Evolving the General Text Parser (GTP) Utility into a Usable Application Via Interface Design

A Thesis Presented for the
Master of Science Degree
The University of Tennessee, Knoxville

Patrick A. Lynn
December 2002

Acknowledgement

I would like to thank Dr. Michael W. Berry for all the direction he has given me throughout my thesis project as well as his take on the computer science profession and life in general. I would like to thank Dr. Bradley Vander Zanden and Dr. Tom Dunigan for agreeing to be apart of my thesis committee. I also wish to thank Prof. Richard Martin who taught me that there is always a creative solution to the problems life throws at you. Having taken courses with each of these professors, I can honestly say they are some of the best teachers I have had in my academic career. There are plenty of professors in the world but not all are teachers.

I also want to say thank you to my family for their support during my pursuit of higher education. My parents never pressured me about school and yet they managed to instill in me the need to become a well-educated person. Without them none of this would be possible. Thanks to my in-laws for their understanding when I had to attend to my studies and could not always visit on the weekends. Also, I am grateful to my wife who put her own graduate plans on hold to allow me the opportunity to work towards my degree. Lastly, to my daughter Kaili, you don't need to ask when will I be done with school anymore.

Abstract

The GTP (General Text Parser) software started out as a single C++ command line utility for the Solaris computing platform and has grown into a multiple language, multiple platform program supporting twenty-nine different options. The GTP software "provides general purpose parsing of document sets and matrix decomposition for information retrieval applications". Current releases of GTP include a second utility, GTPQUERY , which has its own set of options and is used in combination with GTP to allow query processing across the parsed document sets. The combination of multiple command line programs, various platform versions, and a complex array of options began to place a huge burden on GTP's users including the GTP software development group. A method was needed to remove the burden from the user and allow the multiple pieces of the GTP package to function as a single entity. This thesis will describe how the addition of a graphical user interface transformed GTP into a full featured application and allowed users to use the core GTP utilities in a more effective way than could be accomplished through the command line. Also, a visualization concept for viewing query results is introduced. The concept, through the use of graphics, attempts to provide the user with a better understanding of the returned documents while reducing the user's need to physically read each document in order to assess relevance.

Table of Contents

List of Figures

1. Introduction

The GTP (General Text Parser) software was originally developed as a single C++ command line utility for the Solaris computing platform. Today, the GTP package includes a second utility, GTPQUERY, and is available for many different platforms in multiple programming languages. GTP has functioned for years as a command line utility that "provides general purpose parsing of document sets and matrix decomposition for information retrieval applications". Through the use of mathematical modeling, GTP generates vector space models [1, 4] of text documents and terms contained within said documents that can then be used for query matching. The multitude of options supported by the GTP parsing utility greatly affect how the vector space model is generated as well as the model's efficiency. However, the quantity of options and having to specify them through the command line places a large burden on the program's user and effectively hides much of the power the utility is capable of. In addition, GTPQUERY [1], which added query-processing functionality to GTP, has its own set of command line options for the user to remember. Given the multiple elements that now make up the GTP package, a method was needed that would coalesce the applications, provide the user with a way to consistently execute GTP across versions and platforms, and make GTP's options more accessible to the user. A way to accomplish the task was to develop a graphical user interface that would represent the multiple utilities as a single application. This would also allow for future additions to be included as an additional selection on the main part of the interface.

2. Design Criteria

As a part of the initial system design phase, design objectives and performance requirements were established to ensure that by adding an interface to GTP all command line short comings and usability issues would be addressed. The following objectives were identified.

Make the software's options more accessible to the user.

Due to the command line nature of GTP each option must be specified at execution time and is in effect hidden from the user.

Reduce the user's memory workload.

With twenty-nine separate options, a user is not likely to remember all available options or the proper syntax for using those options. Ironically, a software utility that aids in indexing information to make finding information easier requires manually searching through a help file to make use of the program.

Automate option selection where applicable.

Some of the GTP options require the use of other options or, alternatively, restrict other options from use. When GTP is used through the command line a user has the ability to specify incomplete or invalid option choices.

Provide consistency across platforms and languages.

Multiple versions of GTP can be used on either the same machine or separate machines possibly with different computing platforms. The user experience should be similar for any GTP interaction.

Give GTP more of a commercial appearance.

In addition to adding to the user experience through aesthetics, a professional appearance can aid the user's confidence in both the capabilities of the program and user's ability to make use of the software.

These criteria were initially developed for the GTP parser only as GTPQUERY and other functionalities were yet to be developed. Though the scope of the design changed as the concept design phase proceeded, on a whole the main objectives described remained consistent throughout any added functional requirements.

3. Concept Design Phase

Having established design objectives the project moved into the concept design phase. The initial focus for this phase was to organize the multitude of options into a logical format. Options were categorized in a progressive hierarchy beginning with similarity of functions. For example, options used to change default program values were grouped into a "default settings" category and options necessary for program execution were grouped into a "required" category. Next, items were further grouped by whether or not they required parameters (i.e. –m number requires a number to be given with the –m option). Finally, options and categories were given an importance ranking based on how a user might interact with the groupings.

