Integrating Interactive Digital Maps into a Digital Library

A thesis

submitted in partial fulfilment

of the requirements for the Degree

of

Master of Science

at the

University of Waikato

by

Samuel J. McIntosh

THE UNIVERSITY OF
WAIKATO
Te Whare Wānanga o Waikato

University of Waikato

2010

Abstract

Digital libraries and digital maps are two fast-growing technologies in the world of computing. In this thesis we have explored using digital maps to enhance the functionality of digital libraries. The Greenstone 3 digital library system was augmented though the use of the digital mapping system, Google Maps. An automatic place name recognition and disambiguation system was created to obtain geographical information from documents as they were viewed. This extracted information was presented as a map with markers showing the location of the places within the text of the document.

We evaluated the system by performing a user study and an analysis of the speed, efficiency and accuracy of the place name recognition and disambiguation system. Participants in the user study completed most of the tasks easily and made comments expressing their satisfaction with the system. Analysis of the place recognition and disambiguation system was also positive, as the system was fast, relatively efficient and was highly accurate.

Acknowledgements

I would like to thank my parents and family for their love and ongoing support throughout this process, my wife Katrina for her patience, love and encouragement and my cat for providing comic relief.

I would also like to thank my two supervisors: David Bainbridge for his advice and assistance, and Dave Nichols for his guidance and for painstakingly checking through section after section of my thesis.

Funding for my Masters scholarship came from the FRST project "International multimedia management and delivery".

Contents

1 Introduction **1**

1.1 Walkthrough 1

1.2 Thesis structure 7

2 Background **10**

2.1 Similar systems 10

2.1.1 GIPSY 11

2.1.2 Pre-NewsExplorer 12

2.1.3 NewsExplorer 13

2.1.4 Informedia Digital Video Library 14

2.1.5 Perseus Project 17

2.1.6 Web-a-Where 19

2.1.7 G-Portal Digital Library Project 20

2.1.8 Summary 21

2.2 Interactive visualisation 26

2.2.1 Fisheye view 28

2.2.2 Column view 34

3 Design and implementation **36**

3.1 Requirements and technology overview 36

3.1.1 Digital library 38

3.1.2 Digital mapping system 39

3.1.3 Gazetteer 42

3.1.4 Place recognition and disambiguation system 44

3.1.5 Place information storage system 44

3.2 System structure 45

3.3 Implementation details 52

3.3.1 Contributing technologies 53

3.3.2 Pseudo AJAX Loading 55

3.3.3 Google Maps 59

3.3.4 Place name recognition 63

3.3.5 Place name disambiguation 66

3.3.6 Place information retrieval 69
3.3.7 Spatial searching 72
3.4 Interface design 76
3.4.1 Page layout 76
3.4.2 Status Area 80
3.4.3 Text presentation 83
3.4.4 System interactions 90

4 Evaluation **98**
4.1 User study 98
4.1.1 Pilot tests results 98
4.1.2 Full user test results 100
4.2 Place name disambiguation accuracy 107
4.3 Gazetteer trie efficiency 113

5 Conclusion **118**
5.1 Contributions 118
5.2 Future work 122

References **124**

Appendix **128**

A User study material **129**

List of Figures

1.1 The start page of Greenstone. 2
1.2 The "about" page of the MGPP demo collection. 3
1.3 A standard text query form. 3
1.4 The text query results page demonstrating the additional ATLAS functionality. 4
1.5 A standard Greenstone document view enhanced with ATLAS. 5
1.6 A menu displaying interaction options for the currently selected place name. 6
1.7 The ATLAS spatial searching interface. 7
1.8 Creating a spatial query for the area around Hamilton. 8

2.1 An example of the visualisation system used in GIPSY (from [WP94]). 11
2.2 An example of the WorldKit visualisation system used by the NewsExplorer system (from [PKS+06]). 14
2.3 An example of a map (bottom half) being shown in sync with a video (from [COH99]). 16
2.4 The fisheye view in the FishNet browser (from [BLH04]). . . . 29
2.5 The fisheye source code editor designed by Jakobsen and Hornbæk (from [JH06]). 31
2.6 The Document Lens system (image taken from [RM93]). . . . 34

3.1 The structure of ATLAS. 37

3.2 A comparison between the (a) standard Greenstone skin and the (b) development skin used by ATLAS. 40
3.3 The original structure of the system. 47
3.4 The second structure of the system. 50
3.5 The final structure of the system. 51
3.6 The page loading cycle. 56
3.7 Two examples of the syntax used to refer to GWT-compiled methods in JNSI. 58
3.8 The *localhost* Google Maps key being inserted into ATLAS. . . 59
3.9 Two different types of map marker: (a) A standard Google Maps marker and (b) customised polygon markers. 61
3.10 A basic trie where "ab" is match and all other nodes are not matches. 64
3.11 The per-place scoring procedure. 68
3.12 The final scoring adjustments. 70
3.13 The table layout of the PostgreSQL/PostGIS database. 72
3.14 The first spatial search method. 75
3.15 The second spatial search method. 75
3.16 Two variations of the horizontal split layout. 77
3.17 The original full vertical split layout of the system. 78
3.18 The full split layout with the Greenstone header and footer sections extending the full width of the page. 79
3.19 The final layout of the system. 80
3.20 Two of the status area position concepts. 82
3.21 The status area showing two updates. 82
3.22 The animated GIF used on the status area. 83
3.23 The document text in fisheye view. 84
3.24 The line finding algorithm used to set up the fisheye view. . . 85
3.25 The compaction view design. 86
3.26 The document text in column view. 88

3.27 The heuristic algorithm used to arrange document text into columns. 89
3.28 The document specific menu used in ATLAS. 92
3.29 Switzerland with its neighbouring countries. 95
3.30 Single box and multi-box spatial query examples. 95
3.31 Multi-shape and polygon query examples. 96

4.1 The browsing method selection page. 104
4.2 Data structure memory usage in bytes. 114
4.3 The speed of each of the structures in milliseconds. 115
4.4 The time taken for each structure to parse the Unicode-oriented data set (in milliseconds). 116

List of Tables

2.1 System comparisons. 27

2.2 Jakobsen and Hornbæk's [JH06] user test average satisfaction scores (and standard error of the mean). Significantly better scores are shown in bold. 32

3.1 Several example Gazetteer entries. 43

3.2 Character counts in the gazetteer. 65

4.1 The 56 top scoring combinations of the disambiguation system scoring parameters (with 98.62% precision). 110

4.2 The three lowest scoring combinations of the disambiguation system scoring parameters (with 73.37% precision). 111

4.3 Summary statistics of the PB, PCS and PBL parameters. . . . 111

4.4 The statistics for the PB parameter when grouped by the PCS parameter. 112

4.5 The statistics for the PBL parameter when grouped by the PCS parameter. 112

4.6 Memory usage of the various trie structures. 114

4.7 Task completion time for each of the data structures. 115

4.8 Time taken for each of the data structures to parse the Unicode-oriented data set. 116

Chapter 1

Introduction

In this thesis we will present ATLAS, the mAp-inTegrated digitaL librAry System. The system is designed to enrich digital library content with additional geographic information. This is achieved by augmenting an existing digital library with features such as: maps displaying the location and information about the places mentioned in a document, locating documents that mention places within a specified geographical area (spatial searching) and by extending the current text searching facilities with an additional map-view displaying the places mentioned in each of the documents that are found. We will now show several examples of the system in use.

1.1 Walkthrough

The system starts by loading in the home page of Greenstone — the digital library used for development. As this first page is loading the system also begins loading the data structure necessary for quickly locating the places in the documents we encounter shortly. Figure 1.1 shows the Greenstone start up page with a message from ATLAS letting the user know that this data structure is being loaded and that they will have to wait before the system is fully operational. For the purposes of this walkthrough we will be using the MGPP Demo collection which is a small subset of the Humanity Development

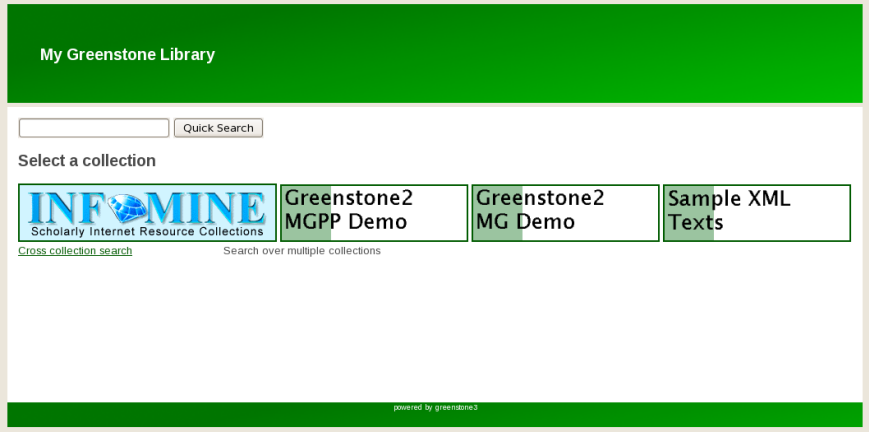


Figure 1.1: The start page of Greenstone.

Library collection[1].

Once the Greenstone home page is loaded the user selects the collection they want to open and are taken to the "about" page of that collection (shown in Figure 1.2), the "about" page is effectively the home page of the collection. On this page the user is given several choices about how they want to locate the document(s) they are searching for. These options include: browsing documents by classifiers such as title and subject, performing a full document text search or even searching an area of the world for documents that mention places within that area. As the first step in the walkthrough we will demonstrate the text searching functionality and how it has been enhanced by ATLAS.

When the user clicks "Text Search" they are taken to a page with a standard text query HTML form like the one shown in Figure 1.3. The user submits their query as usual and they are taken to a page showing the results of the search (Figure 1.4). Here we see ATLAS in action, once the results page is loaded ATLAS begins loading each document in the background and scans their content, looking for places that are mentioned within them. Each document in the results set is given a colour, places located in that document are marked on the map in the same colour to make it clearer to the user which

[1]Located at http://www.nzdl.org/cgi-bin/library.cgi?a=p&p=about&c=hdl

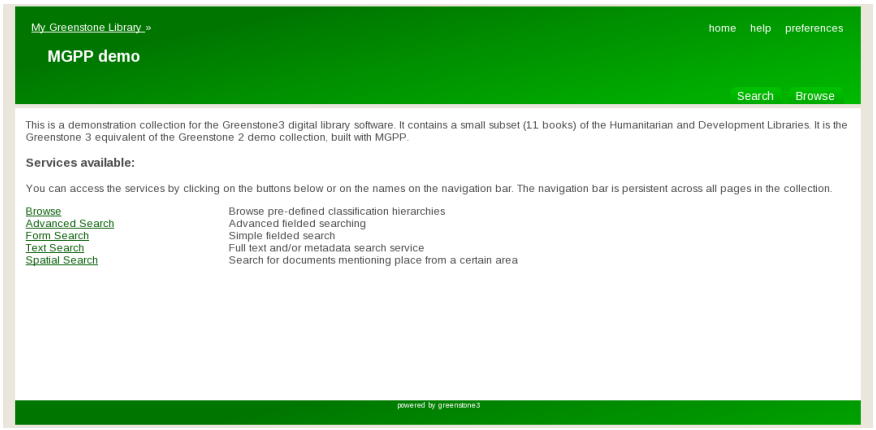


Figure 1.2: The "about" page of the MGPP demo collection.

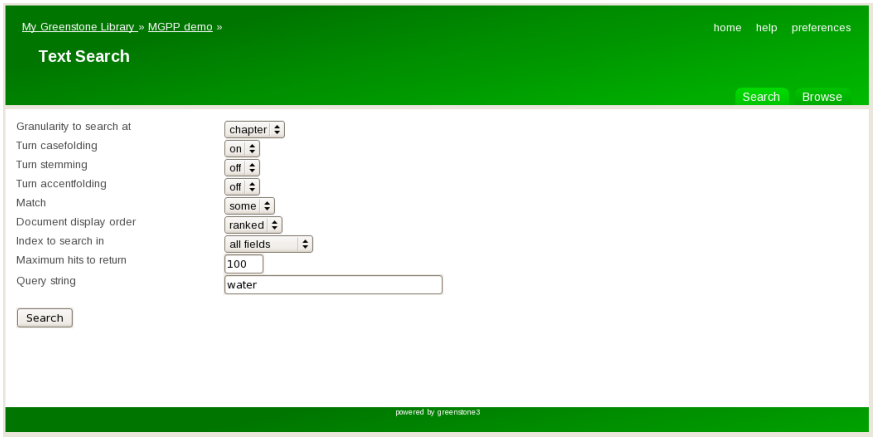


Figure 1.3: A standard text query form.

places belong to each document. The places belonging to each document can be toggled on and off by using the check boxes located on the left of each document title. From this view the user may select a document to read from the list of results. Next in the walkthrough we describe how the standard document view has been enhanced with ATLAS.

ATLAS enhances the standard document view of Greenstone by adding a map to the right-hand side of the screen like on the text results page. This map contains markers showing the locations of the place names that have been found in the document text. These place names are matched to their most likely candidates based on the contents of the rest of the document. These

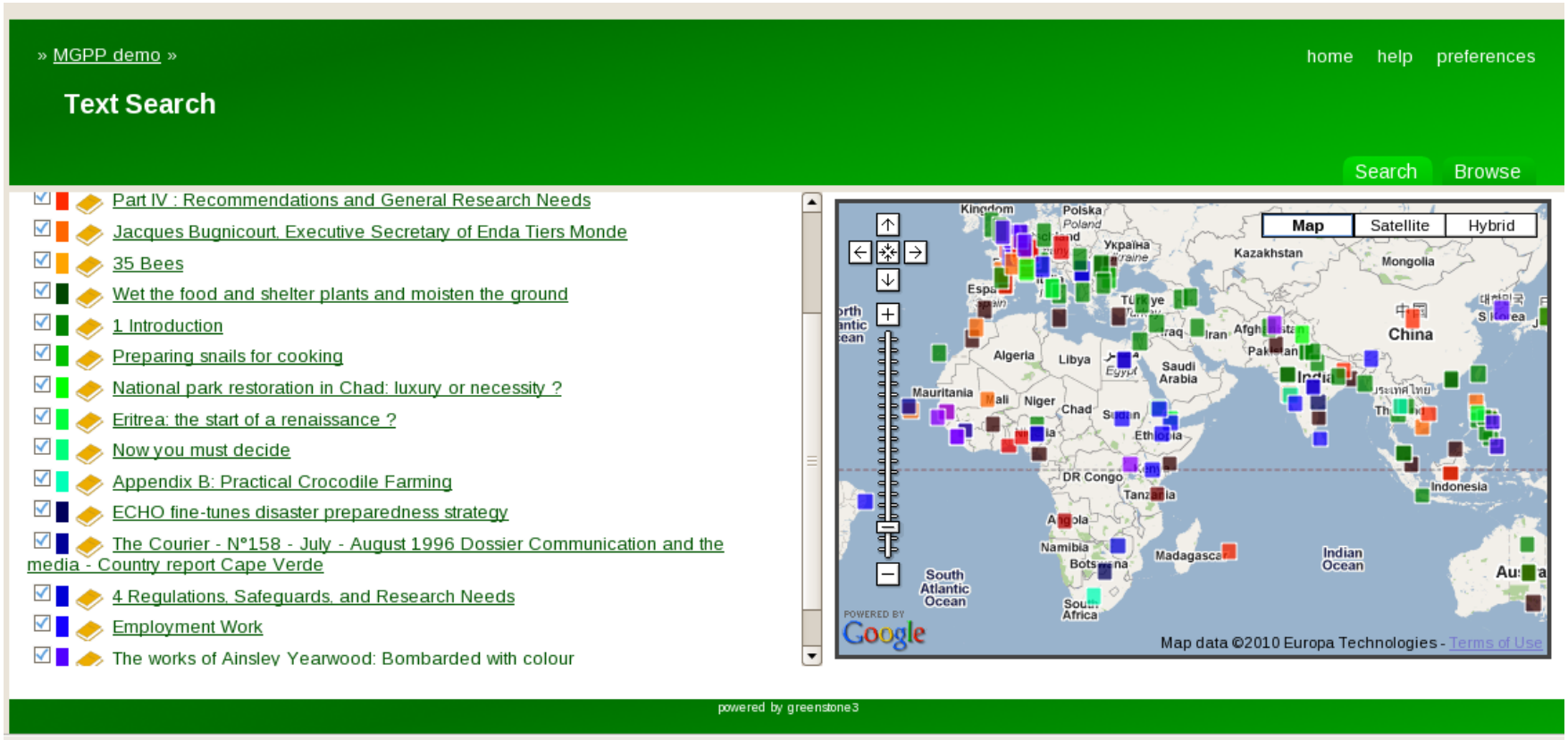


Figure 1.4: The text query results page demonstrating the additional ATLAS functionality.

