are displayed in their place.

Once a search is performed, a request is sent to a PHP script that accepts a query, a start index, and an end index. The start and end indices indicate how many results the user wants on a single page. If the query is the original search, a desired number of results from the beginning of the search results is requested. If the user has clicked a next or previous results page, however, the GUI only requests documents from the current subset of total results. The PHP script interacts with the Java server, which processes the query, performing a complete search while only returning results between the given indices. Results are then displayed in a list format. The GUI receives plain text containing document titles, document URLs, and corresponding snippets. This text is capable of then being parsed, in order to present the document titles, URLS, and snippets in their ranked order.

Stored search results can be displayed within a series of panels. Titles are created as html links to the corresponding URL, while the snippets are presented as plain text to describe each document. Panels containing each of these objects are then added to a results panel in ranked order, set to be visible once a search is performed. The results panel displays documents in a vertical format, making it easy for the user to visualize the ranked results of their search.

The presentation of results is designed to be aesthetically pleasing for the user using CSS. We opted for documents to have bold, black titles that could stand out clearly next to the snippets. They also highlight in yellow when scrolled over so that the user can easily understand that the title is a link to the document's URL. Each snippet is presented in smaller, normal text directly below its document's title. In doing so, it should be clear that the snippet is there to describe the document and how it relates to their query.

: Figure 1

Our search page was also made to handle the user's requests for different information. The search, next, and previous buttons were created to react to the setting at the time of the click. The search button will always perform a new search, clearing the older results if the search box is currently empty. Additionally, the next button is set to always give the next set of results for the most recent query. If there are no more results after the current set for a query, the GUI will display a message stating so, remove the next button, and keep the previous button. Similarly, the previous button is set to not appear unless there is a previous page of results. The previous button, as expected, always returns the user to the most recent page of search results for the current query.

4. RESULTS

A GUI is difficult to evaluate in the traditional sense, as the GUI is largely tailored to appeal to individual users. The quality of an interface is difficult to measure with standard metrics because of its subjective nature.

Our first goal for the search engine was to be able to process queries and display results in a short amount of time. Thus, we evaluated the time it took Bursti from the moment a query was submitted until results were displayed. Over a significant number of tests, we found that a single search took an average of 3.29 milliseconds to complete. This is a much smaller time than a human can recognize which indicates that our system returns results almost instantaneously. We conclude that we completed our goal of creating a GUI that processes a query and displays results for the user in a trivial amount of time.

The appeal of the GUI itself, rather than the entire system, however is more difficult to analyze using concrete metrics. Our goals for the GUI were to have a sleek display that limited the users options while maintaining functionality.

We successfully completed an interface with simple functions such as fetching the next set of results and going back to previous ones. We also desired to maintain functionality, with the ability to perform a new search at any time and to traverse the extent of the results. We were largely successful in this regard, as the user can always perform a new search and all results are accessible at any given time. The one fault we found was that the back button within the browser does not function within the context of our GUI. We do not provide support for saving results after a user has visited a page and wants to revisit the results they just recieved. Because it dynamically changes the page using javascript, their is no unique url to a set of search results or a certain page. This lacks some of the functionality of an ideal GUI, which was our goal. In order to fix this issue, a different page would have to be generated each time the user changes something, rather than changing the current page. This is a complicated process that would drastically change our current system, and thus is very difficult to implement within our existing code.

We were able to evaluate the aesthetics and functionality of *Bursti* by informally surveying some users. They all commented that they wished they could use the back button within their browser to return to results after clicking on a result. Additionally, some users stated that it was unclear that document titles were also links to their associated text. However, the majority of user comments showed approval for the aesthetics of the Bursti interface. Users appreciated the simple layout and presentation of results, describing it as easy to use and straightforward. On the whole, the informal survey showed that we have met our goal. Bursti is a simple search engine that is straightforward and easy to use, making it appealing to the user that are looking for a simple and effective tool for searching.

5. CONCLUSION

In recent times, the demand and popularity for customizable yet simple web applications has risen. In response, Google has created a tool that harnesses simple yet powerful browser technologies such as HTML, CSS, and Javascript. In the creation of a web application with GWT, all of these tools can be used indirectly with the programming language Java. This is a big step for web application design. All of the browser technologies, like the ones mentioned above and countless others, are necessary to make web applications, but there is no simple way to interface them together. GWT is not a perfect way of doing so, but Google has at least cobbled together smaller tools to create a larger more powerful one that is actually quite capable of creating good web applications. Once we conquered the steep learning curve that is GWT, we were able to successfully create a simple search engine interface that met our goals with our existing knowledge of Java. Our GUI for Bursti met our design goals, as it was shown to be quick and appealing to the user.