With the interface's content organized, hand sketches of possible layouts were drawn and the most promising elements were combined in a graphical representation of how the interface might look (see Figure 1). One principle of interface design is providing users with different *locuses of control* [3] based on their experience or needs of a system. This was an important consideration at this stage of the design and the initial representation of the interface was constructed to depict a simple display showing only minimal information needed to use the GTP parsing software. However, one of the features of the simple display allowed the interface to be expanded giving access to all of the more advanced features GTP is capable of.

At the same time, thought was given to creating a professional or commercial atmosphere for the interface. A graphic that would act as a header for the interface window seemed the best option. The header would contain a logo treatment that would provide the GTP package with an identity. It would also provide a suitable place to display copy write or version information. As with the interface layout, sketches were drawn and the more promising concept was chosen for a base of development.

After the GTP software group reviewed the concept layout and deemed it a good starting point, a prototype interface was ready to be constructed in software. The Java programming language was chosen for the interface construction in order to meet the requirement of consistency across platforms and versions. The Java language is

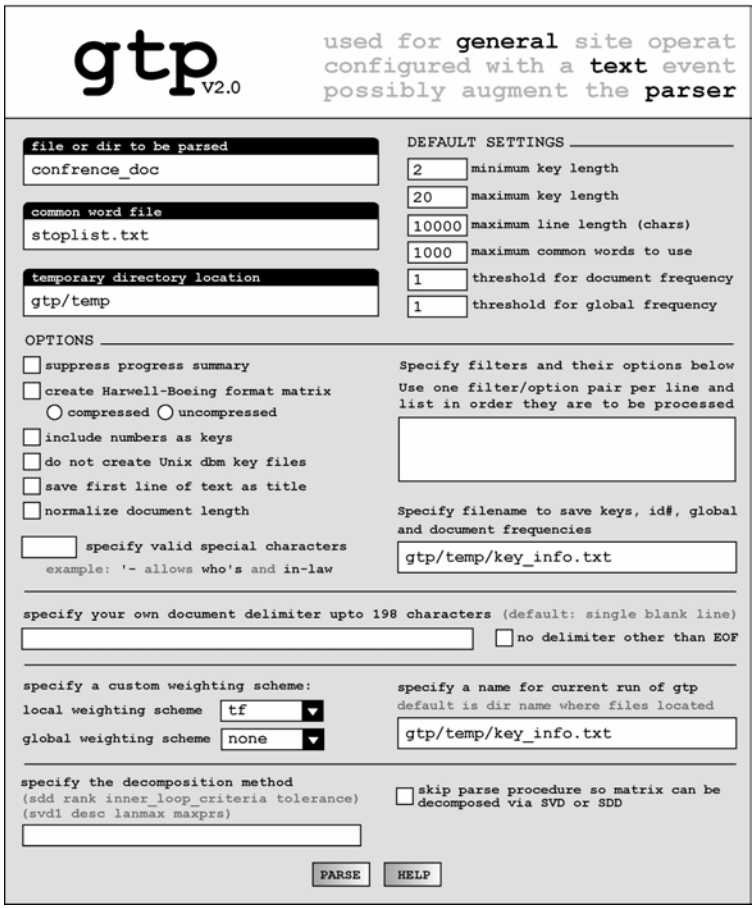

Figure 1. Original interface concept

supported on all the computing platforms that GTP is currently released for as well as many that GTP has yet to be tested on. Programming in Java would allow the interface to be created once and eliminate the need to rewrite for multiple platforms. This would also allow the interface to hide the underpinnings of GTP and allow new utilities to be rolled into the interface while maintaining the appearance of a single application. In order to accommodate the multiple versions of GTP, the interface would need to include the functionality of selecting the version of GTP that will be used.

Once the prototype for the single expanding window interface design was completed, it was presented to the GTP software group for evaluation (see Figure 2). One of the primary concerns posed during the evaluation was that, when expanded, the interface window filled most of the screen height on a monitor displaying with a high resolution. If used at lower resolution setting the interface would not fit within the monitors display area. It also became apparent that, when expanded, the interface was attempting to present too much information and making it possible to overwhelm the user with the layout. A design decision was made to use a stationary window that would utilize tabbed panes to further segment the options as well as reduce the amount of screen real estate the interface would take up.