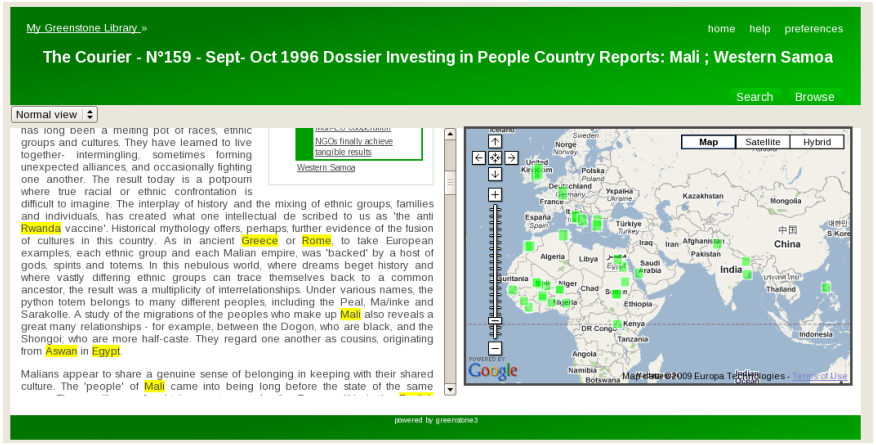


Figure 1.5: A standard Greenstone document view enhanced with ATLAS.

places in the text are also highlighted to allow the user to see where they occur in the document. Figure 1.5 shows this enhanced document view.

The user is also able to interact with the system to find out more information. For example, each highlighted place name can be moused over to produce a menu that gives several options specific to that place name. This allows the user to perform tasks such as: centering the map on this specific place, highlighting the place on the map, highlighting the place in the text or correcting the disambiguation done by the system (such as "Cambridge" being incorrectly classified as Cambridge in England when the text refers to Cambridge in New Zealand). An example of this menu functionality is shown in Figure 1.6. Here the user has moused over "London" and have chosen to change which London the place name refers to.

Because of the decreased space now available for the document text — due to the map filling half of the screen — two alternative text views have been implemented that are designed to make better use of the limited available space. These are the fisheye and column views which are designed to allow the user to see more of the document than would normally be possible with the standard text view. These are demonstrated and discussed in more detail in Section 3.4.3.

Finally, in addition to augmenting the text searching functionality of Green-

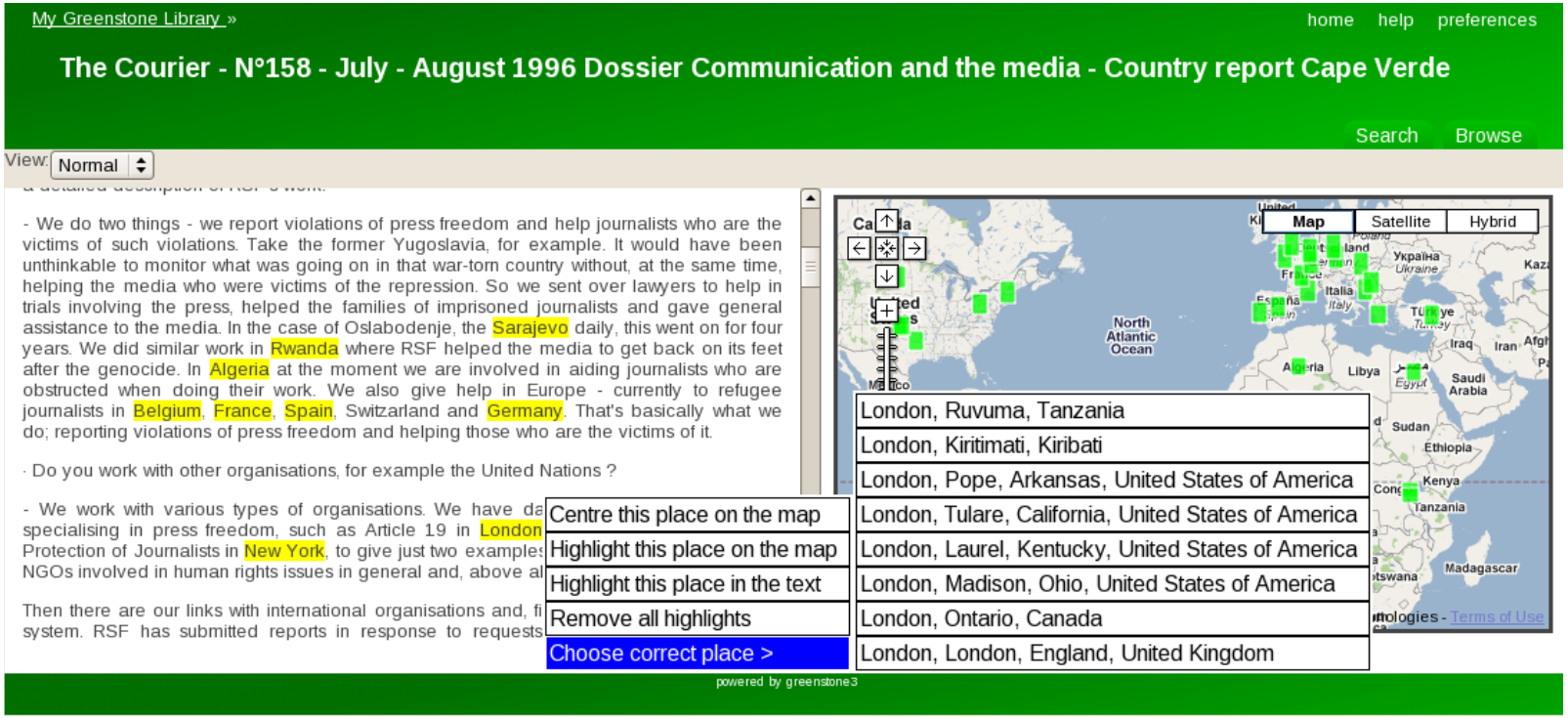


Figure 1.6: A menu displaying interaction options for the currently selected place name.

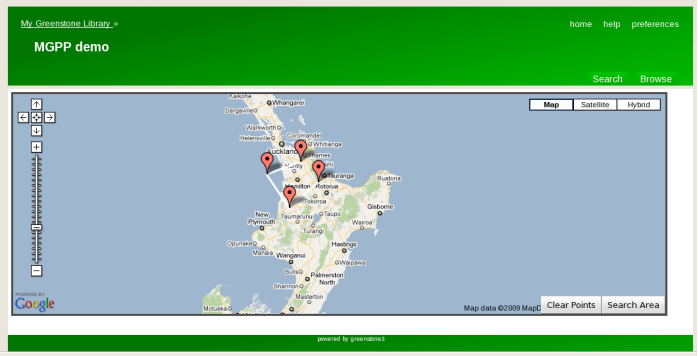


Figure 1.7: The ATLAS spatial searching interface.

stone, ATLAS provides searching functionality of its own in the form of spatial searching. The user selects an area that they wish to locate documents related to and clicks the "Search Area" button to perform the search. The documents that are found are then displayed in order of relevance on the opposite side of the web page.

Selection is performed by clicking on the map, which creates a point at that location. Successive points are joined together to create a polygon which is then used in the spatial query. If the user makes a mistake they can either drag the points to a new location or clear all of the points if they wish to start again. Figure 1.7 shows the spatial search interface with an area around Hamilton, New Zealand selected. Figure 1.8 shows step by step how a spatial query is created, the final step (shown in Figure 1.8(e) is not actually necessary as ATLAS will automatically close off the polygon before it performs the query.

1.2 Thesis structure

In Section 2 we discuss work related to the development of ATLAS in the form of similar systems and work done in the area of visual text manipulation. In Section 3 we discuss the design and implementation of the ATLAS system. This covers the technologies used, the structure of the system, specific implementation details and the design of the user interface. Section 4 evaluates three key aspects of the system: the user interface, in the form of a user study;

Figure 1.8: Creating a spatial query for the area around Hamilton.

the efficiency of the data structure used for place recognition, discussed in Section 3.3.4 and finally the accuracy of the place name disambiguation system. In Section 5 we present our conclusions and discuss possible future work.

Chapter 2

Background

In this chapter we review systems related to ATLAS in the areas of digital libraries combined with digital maps and automatic place name recognition and disambiguation. We then present research done in the field of information visualisation, specifically focusing on the two visualisation methods that have been implemented in ATLAS: the fisheye view and the column view.

2.1 Similar systems

In this section we discuss several systems related to various aspects of ATLAS. Four of these systems — specifically the GIPSY, Perseus, Informedia and G-Portal systems — used a digital library combined with automatic place name recognition and disambiguation for the purposes of geographic indexing. Four of the systems — specifically the GIPSY, NewsExplorer, Perseus and Informedia systems — used automatic place name recognition and disambiguation to produce a digital map of the content they analysed. The Web-a-Where system uses automatic place name recognition and disambiguation to assign a locality to arbitrary documents.

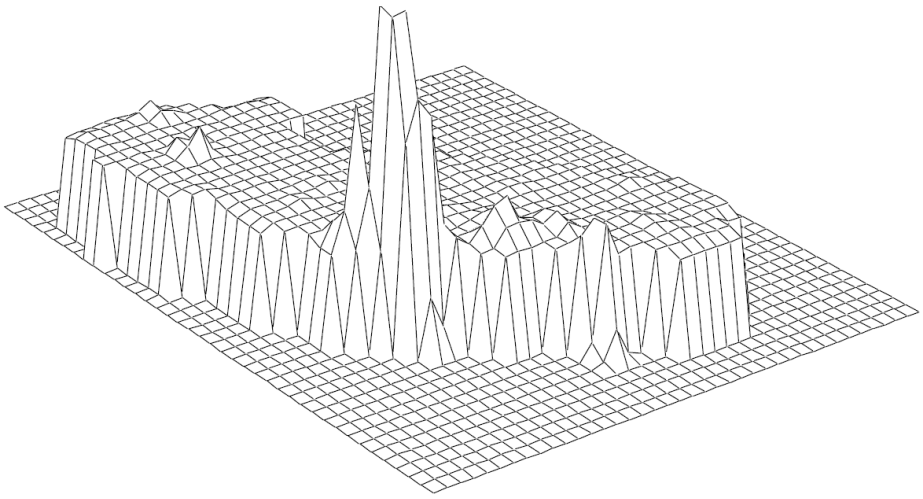


Figure 2.1: An example of the visualisation system used in GIPSY (from [WP94]).

2.1.1 GIPSY

In 1994 Woodruff and Plaunt created the Georeferenced Information Processing SYstem (GIPSY) [WP94]. The focus of the system was automatic geographical indexing of text documents that could then be searched with a spatial interface. The system was also capable of some basic visualisation that created a 3-dimensional polygon mesh where the x and y axes represented the latitude and longitude of the place being visualised and the z axis represented the frequency that each place within the area was mentioned (Figure 2.1 shows an example). This created a skyline where it was clear which geographical areas the document(s) focused upon by the height the mesh rose out of the surface at different points on the mesh. For example, if a document about the state of Nevada in the United States of America makes frequent references to Las Vegas, then in a map of the document the area of interest would be Nevada so the shape of Nevada would be raised out of the mesh. Also because Las Vegas is frequently mentioned the shape of Las Vegas would be raised out of the map as well (higher than the level of Nevada as the z height is cumulative).

To disambiguate between words that are places and words that are not places GIPSY used a gazetteer combined with set of stop-words (words that are never to be considered place names). All places that matched in the

gazetteer and were not in the list of stop-words were treated as place names. Disambiguation was not discussed in the article, it is possible that it was not necessary due to the area of focus being small enough to not contain place names that mapped to more than on place.

The gazetteer used by the GIPSY system was created by combining information from multiple sources. One of the place name data sets used by GIPSY is the US Geological Survey's Geographic Names Information System[1] (GNIS) which contained over 60,000 geographic place names in California — their area of interest. They also used another set of data, also from the US Geological Survey, the Geographic Information Retrieval and Analysis System (GIRAS) [AHRW76]. This data contains over 60,000 entries, each with many points describing features of interest such as parks, harbours and bridges. These points were converted to polygons to use in their visualisation system.

2.1.2 Pre-NewsExplorer

In 2004 Pouliquen et al. created a system designed to recognise place names in multilingual texts [PSIDG04]. To achieve this they used a combination of Gazetteers containing place names in 8 different languages for place name recognition. For place name disambiguation they used a series of heuristics. They used the Global Discovery gazetteer[2] and used some of the resources created by the KNAB project[3]. The Global Discovery gazetteer is a commercial gazetteer with over 850,000 places. The researchers were mostly interested in European places names however and therefore reduced the gazetteer down to less than 100,000 entries. The KNAB database contained less than 100,000 entries, however this would still have been useful for their purposes as it contains many alternative names for places.

Because the aim of the program was to create a system capable of geocoding texts of many different languages, they decided against the use of per-language

[1] http://geonames.usgs.gov/pls/gnispublic/
[2] Available at http://europa-tech.com
[3] http://www.eki.ee/knab/knab.htm

linguistic rules. They believed that these would have improved the accuracy of the system but they claimed that developing these rules for the multiple languages their system used was out of their reach.

Their system used a combination of two techniques to decide which of the possible places matches the place name in question. The first technique used the concept of the "importance" of a place. Each place in their database is given a value from 1 to 6 that denotes how "important" that particular place is. For example, 1 means that this place is the capital city of a country and 6 means that the place is a small town or village. These classifications are found in the Global Discovery gazetteer and have not been created by the researchers themselves.

The second technique is to use places that have been located nearby in the text to help decide on the correct disambiguation. For example they use "Victoria is the business and cultural centre of the Seychelles" to decide that "Victoria" refers to Victoria in the Seychelles rather than Victoria in Hong Kong. How precisely these two techniques interact is unclear (e.g., in what situations does each have priority over the other?) but they report good results in their evaluation of the system.

2.1.3 NewsExplorer

In 2006 Pouliquen et al. presented an extended version of this system which had been integrated in to a system called NewsExplorer[4] [PKS+06]. This version of the system had improved the amount of languages it could process from 8 up to 15. The new version of the system also improved upon the disambiguation method used in the first system. A minimum distance heuristic was added to find the minimum distance from an ambiguous place to an unambiguous place. In the extended system Pouliquen et al. used a scoring method to combine the outputs of their various heuristics, with each heuristic contributing part of the score.

[4]http://press.jrc.it/NewsExplorer

Figure 2.2: An example of the WorldKit visualisation system used by the NewsExplorer system (from [PKS+06]).

Another area where this system was improved was in the area of visualisation. At the time of writing they were using three different methods of visualisation. Along with the SVG (Support Vector Graphics) visualisations they had used in the previous system two new technologies were adopted into the system. The first, WorldKit,[5] is a freely available tool that takes a RSS (Really Simple Syndication) file as input with article text and a latitude and longitude. It can then display these articles on the map at the given coordinates (Figure 2.2 shows an example of the WorldKit visualisation system).

The second new visualisation technology used was Google Earth,[6] which is an alternative way of viewing this information. The KML 2.0[7] (Keyhole Markup Language) specification is used to define the information to place on the virtual globe.

2.1.4 Informedia Digital Video Library

The Informedia Digital Video Library project[8] is an ongoing research project that began in 1994 at the Carnegie Mellon University. This project focuses

[5] http://worldkit.org
[6] http://earth.google.com
[7] http://code.google.com/apis/kml/documentation/
[8] http://www.informedia.cs.cmu.edu/

on allowing video retrieval to have all of the same capabilities as text retrieval while at the same time making use of the visual properties of video to provide rich delivery of information.

In 2000 Christel et al. reported on using interactive maps in this digital video library [COH99]. At the time of writing this paper they had accumulated over 2000 hours of video — growing at a rate of more than 10 hours a day — from sources such as the British Open University, the Discovery Channel and NASA. They claim that when the video library was much smaller (hundreds of hours of video rather than thousands), simple text queries were enough to provide a small set of satisfactory search results. As the repository grew however it became clear that using text queries was no longer providing an adequate subset of results as each query could return hundreds of video segments, potentially overwhelming the user.

As a solution they developed an information-visualisation system that allowed users to browse and search for video segments based on key words and date. They also realised that there was a rich set of information that was yet to be utilised in the form of geographic information. To utilise this information they began the development of a system that could automatically extract this information from the narrative of the video (obtained through the use of the Carnegie Mellon University Sphinx speech-recognition engine [HAHR93]). The system also extracted any words that had been shown on the screen through the use of OCR and checked these for place names. This information could then be used to provide spatial video searching and display map footage relative to a video in sync with the content as shown in Figure 2.3.

As a gazetteer they have used a subset of the Environmental Systems Research Institute's (ESRI) world gazetteer which contains over three million entries. The subset they used consisted of about 300 countries, states and administrative entities, and about 17,000 major cities. This small subset was likely to still be sufficient for their needs however as the majority of the video

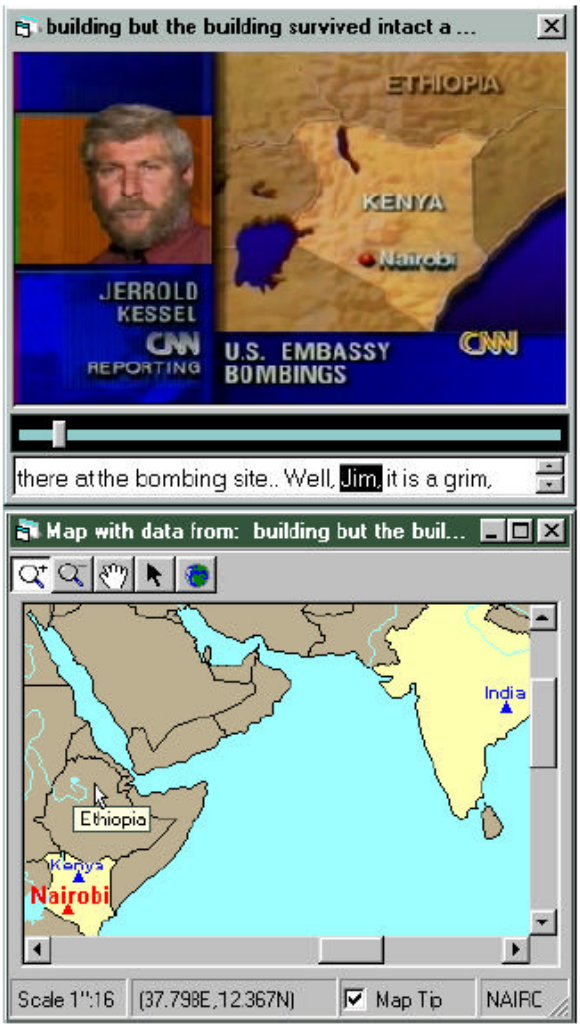


Figure 2.3: An example of a map (bottom half) being shown in sync with a video (from [COH99]).

segments are news and documentary footage, which do not often mention places other than countries and major cities.

The place name recognition and disambiguation methods developed by the researchers were relatively simple. To prevent common words from matching as place names (e.g., Of in Turkey, Data in India or Many in France) they were simply removed from the list of possible place names. The precise method they used to disambiguate between places with the same name was unclear, but they did mention that within in this process they made use of the local context in which the place name had been found. The use of a smaller gazetteer also would have simplified this disambiguation process as there would be far fewer places with duplicate place names to attempt to disambiguate between.

A total of nearly 20,000 geographic references were found within the roughly 40,000 video segments. To help query video segments that had been geographically indexed they provided a spatial searching system. The user was able to draw a rectangle around an area on a map and the system returns video segments related to that area. They also discussed another spatial searching method that allowed the user to click on a country for example and have the system find all of the video segments relating to that country. This method of spatial querying is intuitive, fast and powerful, however it is not clear whether or not this had actually been implemented into their system at the time they wrote the article.

2.1.5 Perseus Project

The Perseus project was founded in 1987 at the Tufts University in Massachusetts, USA. The initial goals of the project were the collection and presentation of materials relating to ancient Greece. The Perseus Digital Library was created in 1995 as the project was moved to the Internet. Since then, the project has grown and now includes material from both Greek and Roman origins, material from the 19th century United States and more.

In 2001 Smith and Crane from the Perseus project reported on disam-

biguating geographic names in their historical digital library [SC01]. At the time of writing they had over 70 million English words in their digital library with over one million of those words being place names. In a collection of this nature and magnitude the benefits of maximising the geographic connectivity of its resources are enormous. It was clear to the researchers however that manually tagging this amount of data was impractical and that some form of automatic tagging system — or at least machine assisted tagging system — was needed to fully unlock the full potential of this resource.

Developing a system to perform this automatic tagging in a historical context provided some interesting challenges for the researchers. They estimated that 92% of place names within the collection could refer to more than one place. This was due to two main factors, the first being places that have been settled for a long time (such as those in Asia, Africa and Europe). Over time, places within these areas are likely to have been known by more than one name. The second factor contributing to the complexity of this problem was that, in countries such as the United States of America, places are often named after people. This decreased the likelihood of the name of a place having a unique name and as a result, place names in these areas often could refer to more than one place.

As a gazetteer they used a combination of several different sources such as the Getty Thesaurus of Geographic Names[9] and Crunchley's gazetteer of London. In total the gazetteer that they used contained over one million place names.

As part of their place name recognition procedure they first attempted to locate all proper names and split them into one of three categories; person names, places or dates. References to places or words where the classifier was uncertain were searched for in the gazetteer. Once a reference was located it then usually required disambiguation (assuming it was one of the 92% of places where the place name is ambiguous).

[9]http://www.getty.edu/research/conducting_research/vocabularies/tgn/

In the simplest of cases the local context would provide the necessary information for disambiguation such as “Cambridge, New Zealand” or “Newcastle, Australia”. They also use local context in instances where multiple places are mentioned in the same paragraph. They assume in such cases that the places mentioned are likely to be close to each other and used a distance algorithm to work out the appropriate match. In some cases, such as news articles from a newspaper, places can be mentioned without any local contextual hints. In this situation the context of the document can be used. For example, if the article is known to be from an Australian newspaper then this can be used to assume that mentions of Newcastle will be Newcastle in Australia. In all other situations places were scored on the following factors: their distances to the places around them in the text, the proximity to all of the places in the text and its relative importance (i.e., countries are more important that cities).

2.1.6 Web-a-Where

In 2004 Amitay et al. developed a system, called Web-a-Where, for tagging the places mentioned in arbitrary web pages. The system also has a secondary focus on assigning a locality to each web page as a whole [AHSS04].

Their first step in tagging a web page was to identify words that are potential place names. This requires disambiguating between words that are place names and words that match in the gazetteer but are not place names (e.g., “Of” in Turkey and “Humble” in Texas, USA). For this procedure they compiled a list of words that were more likely to be common words and would be discarded if they were matched during the place name recognition phase. The only exception to this is if the word is followed by another place name indicating that the text actually refers to this unlikely place (e.g., “Of, Turkey” or “Many, France”). It is not explicitly clear how they disambiguated between words that were the names of people but matched in the gazetteer (such as “Anna” in Ohio, USA and “Stephen” in Minnesota, USA) though it is possible that these were included in their list of common words.

For a gazetteer they used the World Gazetteer as well as information from GNIS[10] (Geographic Names Information System) for US names, UNSD[11] (United Nations Statistics Division) for countries and continents and the ISO 3166-1 code lists[12] for country abbreviations. Despite the World Gazetteer containing over 300,000 places they reduced it down to 40,000 places. Most of this reduction would have been from their decision to remove the entries of places that had a population of less than 5000. This was likely done as an attempt to improve the disambiguation accuracy of the system.

Once the places in a web page were found, any that were ambiguous were taken through another series of steps. These steps use a small set of heuristics that assigned each potential place a confidence value based on factors such as its surrounding context in the document and its population compared to other places of the same name. Once the places were disambiguated the system would then calculate the locality for the document as a whole. Each disambiguated place was given a score (e.g., Hamilton, Waikato, New Zealand is given the score X^2) and each of its parents were also given part of that score (Waikato, New Zealand is given X^2Y score and New Zealand is given X^2Y^2 where $0 < Y < 1$ in both cases). The resulting scores are then used to calculate the locality of the article.

2.1.7 G-Portal Digital Library Project

In 2005 Zong et al. created a system for assigning place names to web pages containing geographic material for their digital library system called G-Portal [ZWS+05]. For place name recognition GATE[13] is used (or more specifically the ANNIE module). They extended the standard GATE gazetteer (which contains over 6,000 major place names) with the US Census 2000 gazetteer[14] information which contains 52 states, 25,375 cities, 3,219 counties and 36,351

[10] http://geonames.usgs.gov
[11] http://unstats.un.org/unsd
[12] http://www.iso.org/iso/country_codes/iso_3166_code_lists.htm
[13] http://gate.ac.uk/
[14] http://www.census.gov

county subdivisions. This data was sufficient for them as the web pages used for their research were mostly about the United States.

Their disambiguation process consists of four main steps. In the first step, the local context of the place name is checked for place name senses such as "state" or "city". If a place name sense is found then all potential places for that place name that do not match that sense are discarded. In the second step, place names that have only one potential candidate are added to the list of disambiguated place names. In the third step, places that have already been disambiguated are used to help disambiguate other place names that are still ambiguous, for example, if an instance of "Colorado" has been found to be referring to the state of Colorado then we can assume that "Denver, Colorado" refers to Denver city in Colorado state. The fourth and final step is to compute the distance between each ambiguous place to find its nearest disambiguated place. The ambiguous place with the shortest distance between it and a unambiguous place is added to the list of disambiguated places.

Like the Web-a-Where system discussed earlier, the goal of this system was to assign each page a geographical locality. The algorithm used to calculate this locality splits the document into sections and creates a subtree of the gazetteer for each section. Factors such as place name frequency and the distribution of a place's children (a place is considered a child of another place if the child resides within the parent place) are used to decide on the specific place name labels to assign to this specific section, resulting in a web page potentially being defined by multiple place name labels.

2.1.8 Summary

Here we will compare and contrast these systems in several areas including: display methods, gazetteer size and methods of recognising and disambiguating place names. Table 2.1 at the end of this section summarises these comparisons.

Visualisation method

Five of the seven systems discussed were capable of displaying a map to visualise geographic information. These methods varied from system to system based on factors such as resource constraints or the technology available at the time of writing.

The earliest system, GIPSY, took several sets of shapes and coordinates to create a 3D mesh. Each shape defined in the mesh area increased the height of the mesh at that point. Intersecting shapes increased the height of the mesh additively at the points of intersection, resulting in frequently mentioned areas being raised higher out of the mesh.

The 2004 version of the NewsExplorer system was capable of using three different methods to display maps [PSIDG04]. The first method was by using "shape files"[15] which could be used to create GIF, PNG or JPG files using the GD Perl module.[16] The researchers claimed that this method was computationally heavy and slow but had the advantage that it was independent of commercial software.

The second display method provided by the system produced maps using the DMA[17] (Digital Map Archive) tool from the JRC's ISFEREA project. This program is a commercial tool capable of rendering maps quickly and is also capable of overlaying different layers of information over the map such as roads and population density.

The final method used for displaying maps is by using the SVG[18] (Scalable Vector Graphics) format defined by the W3C (World Wide Web Consortium). This method provides a relatively simple looking map but has the advantage that it renders quickly. The article does not mention how the SVG files for countries were obtained.

The 2006 version of the NewsExplorer system was capable of displaying

[15]The ArcGIS Explorer tool from http://www.esri.com/ allows downloading the shape files for countries

[16]http://search.cpan.org/dist/GD/

[17]http://dma.jrc.it/

[18]http://www.w3.org/Graphics/SVG/

maps using two new display technologies. The first new technology was a web mapping system called WorldKit, which is a free, open source system that uses flash to display the contents of specially formatted RSS files. The second new display technology was Google Earth[19]. Google Earth is run as separate program that can import a KML file from the NewsExplorer system. This KML file contains the position to put a marker on the globe and the article content.

The Informedia digital library system used the commercial ESRI MapObjects[20] library to visualise its geographical content. As MapObjects was a library, it meant that they were able to program their own interfaces and functionality to fully utilise its capabilities.

It is assumed that the Perseus project used Google Maps[21] as their visualisation tool but this is not specifically mentioned in the article. This assumption was made because the Perseus system as it exists in 2010 uses Google Maps.

The decision of what visualisation technology to use for ATLAS is discussed in Section 3.1.2.

Place name recognition methods

Before place name disambiguation is performed, place names must first be located in the document. This requires the system to scan the document and identify place names, while at the same time identifying words that appear to be place names (i.e., a match is found in the gazetteer) but are actually other entities such as person names or regular words.

All of the systems discussed in this chapter compare words against a gazetteer to identify whether or not a word is potentially a place name. The way in which each system tests whether or not a potential place name is actually a place name differs for each system.

Both the GIPSY and Informedia systems simply used a gazetteer that had

[19]http://earth.google.com/
[20]http://www.esri.com/software/mapobjects/index.html
[21]http://maps.google.com/

been altered to remove place names that were likely to be incorrectly identified. These systems were then able to assume that every word that matched in the gazetteer was a place name.

Both the earlier and later versions of the NewsExplorer system and the Web-a-Where system used a list of stop-words to prevent the system from recognising words that were unlikely to be place names. If a located place name was also found in the stop-word list then it was discarded.

The system used to recognise places in the historical digital library at the Perseus project first attempts to recognise all proper names, it then classifies those names into categories (i.e., person names, organisation names, locations etc.). The words that are found to be locations and words that the parser is uncertain about are then moved on to the next step in the disambiguation.

The G-Portal system used the GATE ANNIE module to locate words that were to be considered place names. This was achieved by extending the standard ANNIE gazetteer — which contains around 6,000 place names — with over 60,000 new entries.

The place name recognition system used by ATLAS is discussed in Section 3.3.4.

Place name disambiguation methods

There were several methods of disambiguation used by the different systems, these included: methods based on linguistic rules (e.g., understanding "Cambridge, England" to mean Cambridge in England); methods based on other heuristics such as minimum geographic distance, population comparison, the importance of a place (i.e., a capital city is more important than other cities), examining the local context (i.e., other surrounding place names) and score-based methods.

The article written on the GIPSY system does not discuss a disambiguation method. As their geographical focus is small it is likely that disambiguation between places of the same name is not necessary.

Both the earlier and later versions of the NewsExplorer system use the idea of importance to help decide which place is the best candidate for each place name. This is also combined with an analysis of the local context of the place name. The later version of the system also adds a geographical distance calculation and combines the output of these various heuristics into a score for each place. In a similar manner, both the Perseus project and G-Portal use both local context and geographical distance calculation to disambiguate between places.

The place name disambiguation method used by the Informedia digital video library is made clear in the article. The article does mention of the use of local context but does not mention any specific details of the implementation used.

The disambiguation procedure used by the Web-a-Where relies on the use of confidence values. The ambiguous places located by the system are processed by a series of heuristics (including examining the local context of the place name and using population comparisons) that calculate their confidence values.

The place name disambiguation system used by ATLAS is discussed in Section 3.3.5

Gazetteer size

The size of the gazetteers used by each of the systems varied significantly. Several of the systems focused on higher level disambiguation (i.e., countries and major cities) or specific areas of interest (e.g., Europe) and therefore required smaller gazetteers in order to achieve their goals. The GIPSY, Web-a-Where, G-Portal and the Informedia systems all use small and focused gazetteers designed for their specific purpose using gazetteers with around 60,000, 40,000, 60,000 and 17,000 entries respectively. The NewsExplorer and Perseus systems however chose to use wider gazetteers for lower level disambiguation, using gazetteers with 100,000 and 1,000,000 entries respectively. Lower level disambiguation is a more risky approach due to there being many place names

in the world that either map to people names or to more than one place. By using a smaller gazetteer this risk is significantly reduced as many of these ambiguous places are removed due to them not being major cities. For ATLAS we have chosen to design the system to use a low level of place name recognition with a gazetteer with over 300,000 entries.

Despite using such a large gazetteer, the disambiguation problem faced by the ATLAS system is still significantly smaller than the disambiguation problem that was faced by the developers at the Perseus project. The size of their gazetteer (1,000,000+ entries) combined with the large amount of ambiguity of the place names in the collection meant that this disambiguation problem was significantly more complex than the disambiguation problem faced by ATLAS. This because the ATLAS gazetteer is smaller in comparison and as system is likely to deal mostly with modern texts it will not often have to handle cases where places were known by a different name in different point in history — which is the case in the historical digital library at the Perseus project.

Some gazetteers — such as the one used by the GIPSY system — provide extra information such as the shape of the each place (defined as points in latitude and longitude coordinates). This extra information was useful to them as it helped to add the places to the 3D visualisation but it would have not been a helpful resource to consider using in ATLAS. Latitude and longitude coordinates are sufficient to place markers on a map and therefore this extra information would be wasted.

2.2 Interactive visualisation

One of the experimental features of ATLAS was the use of alternative ways to view document text. In the standard document view, ATLAS divides the web page in half, with document text on one half and a map displayed in the other half. Alternative text views were explored as a way to maximise the use of such a confined space. Two alternative views have been implemented into

System name	**Display method**	**Gazetteer size**	**Disambiguation method**
GYPSY	3D polygon mesh	60,000+	None
Pre-NewsExplorer	SVG/Shape files/DMA	100,000+	Importance + local context
NewsExplorer	SVG/WorldKit/Google Earth	100,000-500,000	Scoring (with importance + local context + geographical distance as inputs)
Informedia	ESRI MapObjects	17,000-18,000	Local context
Perseus	Google Maps	1,000,000+	Local context + geographical distance
Web-a-Where	N/A	40,000+	Scoring (with local context + population as inputs)
G-Portal	N/A	60,000+	Local context + geographical distance

Table 2.1: System comparisons.

the ATLAS system, specifically the fisheye view and the column view.

2.2.1 Fisheye view

Research on the fisheye concept dates back to 1986 with Furnas' article on generalised fisheye views [Fur86]. Furnas discusses the fisheye concept in detail and demonstrates several areas where it is useful. One of the reasons for the fisheye research at the time of writing the article was the low resolution of computer monitors. Due to the limited amount that could be shown on a screen at one time it was useful to be able to maximise the use of that space. This was done with the intention of giving the user as much useful information as possible about their surrounding context while still allowing them to focus on an area of interest. Despite being originally designed for low resolution computer monitors, this research is still relevant today and has been experimented with as a useful tool to help present information in many different situations, for example: [SB92], [Fur99] and [Bed00].