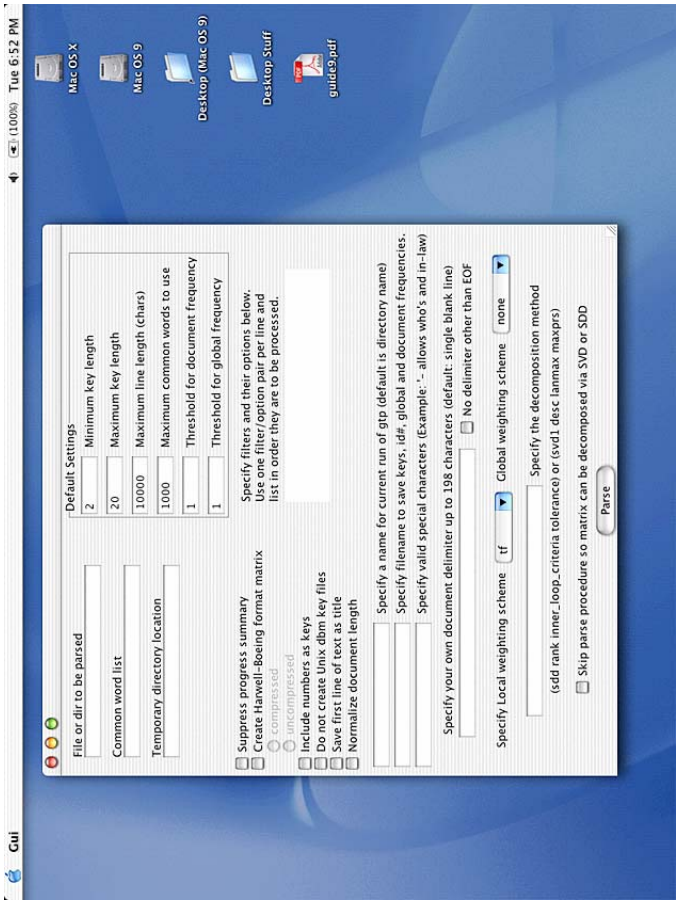

Figure 2. Single expanded window interface

4. Final Design Phase

With the decision made to use a tabbed pane layout, the options' organizational criteria were revisited. The groupings were reviewed to re-determine placement of options according to tab selection and the ordering of the tabs themselves. Once reorganized, software construction of the new layout began. Features were added to the software to accomplish some of the established design objectives. To allow execution of different versions of GTP, a selection box was added to the main tab of the interface and a model for invoking GTP was developed. The interface would act a "driver" for the GTP programs and execute them through an external system call. To accomplish this the input, output, and error I/O of the invoked process had to be trapped and dealt with through the interface. Since GTP would be executed as a separate process with its own error checking routines, the same error checking functionality had to be built into the interface. Any requests for user input that GTP might request was determined prior to the GTP process execution so that the interface could receive the user input and pass it on to GTP. To reduce some of this cross process overhead as well as reduce the user's memory load option compatibility checking was added to automatically prevent syntax issues based on the selected options. If the user selects an option that cannot be used with another option, the second option will automatically be disabled. If the user selects an option requiring another option, the second option will automatically be selected at the same time. For similar reasons general error checking of interface input and user feedback about possible errors were added as well.

With the interface revamped and most of the functional code complete, initial testing began to determine the usability of the interface. Functionally the interface performed well. A text string consisting of the selected options could be generated and used to invoke the external GTP parsing program. The interface was also receiving the process's output and able to display it back to the user. Visually, the testing revealed that the interface was fluctuating in size depending on which interface tab was selected. The sizes of the panes for each tab were different due to the varying degrees of content assigned to each tab. An adjustment to the interface was made such that the interface would determine the largest tab component and force the other components to use the same amount of space. This stabilized the size of the interface window and allowed it to remain stationary throughout the user session.

About this time in the design process a second GTP utility, GTPQUERY, was being completed. GTPQUERY would allow searching of the vector model space created by the GTP parser. The new query utility had ten options of its own and would require separating its place in the interface from that of the parser. The query options were organized using the same criteria established for the parser (similarity of function, options requiring parameters, importance of use) and built into a separate tabbed interface to maintain a consistent feel between the two sections. A menu structure was developed that would appear upon interface execution allowing the user to select between the two functions. The menu would also reappear after the interface had successfully run the selected utility (see Figure 3).

Having rolled the GTPQUERY interface into the overall interface, a user now had the to ability to parse a document collection and execute a query against the parsed information. There was, however, no method for the user to view the results of a query. This was an important piece of functionality not present and one that would be needed in order to provide the user with a complete experience. A decision was made to add the "view results" functionality into the interface in a manner that would show the results of the query and allow the original documents listed in the results to be displayed. Up to this point the interface had functioned as a "driver" for executing external programs. The interface was just a method to gather information and pass it back and forth between the user and the software. Now the interface would need to be its own utility and this presented a few problems. The first issue was that the interface had no knowledge of the parsed documents and depending on the options used with the GTP parser, a single file may contain several documents. This excluded any assumptions about treating individual files as documents and using a directory listing as a document index. Furthermore, the only files readily available for interaction with the interface were the file used to specify the query to GTPQUERY and the result files GTPQUERY generated. The result files returned by GTPQUERY contain the document id numbers of the relevant documents along with ranking numbers for each returned document. This file could easily be read and displayed by the interface but without knowledge of how the documents were parsed an individual document could not be read and displayed. When the problem was posed to the GTP software group it was determined that the GTP parser would be