As mentioned above, ATLAS uses fisheye to make the viewing of text more manageable within the confined space provided for them. For this reason our primary focus in this discussion will be on the use of fisheye for viewing text rather than using fisheye for other purposes. For details on the ATLAS implementation of fisheye see the fisheye Section in 3.4.3

FishNet

In 2004 Baudisch et al. presented FishNet, a fisheye web browser [BLH04]. The system was capable of displaying web pages using one of three different methods. The first display method was the standard web browser view that cuts off the bottom of pages that are too long. The second display method was proposed by Suh et al. in 2002 [SWRG02]. This method shows a "Popout Prism" that displays a miniature version of the entire web page on the left of the screen, with a square showing the area that the user is currently viewing. The third method, designed by the researchers, always shows web pages in

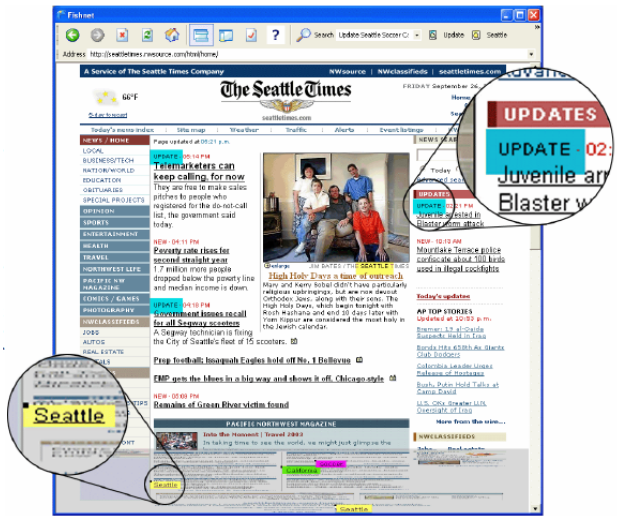


Figure 2.4: The fisheye view in the FishNet browser (from [BLH04]).

their entirety despite their size. Both the second and third methods use what they call "popouts" — not to be mistaken with the "Prism Popout" mentioned earlier — to show search terms that have been located in the context area of the page (the context area is either window on the left of the screen for the second method or the area at the top and bottom of the page where the fisheye effect is being applied for the third method). Figure 2.4 shows the fisheye display method of the FishNet system with examples of the "popouts".

FishNet treats the web page as if every part of it is of equal importance, meaning that when parts of the document are scrolled into the context area they are resized equally. A contrasting example is that of fisheye source code editor mentioned below. Code that is deemed more important than other parts of the code remains visible in the context view while other parts of the code are removed from visibility.

The researchers also performed a user test on the FishNet system with a goal of finding out how well each of the three different views perform in different scenarios. The performance of each of the display methods was measured in completion time and error rate.

Each participant in the user test was to complete several tasks on each of

the three interfaces. After completing the tasks on each interface the participants were asked to answer a series of questions about their satisfaction with the interface. After all of the tasks were complete they were given another set of questions.

In terms of task completion times, each of the different interfaces performed differently — relative to each other — in each of the four tasks. The fisheye view and the overview views performed better in tasks that required fast, full page searches for the existence of several highlighted terms. For tasks that required more complex analysis of terms, the fisheye and column views performed as well as or slightly worse than the standard view.

The results found that, overall, the fisheye view had the fastest average task completion time of all of the interfaces. But as is shown above, task completion time is task dependent. Also, satisfaction results did not match up to the performance results.

In terms of preferred interface, 10 of the 13 participants preferred the overview interface over both of the other interfaces, the remaining three participants preferred the fisheye interface. Six of the participants rated the fisheye view last out of the three choices. The researchers proposed doing a long term study to find out whether or not the users would feel differently about the fisheye interface if they were to use it more regularly.

Fisheye for source code - lab experiment

In 2006 Jakobsen and Hornbæk evaluated a fisheye source code editor they created [JH06]. In their design of the fisheye effect they needed to consider carefully what parts of the source code were important enough to be kept visible at the top and bottom of the screen (i.e., the area where the fisheye effect was being applied). Examples of code that was deemed important were: context defining code (e.g., loops and conditional statements), method declarations and class declarations (Figure 2.5 shows an example of code with the fisheye effect applied).

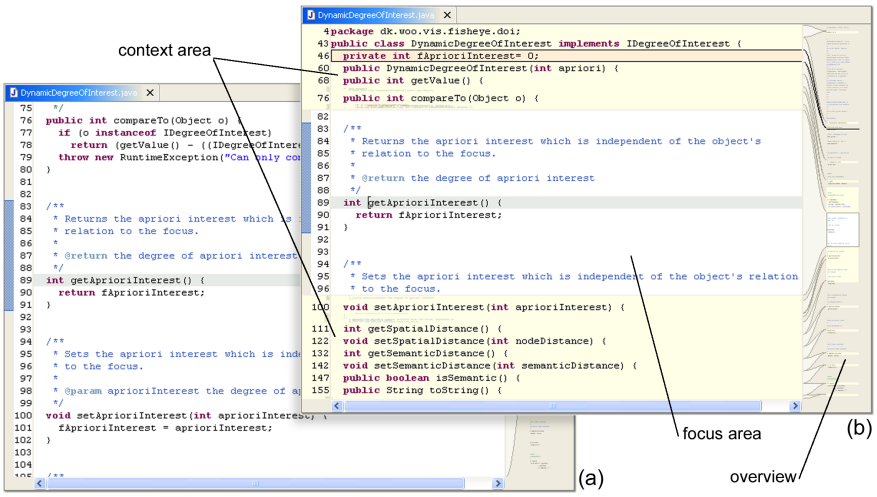


Figure 2.5: The fisheye source code editor designed by Jakobsen and Hornbæk (from [JH06]).

One of their main goals was to test the usability of their fisheye system as, to their knowledge, there had been no empirical user studies done at the level of interaction that their system was based. The user study was designed so that the researchers could assess the participant's task completion times, accuracy, interface preference and their satisfaction with the interfaces.

A total of 18 small tasks were performed by each participant, nine on a standard linear interface and nine on the researcher's fisheye interface. After each set of nine tasks were completed they were given a questionnaire with 14 questions to answer, each expecting a rating from one to seven.

Although results were mixed, overall the researchers discovered that users completed tasks significantly faster using the fisheye interface. No significant differences were found in the ability of participants to perform tasks correctly on either interface. In terms of satisfaction and preference, it was found that users liked the fisheye interface better than they liked the linear interface. The specific results are shown in Table 2.2.

Satisfaction question	**Linear**	**Fisheye**
How did you find the interface in general?		
Very poor - Very good	4.13(0.34)	**5.44**(0.20)
How was the interface to use?		
Terrible - Wonderful	4.00(0.29)	**5.13**(0.15)
Hard - Easy	5.19(0.37)	5.13(0.31)
Frustrating - Pleasant	3.81(0.41)	**5.00**(0.29)
Boring - Fun	3.56(0.29)	**5.25**(0.35)
Confusing - Clear	**5.81**(0.31)	4.50(0.37)
It was clear where I was in the source code.		
I disagree - I agree	5.88(0.31)	5.25(0.36)
I often lost my orientation in the source code.		
I disagree - I agree	2.88(0.43)	2.56(0.26)
How do you perceive these tasks?		
Very challenging - Very easy	5.31(0.27)	5.56(0.24)
How were your answers to the tasks?		
Very poor - Very good	5.56(0.26)	5.75(0.27)
Was the source code...		
Hard to understand - Easy to understand	4.81(0.31)	5.19(0.23)
Hard to overview - Easy to overview	4.44(0.38)	4.94(0.28)
Were methods you were trying to locate...		
Hard to locate - Easy to locate	3.50(0.39)	**5.31**(0.35)
Were other information...		
Hard to locate - Easy to locate	3.50(0.35)	**5.60**(0.22)

Table 2.2: Jakobsen and Hornbæk's [JH06] user test average satisfaction scores (and standard error of the mean). Significantly better scores are shown in bold.

Fisheye for source code - field experiment

In 2009 the same researchers, Jakobsen and Hornbæk, conducted research on their fisheye source code visualisation system in the field [JH09]. In the article they expressed their concerns about traditional lab testing of information visualisation systems. Three important points were raise: firstly, tasks in a lab tend to be easier than those faced in the real world. Secondly, one aspect of programs that is rarely tested in usability experiments is how well they integrate into a user's already existing set of tools. Finally, lab studies do not often extend past initial use of the of the program (as is discussed above in the FishNet section). Taking all of these factors into consideration, the researchers devised a long term field study.

They deployed the system to 10 professional programmers for them to use as part of their usual development tools for several weeks. Each programmer had between one and 20 years of programming experience, and of the 10 programmers eight had an IT background and two had a business-oriented background. The researchers originally aimed to study each participant for at least 10 working days. The actual period of study varied from two weeks to five weeks. Multiple methods of data collection were used including interviews, activity logging and probes.

The researchers discovered that participants used the fisheye view as often as they used any other core tools provided by the programming interface. Most of the programmers said that they would continue to use the fisheye interface even after the study had finished. One finding that the researchers found interesting was that the context view was most often used to locate occurrences of a method or variable and the information the researchers expected users to want to see proved to be less useful than expected. Overall however, the results of the experiment were promising for fisheye technology.

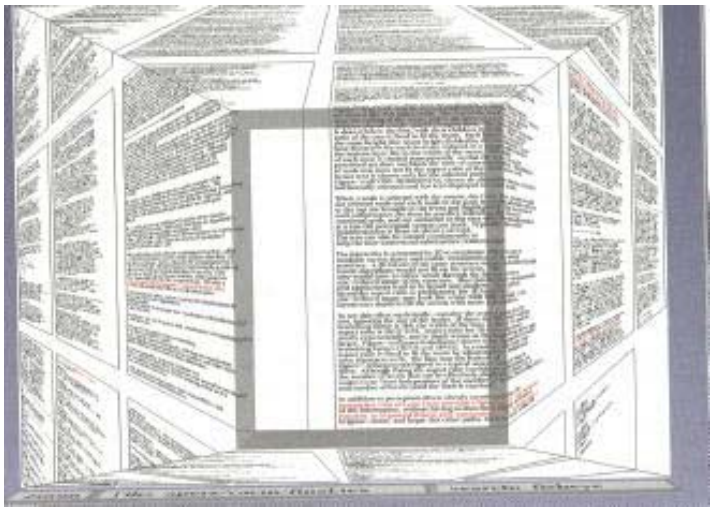


Figure 2.6: The Document Lens system (image taken from [RM93]).

2.2.2 Column view

In ATLAS the second alternative text view that is provided is the column view. The column view essentially allows the user to see the whole document at once by shrinking the text size and arranging it into several columns. The user can then magnify areas of interest by moving their mouse over the document. The magnification box follows the movement of the cursor so that it is easy for the user to quickly read the magnified area as they move the mouse.

The Document Lens

In 1993 Robertson and Mackinlay presented a document visualisation method they called the Document Lens [RM93]. Although not implemented in the same way as the system in ATLAS it is essentially similar. The goal of their system is to help users understand multi-page documents whose structure is unknown. This is done by arranging all of the pages into a grid and allowing the user to position a "lens" over them to focus on a particular point. As well as magnifying the area that the user is focused on, the system uses 3D graphics to distort the pages on the grid that are outside of the lens so that the user can see the context of the page (Figure 2.6 shows the system in action).

Comparisons between this system and the ATLAS column view system can be found in Section 3.4.3.

Text display research

A separate area of research, strongly related to the interactive visualisation of text, is how the layout of text on the screen affects a user's ability to read the text. In 2004 Dyson presented a paper discussing 20 years of empirical research in the field of text presentation [Dys04]. The research included discussion on variables such as line length, window size and line spacing and how these affect a user's ability to read the text effectively on a computer screen. The effectiveness of each layout was quantified by measures such as reading speed and reading comprehension.

One important area that was discussed in the paper was the use of multiple columns to display text. Research conducted in the area suggests that reading text from a document split into multiple columns is slower than reading text from a document that is displayed in a single column. Interestingly however, in two studies ([DK97] and [Gra93]) researchers found that participants felt that text was more organised and visually interesting when arranged into two or three columns.

Chapter 3

Design and implementation

In this chapter we discuss the design decisions and implementation details of ATLAS. Firstly we look at what a system like ATLAS requires in order to offer its functionality to users. Secondly we discuss the structure of the system and the decisions that were made during the earlier stages of development, especially decisions regarding what system layout to choose given the restrictions and benefits offered by each possibility. Thirdly we discuss the implementation of the system in detail, and lastly we discuss the design of the user interface.

3.1 Requirements and technology overview

In this section we outline the various technologies that make up ATLAS. Figure 3.1 shows a diagram of how the ATLAS system is structured. Most of the technologies shown are generalised as the specific implementations of these is not important; they can be interchanged and still provide a similar system. We will now briefly describe the purpose of each of these technologies.

- **Digital library:** The digital library provides the content that is to be enriched by ATLAS. The digital library also provides the majority of the user interface as ATLAS is designed to provide extra functionality to users of the digital library rather than be a complete system in itself.

- **Digital map:** The digital map is used to aid the enrichment of the

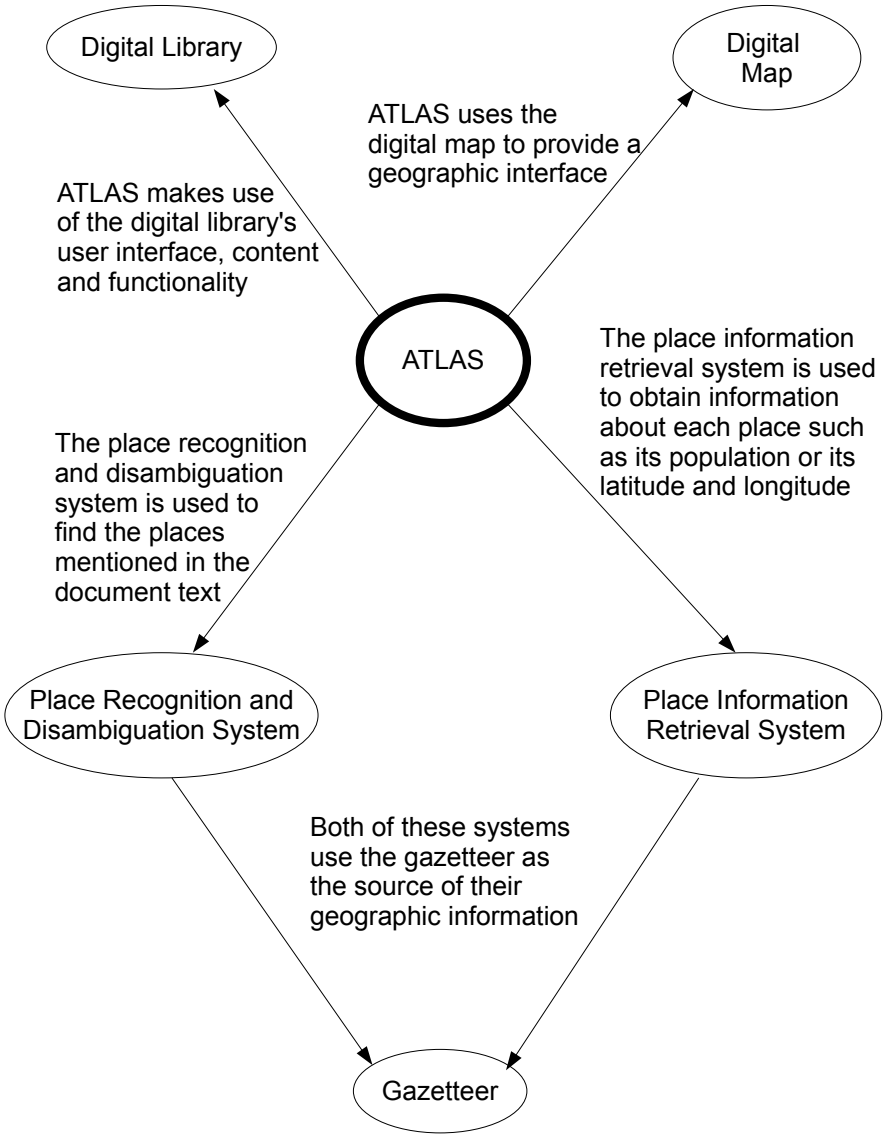


Figure 3.1: The structure of ATLAS.

digital library content by allowing the users to view the places located in the text on the map. The map also provides other functionality such as clickable markers that provide additional information about the given place.

- **Gazetteer:** A gazetteer is used to provide the information about the places of the world, such as their location and population.

- **Place recognition and disambiguation system:** This system is used for locating potential places names in the document text. It then disambiguates between place names that are ambiguous. It is built from the place names contained in the gazetteer and does not contain any additional information about each place. The place information retrieval system is used for this purpose instead.

- **Place information retrieval system:** This system is used to store the large amount of information present in the gazetteer in a way that can be easily and efficiently accessed.

In the following sections we discuss all of these technologies in further detail.

3.1.1 Digital library

The digital library that was chosen to be used for development was the Greenstone Digital Library. In particular the third (and latest) major version of the software, Greenstone 3, was used. Greenstone 3 was chosen because, unlike previous versions, it has been purposely designed to be an open and extensible digital library framework that makes use of modern web technologies.

The Greenstone Digital Library project can be traced back to the University of Waikato in New Zealand. The original system was written in Perl and used a CGI (Common Gateway Interface) web interface to allow the user to search and browse Greenstone collections. The MG (Managing Gigabytes)

text compression and indexing system is used as a back-end to the system, it allows very fast searching and document retrieval while still maintaining a good compression ratio.

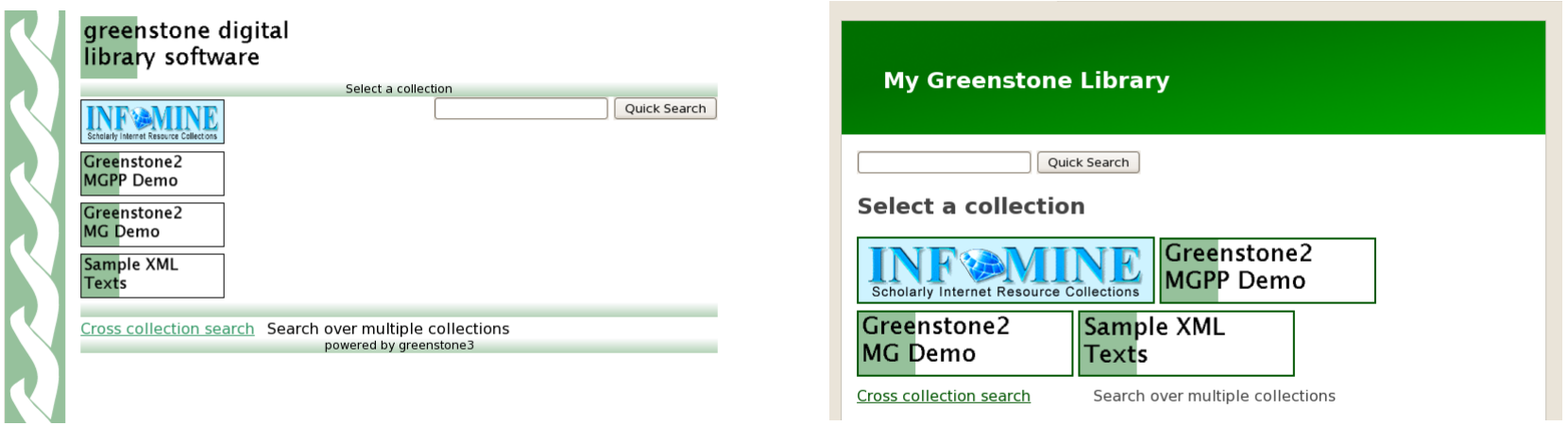
The second version of the Greenstone software was written in C++ and upgraded several features of the system to help meet the expectations of UNESCO (the United Nations Educational, Scientific and Cultural Organization) — whom the developers were in cooperation with at the time. The C++ version of the back-end of the system, called MGPP, allowed for more complex searches. The most recent version at the time of writing is Greenstone 3 which is programmed in Java and uses a combination of Java Servlet technology, XML (eXtensible Markup Language) and XSLT (eXtensible Stylesheet Language Transformations) to deliver the Greenstone interface to the end users. This most recent version is what ATLAS is built upon as it is a good match with GWT (Google Web Toolkit) development because GWT also makes use of Java Servlet technology. See Section 3.3.1 for more details about GWT development.

One of the features of Greenstone is the ability to change the interface's look and feel by changing the "skin" used by the system. For the development of the ATLAS system we chose to use the "dev" skin which is still in its early stages. The "dev" skin (shown in Figure 3.2(b)) was chosen over the standard Greenstone skin (shown in Figure 3.2(a)) due to it having several visual and functional improvements. As well being more aesthetically pleasing the skin has features such as better table of contents placement and better document hierarchy position indication.

3.1.2 Digital mapping system

As the digital mapping software Google Maps[1] was chosen to be used for development. There were several reasons why this mapping software was chosen over other alternatives, these include:

[1] http://maps.google.com



(a) (b)

Figure 3.2: A comparison between the (a) standard Greenstone skin and the (b) development skin used by ATLAS.

- GWT has a Java API for Google Maps which allows for easy integration with the rest of the web application.
- Included in the Google Maps API is the functionality to call Google's geocoder[2]. The geocoder is a powerful tool that takes an address (e.g. "Hamilton, Waikato, New Zealand") and attempts to return the latitude and longitude of that place.
- It has a well designed user interface with a large number of useful features.
- Google Maps is free to use for non-commercial purposes.
- Google Maps is an interface that many users will already be familiar with due to its wide spread use around the world.

It was decided that a mapping system like that presented in the GIPSY system would not be sufficient for the visualisation required in ATLAS. One of the main reasons is that there are certain situations where the information the 3D mesh view portrays becomes unclear. For example, if places around the outer edge of the mesh are mentioned frequently and places near the middle are mentioned comparably less, it would be difficult to rotate the view so that the height of the centre regions could be easily seen, due to the height of the surrounding areas. Also, because this view has only been designed to show the frequency of places mentioned within a small area it is unlikely that the visualisation method would be able to scale to the global size required by ATLAS.

The commercial mapping systems discussed in Chapter 2 were not considered for this role due to resource constraints. The rest of the non-commercial systems we considered inadequate for the purposes of ATLAS. The WorldKit system, for example, does not display geographic detail down to a low enough level. Although Google Earth provides many useful and powerful features for free it has the disadvantage that it must be run as a separate program.

[2]http://code.google.com/apis/maps/documentation/services.html#Geocoding

Map Server[3] was also considered as a potential candidate for this position as it is an open source project and would therefore allow full control over its features. It was decided however that Google Maps would be the best option due to its familiar and well designed interface and its large collection of pre-existing, powerful features. The fact that it is not open source was not an issue as the features provided in the Google Maps API are extensive enough for our needs.

3.1.3 Gazetteer

In ATLAS a gazetteer is used to help identify place names present in any given text as place names do not provide enough contextual clues to be recognised adequately without the use of a gazetteer [MMG99]. Also, simply being able to recognise place names in text is not enough; each place name also needs to be mapped to its real world coordinates so that it can be displayed on the map at the correct location.

The gazetteer being used for this purpose is the World Gazetteer,[4] which is the gazetteer used by the Web-a-Where system. Commercial gazetteers could not be considered due to resource constraints. The World Gazetteer is a gazetteer containing upwards of 300,000 entries and is available for free download from their website. Most entries contains an id number, the main place name for the place, a list of alternate names, a list of names written in the native language, the type of place it is (e.g., region, locality or country), a population estimate, latitude and longitude coordinates, the country it is part of, the region or state it is part of and the lower-level region it is part of. Table 3.1 shows some examples of gazetteer entries in this format.

[3] http://mapserver.org/
[4] http://www.world-gazetteer.com

ID	Place name	Alternative names	Native spelling	Place type	Sub-region	Region	Country	Population	Latitude	Longitude
188027528	Hamilton	Kirikiriroa	N/A	locality	N/A	Waikato	New Zealand	160862	-3778	17528
202101121	Sydney	N/A	N/A	locality	N/A	New South Wales	Australia	3654459	-3387	15121
-265	Abu Dhabi	Abu Zabi, Abu Zaby, Abū Ẓabī	أبو ظبى	emirate	N/A	N/A	United Arab Emirates	1975863	N/A	N/A
454362478	Las Vegas	N/A	N/A	locality	Clark	Nevada	United States of America	573014	3621	-11522

Table 3.1: Several example Gazetteer entries.

3.1.4 Place recognition and disambiguation system

To provide fast place name recognition a customised trie structure was used. More naïve approaches were also experimented with, but as the goal of ATLAS is to provide this additional geographic information in real-time, the speed of the structure is the most important factor to consider. For more details about these design decisions and the implementation details of this trie structure see Section 3.3.4.

3.1.5 Place information storage system

The system chosen to be used to store the place information was the PostgreSQL[5] database system. Earlier in development of the system a custom made tree-like structure was used for this purpose, but after some deliberation it was decided that there were several advantages to using a database that could not be ignored. These included: queries capabilities of greater complexity, less memory usage and the ability to easily modify or extend the dataset in the future as development continues.

The database system PostgreSQL was chosen primarily because PostGIS[6] was available as an plugin. PostGIS enables the database to perform spatial queries such as finding the distance between two points (e.g., How many kilometres separate New York city and California?) or finding the area of intersection between two shapes (e.g., What area of Africa was once part of the Roman Empire?). The feature of PostGIS that ATLAS makes extensive use of is its ability to find all of the points within a given area. This is also used to provide spatial document searching (finding documents mentioning places within a defined area) which is described further in Section 3.3.7.

PostgreSQL itself behaves much like other SQL databases and — if not for PostGIS — could easily be changed to another database system like MySQL[7].

[5] http://www.postgresql.org/
[6] http://postgis.refractions.net/
[7] http://www.mysql.com/

3.2 System structure

One of the most important decisions in the design of ATLAS was deciding how to best connect the various contributing technologies together. The first implementation of ATLAS was designed as a module for Greenstone[8], as conceptually ATLAS was to be an add-on to Greenstone's functionality rather than something that completely modified how the current system worked. To achieve this a new Greenstone service was created that would take the text of a Greenstone document, locate the place names in the text and return the text with the places names highlighted and tagged.

Greenstone 3 has six different types of services:

- **Query services:** These services take a query and a list of other parameters (such as the maximum amount of documents to return) and return a list of documents.
- **Browse services:** These services are used to provide classifier browsing. They take a classifier identifier with some structure parameter specifying what to retrieve and return the requested part of the classifier hierarchy.
- **Retrieve services:** These services take a document node identifier and can return the document's content, structure or some/all of its metadata.
- **Process services:** These services can perform tasks such as creating a new collection or importing a collection. They return either an error message or a status message.
- **Applet services:** These services process data needed for an applet. There is no set format for this type of request.
- **Enrich services:** These services take the text of documents, process them in some way (such as adding mark-up) and then return them.

As the program was designed to enrich the current content of the page it was programmed as an enrich service. To create this service a class was

[8]Details in http://www.greenstone.org/docs/greenstone3/manual.pdf pages 45–51

created that extended the Greenstone *ServiceRack* class in the Greenstone 3 service package. The *ServiceRack* class contains a process method which is given all of the documents that are to be processed by that particular service. A Java library containing the code for the both the place name recognition and the place name retrieval structures was created so that the service could have access to these data structures to allow it to find the place names in the text it was given.

This initial design worked by only adding the parts of ATLAS into Greenstone when they were needed (for example, when a document was viewed this triggered a map to add to the page with markers displaying the locations of any places found in the text). All other parts of the digital library functioned the same as they did before.

Each page of Greenstone is initially produced in XML and uses different XSLT files — based on what sort of page needs to be produced — to transform the XML into HTML. The document view XSLT file was specially modified so that, when a document was viewed, an HTML *table* was set up on the page with two columns. The regular Greenstone HTML was moved into the left hand column of the table and the ATLAS map and other features were placed in the right hand column. ATLAS acquired the text from the left hand column by locating the DOM (Document Object Model) node that corresponded to the document text. This text was then passed through the ATLAS place name recognition structure to acquire the place names from the text and to mark them on the map. At this early stage of development the place information was stored in a large tree-like structure and was loaded at the same time and from the same file as the place name recognition system. Figure 3.3 shows a diagram of this structure.

Overall this design proved to be ineffective as the two large data structures (the place name recognition system and the place information retrieval system) were being loaded from the same file using two separate processes, causing the whole 22MB file to be read in twice (which, on slower computers, could take

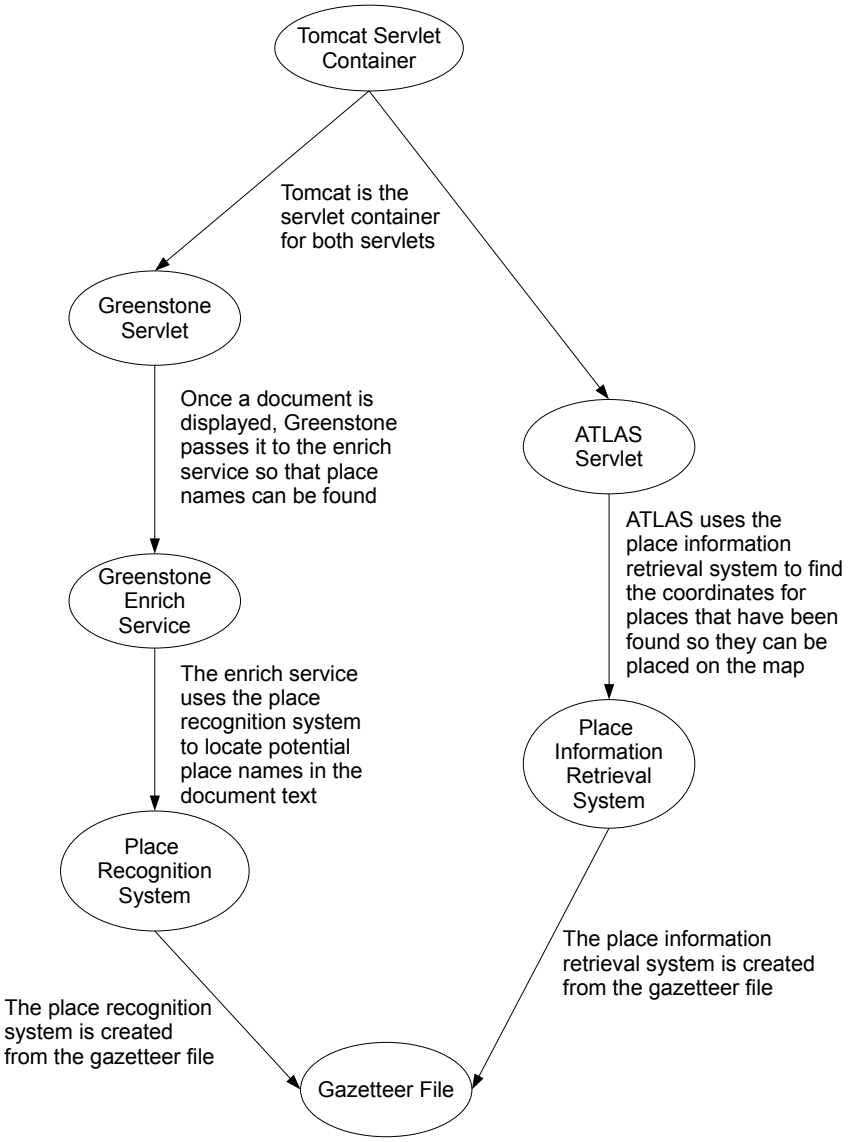


Figure 3.3: The original structure of the system.

several minutes to complete). Ideally both structures would have been loaded simultaneously on the same servlet and process but this proved to be difficult to achieve.

Due to a complication with GWT the two servlets could not communicate with each other via standard methods to share use of these data structures. When creating a servlet in Java, two of the methods that require implementation are the *doGet()* and *doPost()* methods. Whenever there is a *GET* or *POST* request made to the servlet, these requests are parsed by the corresponding method and a response is sent back. Standard servlets can normally communicate by sending XML messages to each other via the servlet container, the receiving servlet takes the XML request, parses it (in either the *doGet()* or *doPost()* methods) and then sends back a response. This standard communication is not easily possible in GWT as the easiest option of client-server communication uses what it calls GWT-RPC. GWT-RPC uses a *RemoteServiceServlet* class which extends the regular servlet class. Unfortunately the developers of the RemoteServiceServlet class have chosen to finalise the *doGet()* method of the servlet. This means that the *doGet()* method cannot be easily modified, which would be required for the Greenstone servlet and the ATLAS servlet to communicate. Other avenues of communication were experimented with such as using Java Sockets, but it was decided that a complete design change would be the best way of solving the problem.

Rather than having Greenstone effectively as the parent program, it was hypothesised that the opposite approach would provide better control over the system (Figure 3.4 shows this hypothesised structure). Instead of the system being mostly based on the existing Greenstone system and having parts of ATLAS only integrated into sections of it, a new system was created. This new system was in control of Greenstone and could load Greenstone pages into it. The system was then started by pointing the system to the ATLAS URL rather than Greenstone URL. This then loaded the start page of Greenstone, but did so in a way that allowed ATLAS to maintain control of the system.

It did this by changing all links and forms so that whenever they were clicked on they were first processed by the ATLAS system before the next page was loaded (discussed further in Section 3.3.2).

Having ATLAS in control allowed the system to know when the various relevant pages were displayed in the web browser. For example, when the home page of a document was loaded, the system knew to add a spatial search link to the links that were already present on the page. This was done so that spatial searching could be provided for that collection in a manner that was consistent with the other default Greenstone methods of searching. It also meant that functionality like the current text searching system could be augmented without much difficulty. One advantage of this design decision was that the large gazetteer file only needed to be read once instead of twice (as was the case in the first system design), which in turn greatly decreased the loading time for the data structures.

During later stages of development the place information tree structure was exchanged for a database that contained the place information (shown in Figure 3.5). This was done for several reasons which will we will now discuss.

The first reason was that using a database requires far less primary memory than the large tree structure (the tree structure used over 100MB of primary memory, whereas the database connection uses about 26KB). This is due to the fact that databases are stored in secondary storage and only require programs to store a database handle to access their information. Although using a database is fractionally slower than using the tree structure, it is worthwhile for the decrease in memory as the speed difference is not significantly noticeable given that the amount of work required to be done by the structure is relatively low (far out of proportion to the amount of memory it consumed).