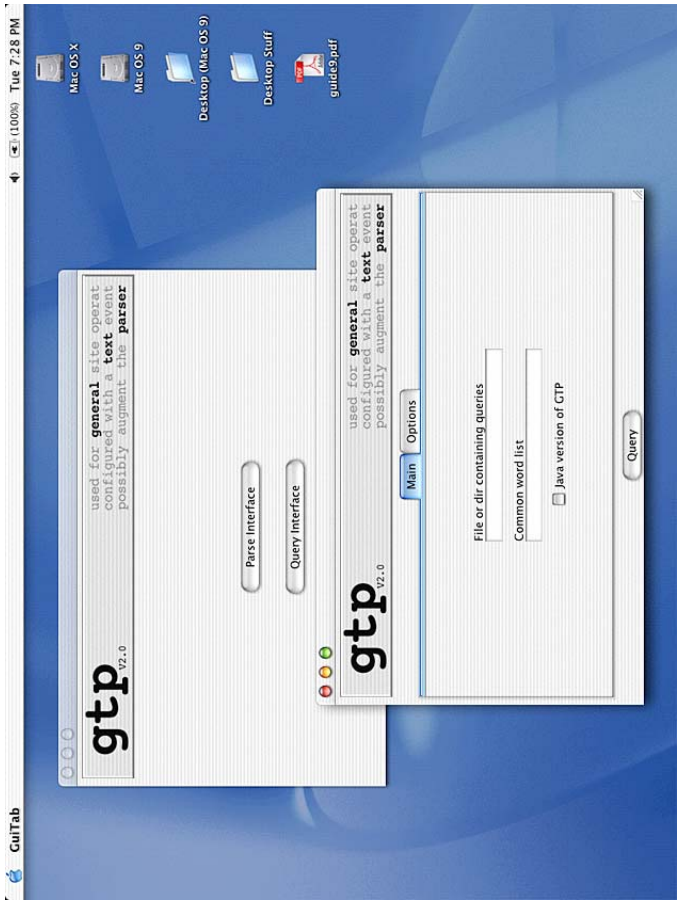

Figure 3. Main menu and query interface

modified to generate an index file that would list an absolute file path and an offset to the starting position of the document within the file. The index list would be ordered such that the line number of the index file related directly to a document's id number. With the addition of the index file, the interface now had a method to resolve the document id numbers from the query result files and display the appropriate document back to the user. An additional element was added to the interface's main menu to allow the selection of viewing the results generated by GTPQUERY. When this choice is selected, the interface displays a text input box for the name of the file or directory used to specify the queries to GTPQUERY. The interface uses the input from the user to read the number of queries and dynamically creates a button associated with each individual query. The buttons are then displayed in a column next to a text window and when one of the buttons is selected the results associated with the particular query are displayed in the window. If the user wishes to view a document listed in the result list, the document id or score can be selected with a mouse click and a popup window will display the document text (see Figure 4). If the user selects a query button and no results exist, a popup dialog window is displayed to alert the user to the fact that no matches were found for that query. There is also a button on the query results interface that clears any result list from the text window, removes all the query buttons, and returns the user to the main interface menu. With this addition to the interface, a consistent path of use was established. A user would now be able to use the GTP package to perform the main actions associated with information retrieval, parsing, querying, and viewing results.

A new feature that is currently under development will bring remote storage capabilities [5] to the GTP package. This feature benefits both the GTP parser and GTPQUERY and was far enough along in development to set parameters on interface elements required by the remote storage feature. Since the feature is functionally the same for both parsing and querying, the same interface tab could be used on both the respective interface sections (see Figure 5). The remote storage addition was made to the appropriate sections but until the feature is ready for release this portion of the interface will be non-functional.

With all of these additions to the interface design, the interface had become a full featured application allowing a user to use the core GTP utilities in a more effective way than could be accomplished through