The second reason to use a database was that, because the database is static in secondary storage (i.e., does not require to be constructed each time it is used), all of the information was already available once ATLAS was started and a connection was made. This effectively halved the time the system required

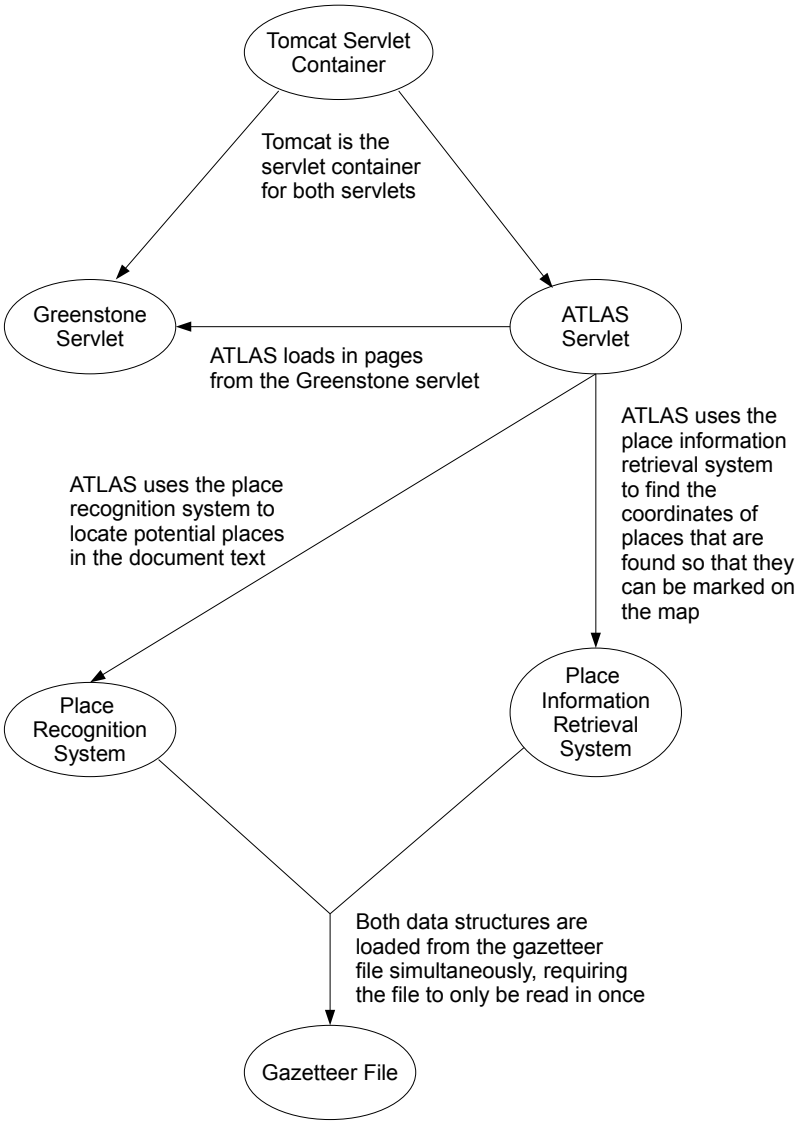


Figure 3.4: The second structure of the system.

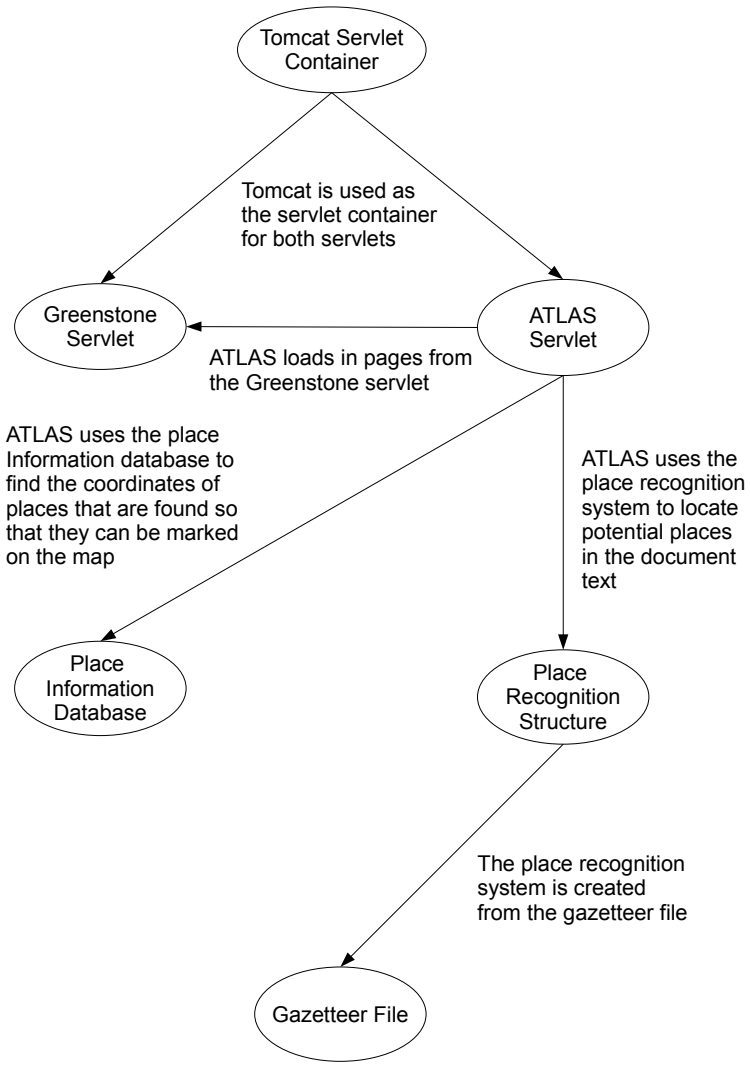


Figure 3.5: The final structure of the system.

to load.

Using a database also had the advantage that it allowed more complex queries to be performed such as finding out all the places above/below a given population or all places with their place type listed as "country" for example. The original tree structure could only find places given a place name and therefore was much more limited and would have been far less efficient in performing queries like this.

The final major advantage to using a database over the tree structure was that it allowed spatial queries to be performed using the latitude and longitude of the each place. This would have been difficult to implement efficiently in the previous structure. Being able to perform spatial queries opened up the ability to search for documents related to a given area of the world. This functionality is discussed in detail in Section 3.3.7.

Although using a database to store the place information meant that it was now possible to revert the system back to the original design of using a Greenstone enrich service, it was decided that it was better to keep the current design. Although conceptually it seemed like programming the system with Greenstone as the effective parent of ATLAS was the best choice, having ATLAS as the effective parent actually had more benefits that outweighed this argument. Having ATLAS as the parent of Greenstone allowed a finer degree of control that would not have been possible by only using the enrich service and XSLT modifications.

3.3 Implementation details

In this section we begin by discussing two technologies that contributed to the ATLAS system. This is followed by details of the page-loading system we implemented so that ATLAS could maintain control of the system at all times. We then discuss how we use the Google Maps API to provide the system's map functionality. This is followed by three sections discussing the implementation

of the place name recognition, place name disambiguation and place name retrieval systems respectively. Lastly we discuss our implementation of spatially searching for documents.

3.3.1 Contributing technologies

Here we discuss two technologies that are not requirements for the system to work but assisted in its implementation. GWT effectively provided the programming language that ATLAS was implemented in and GATE provided useful functionality for the place name recognition process.

Google Web Toolkit (GWT)

ATLAS is a client-server system programmed in Java for server-side coding and uses Javascript on the client side. However, the Javascript is not programmed in the traditional manner, it has instead been created with the aid of Google's Web Toolkit (GWT).[9] GWT is a system which essentially allows the developer to write a full web application in Java (both the client-side and the server-side) and then have GWT compile the client-side code into efficient, browser independent Javascript. With the numerous quirks of different internet browsers, being able to let GWT create code that works around these quirks is an attractive feature. Traditionally developers have had to try and work around these themselves which of course reduces productivity due to having to write more code and often having to do more debugging. GWT is also especially helpful when there are complex interactions between the client and the server, as it has its own RPC system that allows for simplified object transferal. ATLAS uses a large amount of client-server communication and therefore this feature is used extensively.

GWT simplifies client-side interface design by using "widgets" such as frames, panels, labels and text boxes in a way that is similar to Java's own GUI (Graphical User Interface) system: Swing. This allows an easy transition

[9] http://code.google.com/webtoolkit/

for Java programmers that have had some experience programming user interfaces in Java. These widgets are actually built from HTML elements such as *iframes*, *tables*, *divs* and *spans* but, for the most part, this is hidden from the developer. For this project however there were several situations where a lower level of interaction was required than what is provided by simply using the standard widget methods. For this low level of access each widget provided a *getElement()* method that returns the underlying DOM Element object. This allowed access to standard Javascript DOM element methods such as *getAttribute()*, *setInnerHTML()* and *appendChild()*. Why these methods were necessary is discussed in Section 3.3.2.

GATE and ANNIE

Unlike most of the systems discussed in Chapter 2 an additional step in the place name recognition phase that is taken by the ATLAS system is the use of GATE[10] (General Architecture for Text Engineering) to categorise words into their parts of speech (e.g., noun, verb, adjective etc.).

GATE is a collection of tools designed to perform various operations on text documents. One such tool is ANNIE (A Nearly New Information Extraction system) which is a powerful information extraction tool with modules capable of tasks such as tokenisation, sentence splitting and POS (part of speech) tagging.

ATLAS makes use of ANNIE's POS tagger when attempting to disambiguate between words that are places and words that are not places. The POS tagger gives each word a classification such as *proper noun*, *adjective* or *ad-verb*. With this information ATLAS is able to rule out any words that are not proper nouns for consideration as place names (such as "Many" or "Of"). Section 3.3.4 has more details about this process.

GATE uses a pipeline metaphor for structuring how documents are tagged and modified. Documents and corpora (collection of documents) are fed into

[10]http://gate.ac.uk

the pipeline and through each module attached to it in sequence. ANNIE comes with a default pipeline setting and this is used in ATLAS.

GATE is most commonly used via the GUI but, as the system is open source, there is also a Java API available to use. This gives the developer full access to all of the available features including ANNIE. To implement the pipeline in Java requires first informing GATE as to what its home directory is and where its plugins and are kept. Once this is done GATE can then initialise itself. An object is created to control the system, it is then given the ANNIE plug-in to load and the document to parse. After execution a series of annotations (some of which being the classifications of each of the words) are returned and can then be analysed by ATLAS.

3.3.2 Pseudo AJAX Loading

One of the major decisions in the design of ATLAS was to use a Web 2.0 style of interaction. This was necessary so that ATLAS could maintain control of the system, rather than giving control back to Greenstone (this is discussed further in Section 3.2). To achieve this it meant that the page required many dynamic DOM manipulations in order to transform the Greenstone Digital Library system — which is a CGI based system — into a system that updates pages using AJAX (Asynchronous Javascript and XML) rather than full page reloads.

The traditional web development model requires each request that is made to the server to be returned with new web page as a response. AJAX on the other hand allows web developers to make requests to the server without the need for a full page reload. This is done by making an asynchronous call to the server which then responds with the requested information in XML form. This information can then be used to modify the current web page rather than requiring a new page to be created.

Figure 3.6 shows the basic cycle that takes place when a new page is loaded. To achieve this dynamic loading, ATLAS modifies each page so that when

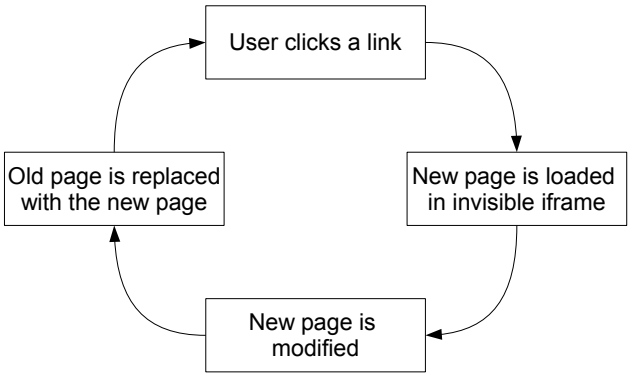


Figure 3.6: The page loading cycle.

a link is clicked the new page is loaded into an invisible *iframe* (simply a regular iframe with it's height, width and border width set to zero). The page is loaded into this *iframe* rather than the browser through the use of a Javascript function attached to the *onClick* handler of each link on the page. This Javascript function sets the *src* attribute of the *iframe* to be the URL of the new page, causing the browser to load the new page in this frame in the background. The click handler function attached to the link returns the boolean value *false* which the browser understands to mean that it should not treat this link as it normally would and does not load the new page into the main window.

In order to preserve this AJAX behaviour throughout the application it is necessary to modify every internally linking hyperlink (i.e., pages that link to other Greenstone pages) in each page before it is displayed. This is because clicking on an unmodified link will take the user out of ATLAS and into a regular Greenstone environment as the link will be treated as a regular link. Anchor tags are modified by adding an *onClick* attribute to the tag that calls the ATLAS *loadPageFromUrlJS()* method, taking the new page URL as a parameter. Forms are modified in a similar way, the action attribute of the form is changed to call the *loadPageFromForm()* method, taking the form's DOM element as a parameter.

To set up these Javascript methods it requires GWT's Javascript Native

Interface (JSNI) which is the lowest level of interaction that GWT provides. It allows the developer to write actual Javascript code — rather than Java code — that will remain unchanged when the code is compiled. For the Javascript methods called to create the AJAX-like behaviour in ATLAS there are added complications. These Javascript methods are required to call Java methods that have been compiled by GWT in Javascript. For instance *loadPageFromUrlJS()* calls the GWT-compiled method *loadPageFromUrl()* which contains the code that actually does the page modifying and loading.

Normally GWT-compiled methods cannot be accessed from outside the GWT code (for example, you cannot call a method compiled by GWT from a separate Javascript file by calling function names directly, which is normally possible with Javascript). To get around this, a special syntax (shown on lines 5 and 13 of Figure 3.7) is used in JSNI to refer to methods compiled in GWT. GWT Java method names compile to different Javascript names depending on what level of obfuscation is being used, so a uniform way of calling these methods is important.

Finally, in order to make functions callable from Javascript the JNSI methods that call the GWT-compiled methods need to be attached to the global Window object. This is achieved by simply calling $wnd.[FUNCTION NAME] = [FUNCTION TO ATTACH]. $wnd is the object through which the Javascript *Window* object can be accessed in JNSI (similarly $doc refers to the global Javascript *Document* object).

Once the page is finished being modified, parts of it are moved from the invisible frame onto the actual page. There are three main *divs* in a Greenstone document: the header, content and footer *divs*. These *divs* are located in the newly loaded page in the invisible frame and used to replace the corresponding section in the main frame (using several *replaceChild()* DOM method calls). Header tags such as stylesheet and Javascript references also need to be updated for each page to make sure they are displayed correctly. This involves removing stylesheet and Javascript references that are only relevant to

```
1  public static native void setUpPageFromUrl(GS3MapLibarary ml)
2  /*-{
3    $wnd.loadPageFromUrlJS = function(url)
4    {
5      m1.@org.greenstone.client.GS3MapLibrary::loadPageFromUrl(Ljava/lang/String;) (url);
6    };
7  }-*/;
8
9  public static native void setUpSpatialSearchPage(GS3MapLibarary ml)
10 /*-{
11   $wnd.loadSpatialSearchPageJS = function()
12   {
13     m1.@org.greenstone.client.GS3MapLibrary::loadSpatialSearchPage() ();
14  };
15 }-*/;
```

Figure 3.7: Two examples of the syntax used to refer to GWT-compiled methods in JNSI.

the previous page and adding the references from the new page. The actual URL in the new header tags need to be modified as well. This is because most of the references are relative links and ATLAS uses a different servlet than the Greenstone servlet. This means a URL like *dev* needs to modified to be something like *http://localhost:8080/greenstone3/dev*. This is also necessary for images as most of these also link relatively rather than absolutely. Any externally linking references are left unchanged however.

One of the most commonly used features of Greenstone is its full-text search, due to both it's speed and accuracy. Using this fact it was decided that it would be useful to be able to extend this functionality with ATLAS. Each of the returned documents were opened in a temporary invisible iframe — in much the same manner as links are loaded. The text in each document was then searched for place names, and the places found were then marked on the map in a different colour for each document (as shown in Figure 1.4). The invisible frames were then deleted from the page.

As a side point, it is important to note that while waiting for an operation to complete — such as waiting for the server to load the place name recognition structure in this case — timers should be used to check whether the operation is complete at regular intervals rather than using a loop. This is because a warning is likely to be displayed telling the user that the Javascript on the

```
1  <head>
2    <meta http-equiv="content-type" content="text/html;
3      charset=UTF-8">
4    <script src="http://maps.google.com/maps?file=api&amp;
5      v=2&amp;sensor=false&amp;key=ABQIAAAAtgBCR-EMOIo..."
6      type="text/javascript"></script>
7
8    ...
9
10 </head>
```

Figure 3.8: The *localhost* Google Maps key being inserted into ATLAS.

page has become unresponsive if control is not returned to the browser within a certain time period. Timers avoid this problem and minimise CPU usage with the only drawback being a small delay once the task is complete for the waiting tasks to continue.