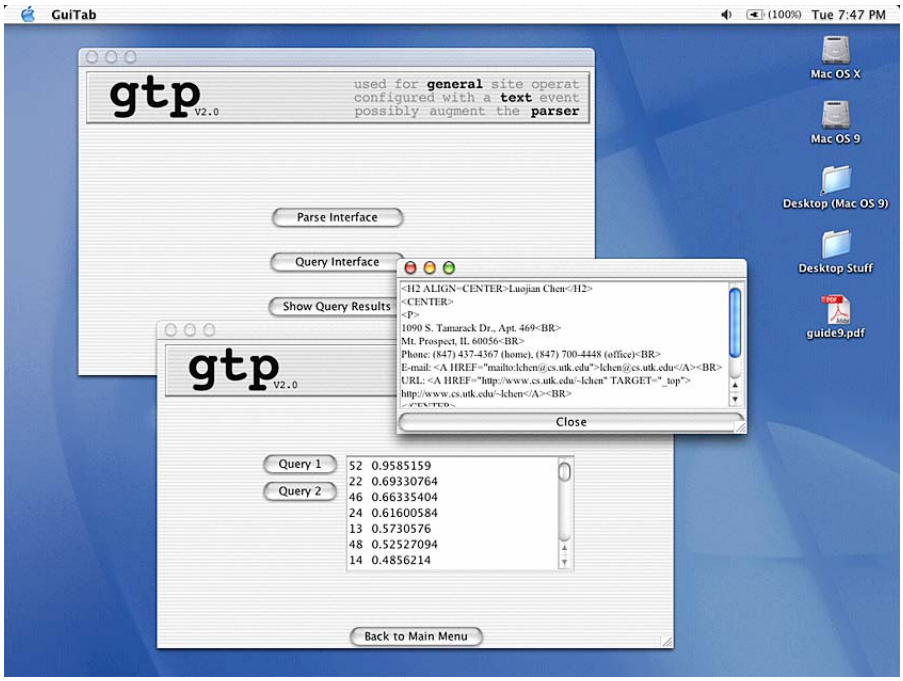

Figure 4. Main menu, query button list and document window

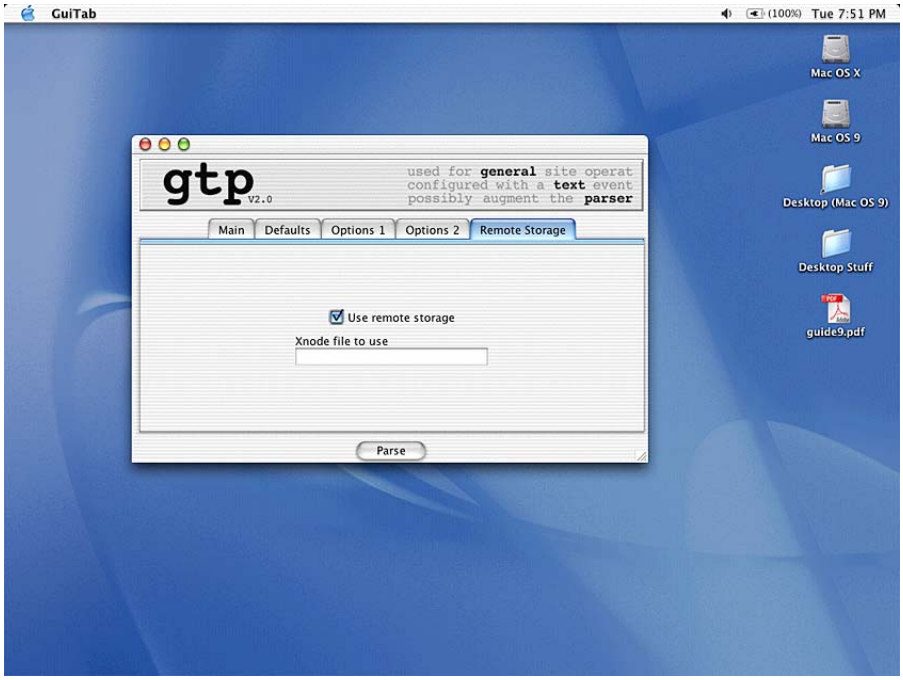

Figure 5. Remote storage tab of parsing interface

the command line. The interface was at this point ready to be tested in real use situations.

5. Testing and Evaluation

Hoping to gain feedback on the effectiveness of the design decisions as well as reveal additional features that might be required to improve ease of use for the user, the interface was released to members of the GTP software group for testing. After only a few days of use both software bugs and new feature needs were identified. Of the software bugs that were revealed, only the ones considered non-trivial and having design implications are discussed here.

The first bug reported was an ArrayIndexOutOfBounds exception that was thrown when a user attempted to view the results of a query. This error proved to be fatal as the interface would become frozen and need to be restarted. After replicating the circumstances to generate the error, the cause of the error was discovered. The "view results" section of the interface reads the file or directory used to specify queries for GTPQUERY and upon counting the individual queries attempts to build buttons for each one. The particular situation generating the error showed a count of 4736 different queries. The exception was actually being thrown by the underlying Java Swing methods rather than a specific part of the interface code. It seems that Java either imposes a limit to the number components that can be contained within a display or cannot resolve the problem of having more display components than screen real estate to place them in. In either case, the solution was to limit the number of query buttons to eight. The eight-button limit was chosen because that was the maximum number of buttons that would fit within the interface boarders and not cause the interface to need resizing. The problem with this limit is that now query results can exist but cannot be accessed through the interface. Currently this limit is considered to be a design limitation that will just be imposed by the interface. However, in the future a more elegant solution will be employed and is likely to function in a manner similar to the "next ten" buttons frequently used by Internet search engines.

The next bug involved a "processing display" that is supposed to become visible after the user selects a button to start the execution of one of the GTP utilities. The problem here was that the display was not being shown, in most cases, until the particular utility had completed execution and a status window of the utilities run displayed. The function of the "processing display" is to signal to the user that something is happening in response to their button click and without it

displaying properly, the interface has the appearance of freezing especially if the utility takes a while to complete execution. Several methods were attempted in order to fix the problem but none of them were successful. It was concluded that the problem likely stems from the *thread safe* nature of the Java Swing components and might be solved by creating a separate thread for the GTP utilities to execute in. Swing components are accessible by only one thread at a time and in this case the "processing display" is set to visible in the event handling thread but the interface repainting does not occur until the main interface thread regains control. A separate thread was not apart of the original interface design because the utilities could be executed with lower overhead and it provided a simple method for the interface to wait on a utility to finish execution. The addition of separate threads will require a good deal of reprogramming and is something that needs to be resolved before the interface is posted for public release.

The remaining bugs were not as severe as the previous two and involved issues with window resizing and text formatting. The testing identified that certain displays used in the interface would disappear if the user manually resized a window to a smaller size. Each display component has a minimum, preferred, and maximum size as well as a method for determining what a component's size is at a given moment. If the size parameters are not initially set, Java will set them at run time and ensure that a display is large enough to show all the components contained within. The solution was to get the initial size of the display, determined by Java, and reset the minimum and preferred sizes to this value. This allowed a display to be manually resized larger but would not allow the display to be resized smaller than its original display area.

The last bug issued involved text formatting of the interface displays. Displays that were used to show text read from files appeared to have different formatting for different users. The formatting appeared to have discrepancies between separate computing platforms as well as between differing versions of Java on the same platform. It was determined that this would have to be accepted as a side effect of using the Java language for the interface's development. Much in the same way that web pages are designed to accommodate different HTML (HyperText Markup Language) browsers, steps were taken to minimize the formatting differences but in general the problem could pnot be completely eliminated.

In addition, the GTP software group also identified features that could be added to the interface. Primary among these was the addition of a demo feature to provide the user with an example of how each of the GTP utilities could be run. To accomplish this demo buttons were added to the interface's main menu that corresponded with the parsing, querying, and viewing results sections. When selected, a copy of the interface is created with only the section for the appropriate utility displayed (see Figure 6). All components of the demo interface are already set to allow the particular utility to be executed in a predetermined manner and cannot be changed by the user. The user must still select the button that runs the utility in the same manner as the regular interface but this way the user is able to see how the options were set in order to produce the resulting output.

Another important need identified was a method to retain interface settings throughout a session. The interface was designed so that once a utility finished execution the interface section of that utility was reset to the default values. It was pointed out that users might parse or query over the same set of documents multiple times in a session and require only minor changes to a previous set of settings. At the same time it was felt that a user should not have to undo every individual setting from a previous run if a completely different set of options were desired. To accommodate this the interface was changed to retain the settings from the last run of a utility and a "reset" button was added to the parsing and querying sections of the interface. No "reset" button was required for the view results section as that portion of the interface is dynamically constructed each time it is run and there are no settings to retain. In order to give the user a finer control over the resetting of interface options, the "reset" button clears only the tab that is currently displayed. This way if, for example, a user wants to reparse a set of documents using the original settings of the defaults tab but retain all other interface settings it can be done with minimal effort on the user's part.

The remaining additions to the interface were mainly minor design changes and did not require a large amount of effort to complete. The changes included a "help" button being added to the main menu of the interface to provide the user with access to the GTP man page information. A "main menu" button was added to each of the separate utility interfaces to allow access back to the main display no matter what part of the interface a user was currently in (see Figure 7).