3.3.3 Google Maps

Google Maps was chosen to provide the map interface for ATLAS. Because Google Maps has a GWT Java API it made it much simpler to incorporate into ATLAS which was already using GWT. In order to use the API a key must be acquired from Google that gives access to its services. To get the key Google must be informed of the host name of the website using the map. For development purposes *localhost* is a good choice as most developers will want to be testing their programs on their own machines before they are deployed to the web. Once development is complete however a new key will need to be acquired. Once a key is acquired it is then inserted into a *script* tag in the main HTML page of the system as can be seen in lines 4, 5 and 6 of Figure 3.8.

The API provided by Google allows a considerable amount of control over the map that is shown to the users. Standard features are available such as adding movement and zoom controls and allowing the user to choose between

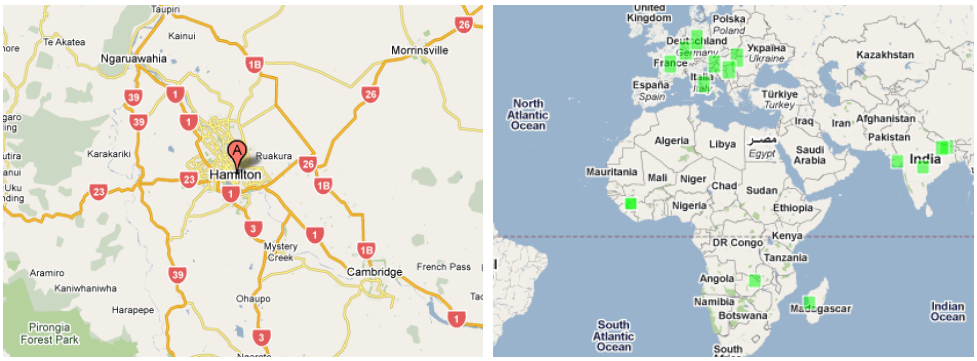
different map types such as satellite view and street map view. More advanced features are also available such as adding markers or other shapes to the map and using the Google Geocoder to locate a place given an address.

Adding controls such as the movement and zoom controls or the control to choose different map types is a simple task in the Java API, as these are all inherited from the *Control* class which is the abstract base class of all of the map controls. The map class has an *addControl()* method that takes a *Control* object and adds it to the map interface in the given position. Defining custom controls is also possible and this is used within ATLAS to provide controls for the spatial searching which is discussed further in Section 3.3.7.

One of the other features that is also used extensively in ATLAS is the use of markers. Standard Google markers (like the one shown in Figure 3.9(a)) were originally used to mark the places that were found in the text. It was later discovered however that these markers were not sufficient for our requirements as the default marker does not allow much customisation. For example, in the earlier stages of development we desired to be able to change the colour of the markers to represent how well places scored. Although it is possible to change the image that is used for the marker, it is not possible to change the colour of an existing marker.

The lack of customisation meant an alternative had to be considered. It was decided that polygons — which are another type of map overlay — would be the best replacement because they allowed much greater control over aspects such as size, colour and transparency.

A polygon is created in the Java API by defining three or more points (ATLAS uses a rectangle made from four points) on the map in *LatLng* format, which is the map API's way of storing latitude and longitude information together. The polygon is also given a fill colour and a border colour as well as a floating point number which defines its level of transparency. Changing the size of the polygon was important because the polygon needed to be visible at different levels of zoom. To achieve this the polygon had to be bigger in relation



(a) (b)

Figure 3.9: Two different types of map marker: (a) A standard Google Maps marker and (b) customised polygon markers.

to the map when zoomed out so that it was still easily visible, and smaller in relation to the map when zoomed in so that it did not cover the entire view, which would make it unclear where the polygon was actually centred. Figure 3.9(b) shows an example of the polygons used in ATLAS.

Being able to change the colour was also important for several reasons. In the earlier stages of development colour was used to distinguish between places that were given a high score and places that were given a low score (higher scores meaning that a marker was more likely to be the actual place discussed in the text). The colour of the polygon marker was interpolated between the two extremes — in this case red and green — based on where its score fell on the scale. As the design of the system was further refined, however, there was no longer a need to distinguish between places with high scores and low scores.

In the initial design of ATLAS, places with the same name were all marked on the map at the same time (i.e., if "Hamilton" was mentioned in the text then all of the Hamiltons contained in the gazetteer were marked on the map at the same time), hence needing colour coding to help the user see which places scored better out of all the possibilities. Gale et al. [GCY92] found that there is a high probability (98%) that a polysemous word in a well-written discourse is likely to have the same sense thoughout the discourse. With this assumption it was decided that only the highest scoring places of each name should be marked on the map (i.e., only the highest scoring Hamilton would

be marked on the map). This is why, in the standard map view of the current version of ATLAS, the rectangle markers are all one colour unless they have been highlighted.

The other area where changing colour is important is when showing places from multiple documents. Colour coding places to match their documents allows the user to easily see which places belong to which documents. This is used in ATLAS-enhanced version of the text search feature of Greenstone. Once the results from the search are retrieved, ATLAS searches for places in each of the documents that are returned. The places found in each document are then marked on the map, an algorithm is used to make sure that each document's places have a colour that is distinct from each of the other colours.

User interface objects like polygons, lines and markers can also be given mouse events such as mouse click events or mouse over events. A mouse click event is used in ATLAS to bring up extra information when the user clicks on one of the place-marking rectangles. A mouse click handler is created in the Java API in a similar manner as most Java user interface systems. User interface objects have an *addClickHandler()* method that takes a user-defined click handler as an argument and this is called whenever the object is clicked on. It is important to note that these user interface objects are clickable by default, which can cause problems if they have no click handler assigned to them. To make them unclickable, a corresponding options object (for example markers have a *MarkerOptions* object) must be created and passed to the constructor when the corresponding user interface object is created. The *setClickable(false)* method should be called on the options objects before they are given to the constructor to disable their ability to be clicked on.

As ATLAS is designed for heavy duty use (large documents with many places) it takes a different approaches to the most of the systems discussed in Chapter 2. The main difference is that ATLAS does not place any place names next to markers — unlike several of the systems designed to show fewer places at once — as in many cases where there are several markers crowded

in one area it would be more detrimental than helpful. Places are not difficult to find however due to features allowing the user to centre the map on a place or highlight it.

As mentioned previously, the Google Maps API allows access to the Google Geocoder which ATLAS makes moderate use of in certain situations. Countries and regions do not contain latitude and longitude coordinates in the gazetteer, so the Google Geocoder is used to attempt to locate these places when these pieces of data are missing. An asynchronous call is made to the Google Geocoder system with the place in a format like "Hamilton, Waikato, New Zealand", this which will either return successfully with a *LatLng* point defining where the place is or it will return unsuccessfully if the place cannot be found.

3.3.4 Place name recognition

The first experimental approach to locating place names was to create a place data structure to store all of the information about each entry together in an organised manner. Each of the entries in the gazetteer file were then read into the program and all of the entries were stored in a Java *HashMap* structure. The text to search was then divided into separate tokens based on whitespace. The hash table was then searched to see what tokens matched gazetteer entries. It became clear very quickly that this approach was too slow to find places in large amounts of texts in real-time as it sometimes took minutes to complete.

The second experimental approach was to use a trie structure. A typical trie structure for English text contains 26 children at each node, one child node for each letter of the English alphabet (Figure 3.10 shows an example of this basic trie). This allows most standard English words to be checked in linear time relative to how many characters are in the word. This basic trie does not allow for upper case letters, punctuation characters, numbers or other miscellaneous characters however, so one of several methods can be employed to allow for these. The naïve method is to simply have as many

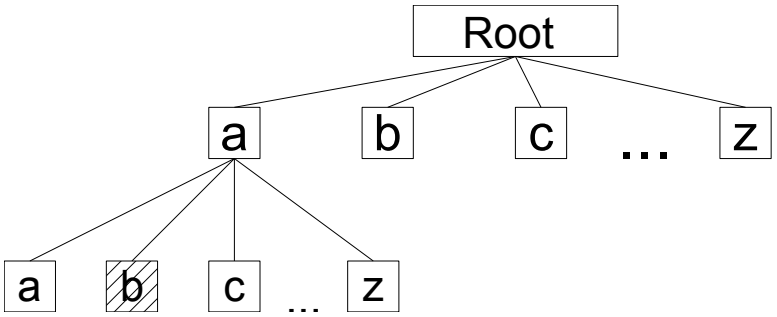


Figure 3.10: A basic trie where "ab" is match and all other nodes are not matches.

children at each node as there is possible characters, so for example if the words use only ASCII characters each node would have around 128 child nodes. In typical English there are many more lower case letters than there is other types of characters, so this means that the majority of the 128 possible ASCII characters (i.e. the non-lower case letters) at each node are likely to be empty. For small trie structures that require frequent access this method is the best option. This is because, even though the structure is sparse and uses a large amount of memory to store a small amount of data, it uses a minimal amount of comparisons to test strings.

Using an extra child node for each of the possible different characters is very inefficient in the gazetteer used in ATLAS. The difficulty lies in the Gazetteer being in Unicode, which has a little over 100,000 characters to allow for compared to ASCII's 128 characters. An alternative to using a large amount of child nodes is to group child nodes into smaller sets and use a different data structure to manage the sets. In ASCII for example, 26 nodes could be used for letters (disregarding case), one node for numbers and one node for other characters. This greatly reduces the sparseness of the trie — which also greatly reduces the memory needed — while still maintaining a search time that is linear in most cases and only a little more than linear in worst case scenarios. Table 3.2 shows a overview of the different characters in the Gazetteer.

The original trie node layout experimented with using 26 nodes for English letters, and one node for all other characters. This experiment produced

Uppercase English Letters	473,342
Lowercase English Letters	2,453,918
Total English Letters	2,927,260
Numbers	10,693
Arabic Letters	66,289
Asian Characters	3,182

Table 3.2: Character counts in the gazetteer.

excellent results despite its simplicity, it did however highlight one area that could be improved. Upon analysis of this trie structure it was discovered that even with this compaction of the child nodes there was not a significant loss of efficiency that was originally hypothesised. The node with the most non-English letter characters was the root node with 858 child nodes compacted into one node, the next largest node was significantly less with 42 child nodes compacted into one node. This led to redesigning the top level node to be a full sized node with no compaction and leaving the rest of the nodes unchanged. This produced a small increase in performance with a comparatively small amount of extra memory usage. For more details see Section 4.3.

When a word is matched in the gazetteer trie structure it is not automatically considered to be a place. ATLAS uses a variety of methods to disambiguate between words that are places and words that are not places. These include using the ANNIE system within GATE to separate each word into its appropriate classification such as verb or noun, using a list of common nouns and using a list of common names of people (these are similar to the stop-word lists used by several systems discussed in Chapter 2. At this stage in development ATLAS is primarily focused on place name recognition in English and therefore can afford to use linguistic rules to help disambiguate between words that are places and words that are not places.

The first and most obvious requirement for a word to be considered as a place name (at least in English and other similar languages) is the capitalisation of its first letter. This is achieved easily by simply discarding words that do not meet this requirement. The second requirement of a place name to be

considered is that is must be a noun. The ANNIE system within GATE has a built in part-of-speech tagger that can be used to classify each word in a document into its appropriate category. Doing this is necessary because there are place names, such as Many in France and Data in India, that simply using the gazetteer will find but are discarded when categorised. However if Many is marked as a noun by the ANNIE system it will be considered as a place name. The third difficulty is there are many place names that are also common words within the English language or names of people. Examples of common words that match in the gazetteer are “Bank” and “Data” which are located in India or “Of” which is located in Turkey. Examples of person names that match in the gazetteer are “Victoria” and “Washington” which both match many places in the world.

3.3.5 Place name disambiguation

With the extensive gazetteer used by ATLAS, place name disambiguation is an important task. Some place names map to many different places in the gazetteer such as San Antonio (49), San Francisco (30) and Washington (25). When places like this are found in the text it is necessary to work out which of the many possible options that the place name actually refers to.

Place name disambiguation uses the contents of the rest of the document (and sometimes outside data, such as what city or country the document is from) to decide what places best fit the place names that are mentioned in the text. The method that was chosen to be implemented for place name disambiguation is designed to make the most of the knowledge gained by using such an extensive gazetteer.

The disambiguation system designed for ATLAS is similar to that of the Web-a-Where system’s document locality finding procedure. It is score-based and uses the frequency of place names combined with their relationships to calculate the most likely matching place for a given place name.

Each time a place name is located in the text a certain amount is added

to its "score" — although 256 is the amount a score is increased in the implementation, in practice varying this number does not make a difference. The potential parent places (i.e., regions or countries that the place resides in) of each place that is found in the text are also given a percentage of that score (Figure 3.11 outlines the exact scoring procedure).

Once all the place names in the document have been found and scored several more steps are performed (These are shown in Figure 3.12). Firstly, as found in Section 4.2 it is important to reduce the scores of all of the places that are not explicitly mentioned in the document. This is done so that their influence on the final scores is minimised. As an example, if "Cambridge" is mentioned in the text then in the initial scoring stage both New Zealand and England will get some of the score that is given to Cambridge each time it is mentioned — as at this stage both New Zealand and England are potential parents of Cambridge. If "New Zealand" is mentioned somewhere else and England is not, then it is more likely that the text is actually referring to Cambridge, New Zealand. If Cambridge is mentioned several times in the text then England's score has the potential to get relatively large. At the next stage of scoring (discussed below), if England has a high score then it has the potential to skew the disambiguation of other places that could possibly be in England. This is why a place that is not directly referenced like this has its score reduced in this first stage of scoring.

The next stage of score manipulation is to add part of the highest scoring place's scores to their children. The definition of how close a place has to be to the top scoring place before being considered high scoring can be varied with a parameter. Having this parameter too small (which causes there to be less high scoring places) is bad if a document is focused on more than one place, such as a news article discussing Afghanistan which could mention places from both Afghanistan and the United States. Having this parameter too small in this case could cause places from the Afghanistan having their scores increased but places from the United States having their scores remain the same. This could

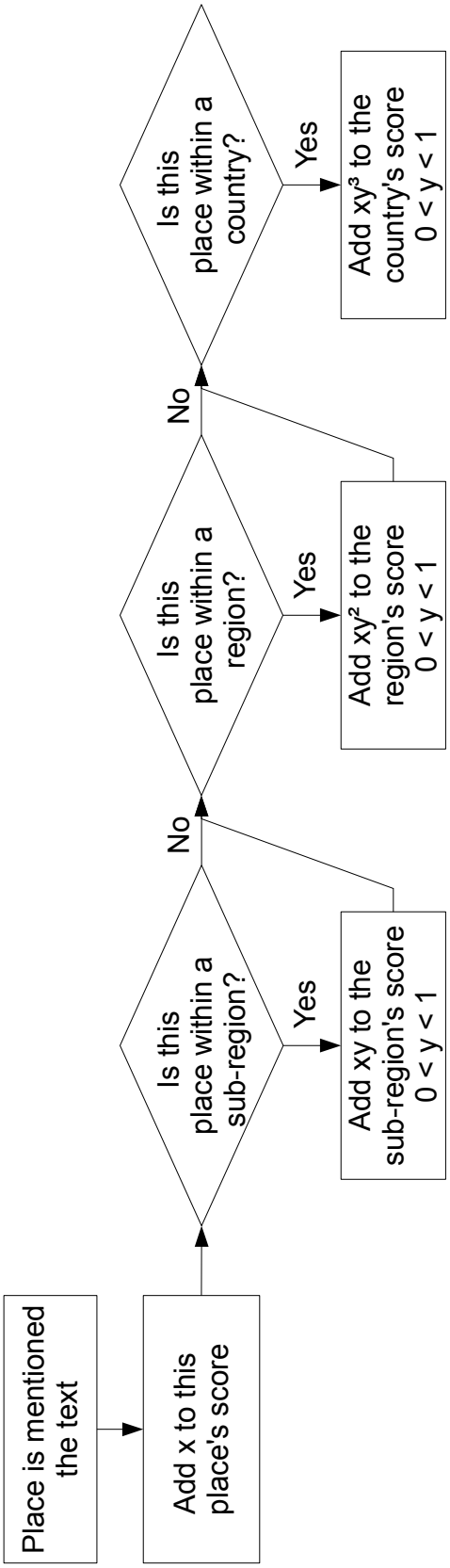


Figure 3.11: The per-place scoring procedure.

possibly cause these places to be scored lower than other places with the same name in a different country. Having this parameter too high (which causes there to be more high scoring places) can also potentially cause inaccurate disambiguations. An example of this is when disambiguating articles from a source like Wikipedia.[11] Although the article may be focused on a particular place it is likely that it makes mention of several other countries at some point in the article. Allowing these other countries to also add part of their score to their child places could skew the disambiguation towards these countries which is likely to be undesirable.

Several of the systems discussed in Chapter 2 used a geographical distance calculation to help the disambiguation process. Adding a geographical distance heuristic to ATLAS was considered as an additional step in the disambiguation scoring system, but it was decided that it was not a high enough priority at this current stage in development. Although place name disambiguation accuracy is important it has not been the primary focus of the project. It is likely however that in future development a distance heuristic will be implemented into the system as geographical distance between places is likely to be an important resource for place name disambiguation.