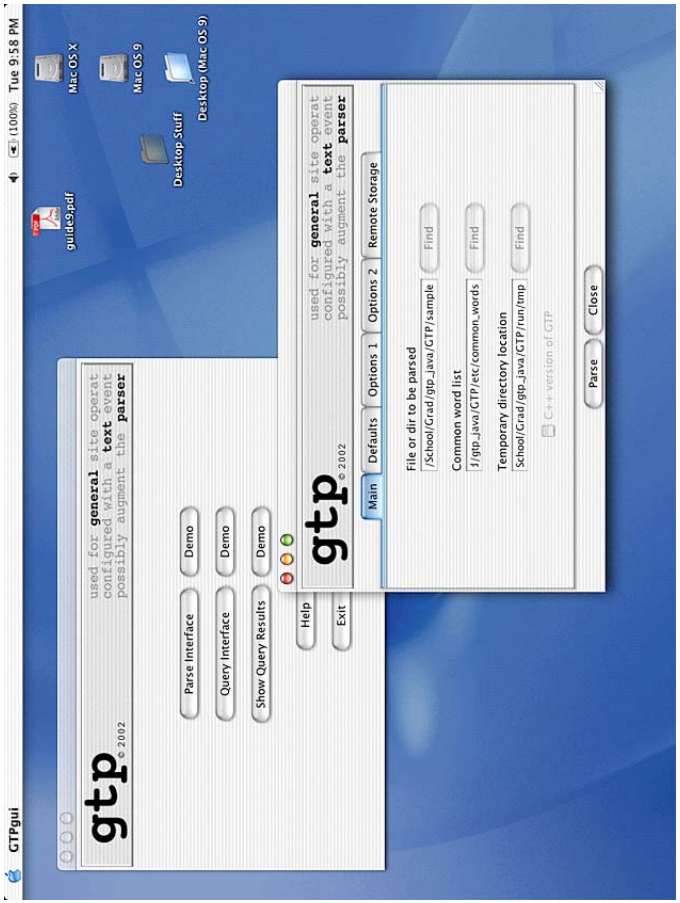

Figure 6. Main menu and parsing demo interface

Figure 7. Parsing interface with reset and main menu buttons

Finally, names were added to the interface window title bars. Even though the interface uses a graphic header for identification, it was pointed out that when minimized there was nothing to inform a user about what the minimized object was.

6. Future work

One area that was researched for the "view results" section of the interface but not implemented was that of information visualization. Information visualization tries to provide the user with a better understanding of information through the use of images as opposed to straight text. Currently this is a major research focus in the information retrieval discipline and many different concepts are being pursued. The concepts range from visually simple displays such as Venn diagrams that aid Boolean query specification [3] to visually complex displays such as three-dimensional landscapes that represent document clustering [3]. One of the better concepts researched was the movieDNA [2] interface which attempts to provide a user with contextual information of video. The concept uses categories and a DNA-like display to give the user an idea of the content or characteristics of a video segment contained within a recording. All of the researched concepts had both good and bad points but they all shared the characteristic of having the potential to confuse the user either through a complex display or an abstract representation of the information. It also appeared that the concepts worked best for the given situation they were designed for and were not necessarily transportable across information applications.

With observations from the visualization research, it was decided that a new concept for use with the GTP "view results" interface should be developed. The problem seen with a traditional ranked order list like a user might receive from an Internet search engine is the user must read the individual documents to really determine if it meets the query criteria. Also, the user has no knowledge of any relationships that may exist between documents in the returned list and therefore has no way to associate a group of documents as potentially relevant based on an individual document's content. The idea of creating document relationships based on word content seemed like a good avenue for providing a user with a more intuitive view of the query results. It should be noted that the intention of this concept is not to completely replace textual information with graphic representations. After all, the information being represented visually consists of text documents and should include text to aid the user in making relevance judgments.

In order to show relationships between the documents a method using a list of the ten most important words for each document was developed. This "word list" would allow both the user and the

interface to compare key words between documents. The user would potentially use the list to gain an idea about the content of a document as well as similarity between documents. The interface would use the comparison of lists as a basis for graphically displaying document relationships through a method such as highlighting or proximity of placement. The algorithm for determining the most important words was not developed as a part of the concept as it was assumed that a suitable algorithm using term frequencies or another weighting scheme could easily be generated.

Having established an idea of how to provide the user with an increased understanding of query results, a graphic medium to execute the idea is needed. The first graphic concept developed was modeled after celestial objects and how gravity ties them together (see Figure 8). The idea would be represented by three-dimensional space with the query positioned in the center. The documents and terms would exist in space around the query and be positioned based on the results ranking and similarity of "word lists". As a user moved through the space titles could be viewed by *mousing over* document objects.

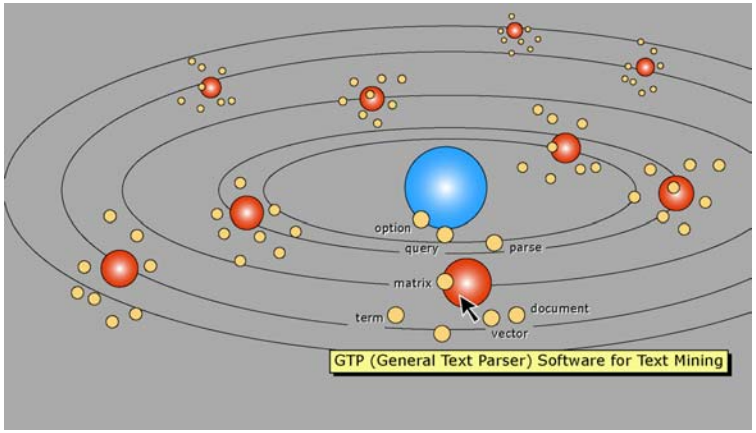

Figure 8. Celestial visualization concept

If a document object were selected then space would be reordered to show objects based on their similarity with the selected document as opposed to similarity with the query. The more the concept evolved, the more it seemed to suffer the same problems seen in the original concept research. It was becoming complex, potentially confusing for a user, and had no real tie to the type of information it was attempting to represent.

In order to create a simpler solution, a decision was made to completely rethink the graphical representation from the standpoint of how text documents are actually used. Before the Internet became commonplace in society people used to use libraries and enlist the aid of a librarian to find information on the subjects they were interested in. The librarian would gather reference materials based on the users requirements, stack them at a table, and have the user sit at the table to review the information. A stack of books in a library setting seemed like a very logical way to graphically represent results of a query. The concept would operate as follows. The user is shown a room with a stack of books sitting on a table and a bookshelf in the background. The ranked scores generated by GTPQUERY would determine the order of book placement. As the user positions the mouse over a book the relevant document information, such as the title, and the document's "word list" are displayed. At the same time highlights appear around other books considered to have a relationship with the book currently being reviewed. If a user selects a book by clicking it then that book along with all that are related to it slide out from the main stack and display their "word lists" (see Figure 9). All "word list" words common across documents are shown in bold typeface to indicate how similarities were arrived at. The bookcase in the display can be used to store documents by simply dragging a book from the stack to the bookcase. By putting a book on the bookcase documents can be stored between queries and GTP sessions. When new books are added to the bookcase they will be ordered by their similarity with the books already on the shelves. This similarity is based on both the comparison of the document "word lists" and comparison of the document vectors generated by GTP's underlying latent semantic indexing (LSI) model [1, 4]. If a close similarity exists, the new book will be placed next to its similar counterpart. If it is not so similar, the book might be spaced away from the other books. If a book is very different from ones already stored, it will be placed on another shelf.

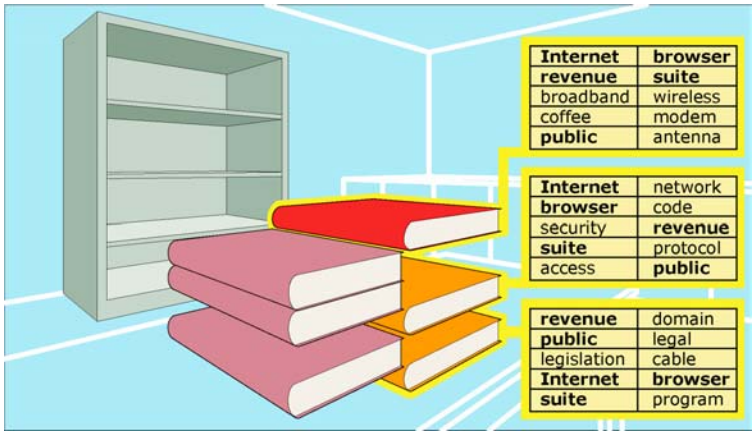

Figure 9. Library visualization concept

The bookcase not only provides a way to store documents deemed relevant by the user but also attempts to maintain the idea of relationships with documents from separate query results. This way if a user executes multiple queries in an attempt to find documents about the same subject the bookcase can relay visually the documents that are closely related and the documents that may not be as relevant to the subject as originally believed.

This "library" concept provides the user with two important functions through visualization. First, it provides the user with a concise view of the query results and enough information to allow the user to make relevance judgments without the need of viewing every single document. Second, it provides the user with a method to store relevant results from a query and retrieve at a later time. This goes a long way to reduce the memory workload on the user as the need to remember where a document was seen from a prior search is virtually eliminated. It is felt that with further development, this concept would not only benefit the GTP interface but also be beneficial to other information retrieval applications as well.

Bibliography

[1] J. Giles, M.W. Berry and L. Wo, "GTP (General Text Parser) Software for Text Mining", Proceedings of the C. Warren Neel Conference on the New Frontiers of Statistical Data Miing and Knowledge Discovery, Knoxville, TN, June 22-25, 2002.

[2] D. Ponceleon and A. Dieberger, "Hierarchical Brushing in a Collection of Video Data", Proceedings of the 34th Hawaii International Conference on System Sciences, 2001.

[3] M. A. Hearst, User Interfaces and Visualization, *Modern Information Retrieval*, pages 257-323, ACM Press, 1999.

[4] Understanding Search Engines: Mathematical Modeling and Text Retrieval, M. W. Berry and M. Browne, SIAM Book Series: Software, Environments, and Tools, SIAM, Philadelphia, PA, 1999.

[5] M. Beck, T. Moore and J. Plank, "An End-to-End Approach to Globally Scalable Network Storage", Proceedings of the ACM SIGCOMM 2002 Conference, Pittsburgh, PA, USA, August 19-23, 2002.

Vita

Patrick Alan Lynn was born in Atlanta, GA on March 9, 1969. He lived in Marietta, GA were he attended elementary, middle, and high school. After receiving his high school diploma from George Walton High School in 1987, Patrick attended the Georgia Institute of Technology in Atlanta, GA. In March of 1992 he received his Bachelor of Science degree in Industrial Design from Georgia Tech. After working several years in the private sector he enrolled at the University of Tennessee to pursue a Master's degree in Computer Science and is expected to graduate December 2002.