All of the parameters mentioned above (e.g., how much of a places score is removed when it is not directly mentioned or how much of a parent's score is added to its children) are modifiable. How specific values of these parameters effect the system is discussed and evaluated later in Section 4.2.

3.3.6 Place information retrieval

Once the places in the text were located it was then necessary to be able to find out information about places by that name and to find about places related to those places (such as the country a place resides in). As the information was effectively organised as a tree (city → region → country) it was initially decided that this would be the best way for it to be structured. So in the initial

[11] http://www.wikipedia.org

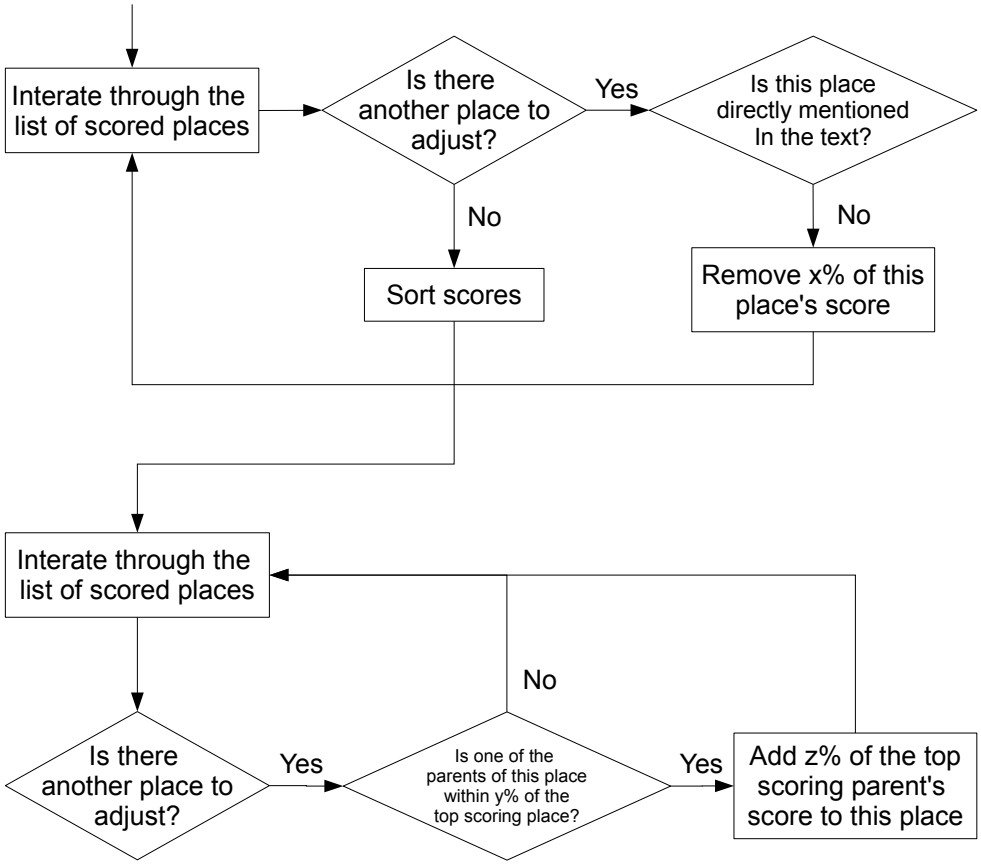


Figure 3.12: The final scoring adjustments.

stages of development a large tree-like structure was used to store the place information. Child places were children of their respective parent places in the tree (e.g., the state of New York is a child of the United States of America and New York City is a child of New York state). The rest of the information about the places (e.g., population, latitude and longitude) was stored in the nodes of the tree.

This tree structure was additionally accessible through a Java *HashMap* structure to allow efficient access to individual nodes themselves by using the place names as the keys. Although this structure was efficient for the operations it was needed for at the time, it used a large amount of memory and required a long time to be created upon initialisation of the program. Another disadvantage of this structure was that it was not efficient when handling a

place that was referred to by an alternative name (e.g., Myanmar being referred to as Burma). A separate *HashMap* was used to map alternative place names to their more commonly known names. This could then be used to locate the actual place in the original structure. Although this was adequate for locating most places, there was one situation where it was inadequate. This was if an alternative place name mapped to a place name that also mapped to more than one place. For example, "Kirikiriroa" is an alternative name for Hamilton in New Zealand, if "Kirikiriroa" was found in the text then the hash table would say that "Hamilton" is the more common name of the place, but because there are many Hamiltons in the world it would not be clear which Hamilton "Kirikiriroa" referred to.

Although ATLAS does store the alternative place name spellings and will find place names with these spellings, at this point in development the focus for ATLAS is not on multilingual place name recognition. Further research into multilingual capabilities is likely to be done as part of future developments to the system.

Later in development it was decided that a database would be a better alternative to this tree structure as it required less physical memory and there was not a significant difference in performance in most situations. The main place information was stored in one table with the alternative name information stored in a separate table, but connected to the original table through the use of foreign keys. Alternative place names were mapped to the identification numbers of the place they referred to rather than to the actual name itself. This solved the problem that was present in the tree structure because there was a one-to-one (rather than one-to-many) mapping between alternative place names and the actual place information. The full database table layout is shown in Figure 3.13.

Using a database significantly decreased the startup time of the program as it is only required required to connect to the database rather than having to create a large structure from scratch. It also allowed more complex queries

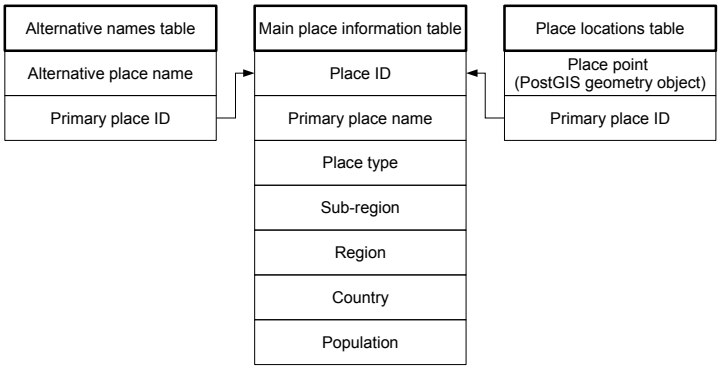


Figure 3.13: The table layout of the PostgreSQL/PostGIS database.

such as finding places above a certain population or finding all the places within a certain country.

The database was also PostGIS-enabled which allowed spatial queries to be performed to locate places within a given area. The data used by PostGIS was stored in another table and again mapped back to the main place information via foreign keys. To fill this table it was required that the latitudes and longitudes in the main place information be converted from their original format into a format that could be understood by PostGIS. PostGIS uses a standard floating point number format to store its coordinates whereas the gazetteer format does not use a decimal point and therefore the decimal point's location must be inferred from the number of characters in the number (for example 12345 actually means 123.45 ° and 6789 actually means 67.89 °).

This combination of trie structure and database gives ATLAS the ability to both find places in long texts quickly, and find information about those places — even when the place is referred to by an alternative place name — efficiently, while still maintaining an acceptable level of memory usage.

3.3.7 Spatial searching

In addition to the standard searching methods of Greenstone, ATLAS allows users to find documents by places mentioned in their content — otherwise known as spatial searching. To allow access to spatial searching ATLAS places

an additional link on the "about" page (home page) of each collection that links to the spatial searching page (as shown in Figure 1.2). This is done so that the link is then located amongst other related links such as the browse and text search links.

A map is used as the majority of the user interface for defining a spatial search. A visual map was used rather than some form of coordinate input system as it has been suggested by Furnas that such interfaces limit the user's ability to cognitively construct spatial queries [Fur91].

In 1996 Larson presented and article on geographic information retrieval and spatial browsing [Lar96]. The most interesting part of this paper for ATLAS development is the discussion of different types of spatial queries. Five different spatial query methods are discussed, specifically: point, region, buffer zone, path and multimedia queries. Of these 5 query types we will discuss 3 of them. Path queries and multimedia queries are irrelevant to the type of spatial searching proposed for ATLAS.

Point queries are the most basic form of spatial query. Point queries allow a user to click on a single point on a map to find information about that point, places surrounding that point or places containing that point. This type of query is not implemented in ATLAS. It is probable however that a variation of this sort of query will be implemented in future development. As discussed in 2.1.4, allowing the user to construct a spatial query by clicking on countries to select them is both intuitive and efficient.

Region queries are the type of spatial query that ATLAS allows users to construct. Users can select a region of a map to search for information about places — or in this case, documents relating to places — within that area. This method is both simple and intuitive for users.

Buffer zone queries are essentially region queries that are constructed in a different manner. They effectively allow a user to ask "What places are within X distance of Y?", for example a user might want to know "What places are within 200km of Paris, France?" which would create a region query 200km

around Paris. Again, this form of spatial searching is not implemented in ATLAS as situations where it would be useful are limited.

When the page is loaded a blank map is shown. Each time the user clicks a point on the map a marker is placed at that point, if there is more than one point then it is joined to the point preceding it (Figure 1.8 shows and example of this). Using this method the user can create a polygon to define the area they wish to search for related documents. When the search is executed the final point will be connected to the original point, creating a closed polygon that is used in the database query.

PostGIS is used to provide the spatial searching capabilities of ATLAS. It allows the creation of tables containing geometric objects (points in this case) which can then be queried in a variety of ways. Functions are available that can perform tasks such as measuring the distance between two points, measuring the area of a polygon and finding all of the points within a given area.

To actually match documents to the area given by the user two different methods were considered. The first method involved knowing in advance what places are mentioned in every document in the Greenstone collection. These could then easily be matched with the places found within the given area, resulting in very quick and efficient spatial searching (Figure 3.14 outlines this procedure). The difficulty with this approach however was the amount of preparation necessary to provide this efficient searching. It was considered whether this information could be retrieved and stored when a Greenstone collection is built. Although this would increase the build time of the collection it has the benefit of only needing to be done once. If a user knows in advance that they would want a collection to be spatially searchable it would be worth the extra build time for fast spatial searching. Obviously this preparation could also be done after the collection is built but it is more intuitive to do it while the collection is being built.

Although this preparation method would work well in situations where the

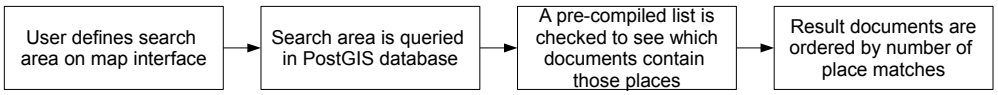


Figure 3.14: The first spatial search method.

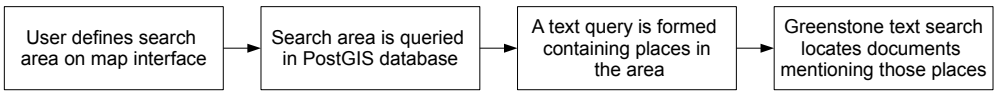


Figure 3.15: The second spatial search method.

user knows they will be wanting to make use of spatial searching extensively, it would be an unnecessary amount of work to have to spend hours preparing large, pre-existing collections if the user will rarely need the spatial searching functionality. For this reason we considered another method to link documents to locations.

The second method of matching documents and locations involved making use of the powerful text searching capabilities provided by digital libraries. Instead of having prior knowledge as to which places map to which documents, we instead treated every place within the user's query area as equally likely to map to a document. We created a large query string made up of all the place names of the places within the given area to find the places within the user's query area and used this query string to search the collection of documents. The results were then returned as with any other text query. This method worked on the assumption that documents containing more of these words in greater frequencies would be higher ranked — and therefore more likely to be relevant to what the user is searching for — than documents where only a few places were mentioned and in smaller frequency. This process is shown in Figure 3.15.

There are some obvious flaws with this method, such as when "Victoria" is a place within the given area it is likely that documents about Queen Victoria will be returned if the collection has any. However, cases like this are likely

to be rare and are will most weeded out by the fact that almost no other relevant places will be mentioned within those documents. Further research into the accuracy of this method is needed and, if after future developments it is decided that this spatial searching method is a worthwhile addition to the system then it is highly probable that this will be explored further.

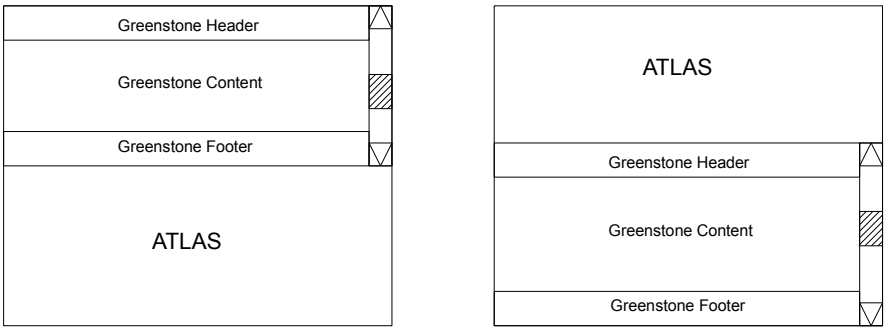
At the time of writing only this second spatial searching method has been implemented into ATLAS. It is likely that the first method will also be included in the system at a later point in time and it is possible that the two methods could work together. The second method could be used for collections that have not been specially prepared and the first method could be used for those collections that have had spatial indexes already prepared.

3.4 Interface design

Here we discuss the design of the ATLAS user interface. We begin with a detailed discussion about the page layout used in the standard document view, including the designs that were considered and the decisions that were made. We then discuss the design of the status area that is used to inform users of what the system is currently doing. This is followed by discussion of the three alternative text viewing methods and details about the two that were actually implemented for the system. In the final section we discuss the interactions that the system provides in standard document pages, the text search results page and the spatial searching page.

3.4.1 Page layout

The basic building blocks for the layout of Greenstone are HTML *divs*, which effectively allow the user to divide the web page up into different sections. The standard Greenstone layout is made up of three main *divs*: the header *div*, which contains the title of the current page and possibly other information such as the user's current position in the Greenstone hierarchy (e.g., "My



(a) (b)

Figure 3.16: Two variations of the horizontal split layout.

Greenstone Library → HDL Collection → Browse"); the content *div*, which is the main display area of the system; and the footer *div*, which displays the message "Powered by Greenstone3" or, in some Greenstone skins, is not shown at all.

For the ATLAS system several major page layouts were considered. Two of these were dividing the page in half vertically or horizontally, with Greenstone being displayed in one half, and ATLAS being displayed in the other. A horizontal split (shown in two variations in Figure 3.16) was quickly ruled out as a possible design choice, because having an area that is wider than it is high is generally accepted as a poor choice of layout for reading long blocks of text like those present in Greenstone collections. The original design of the system had a full vertical split with Greenstone on the left side and ATLAS on the right hand side (shown in Figure 3.17. The advantages of this were that the two separate applications were clearly divided and that the text was laid out vertically, making it visually more intuitive. To further integrate ATLAS into Greenstone the full vertical split was later relaxed to allow the header and footer *divs* of the Greenstone section to extend the full width of the page as shown in Figure 3.18.

Having the map on the left hand side of the page rather than the right was considered, as vertical scroll bars on web browsers are generally on the right hand side. This then meant the scroll bar would be on the same side as the text.

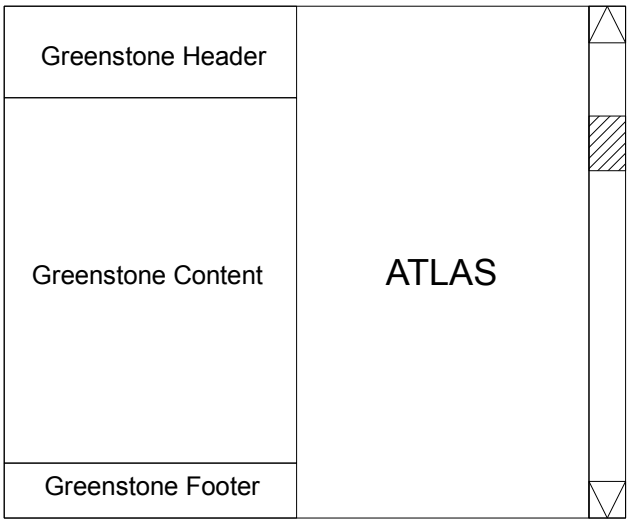


Figure 3.17: The original full vertical split layout of the system.

It was considered that maybe the user would be more comfortable with this layout as the scroll bar would feel more connected to the text than it would if it were on the left hand side. There was a disadvantage to this layout however. Users from languages that read from left to right place natural emphasis on the left side of the page, which in this case would be on the map. This may create confusion for new users as they may expect the map to be the primary source of information which is not the case.

It was clear that using a vertical split had several advantages over other layouts, so this basic concept was chosen to be developed further. Originally, the system used a full page vertical split, with all sections of Greenstone on the left hand side. It was decided that to further integrate ATLAS with the Greenstone page, only the content section of the Greenstone page should be split vertically. This meant that the header and footer sections covered the full width of the page. One option that was considered was having the map follow the page so that when the page was scrolled downwards, the map would also move downwards within its half of the page, meaning it was always in view.

This method of keeping the map within the user's view was experimented

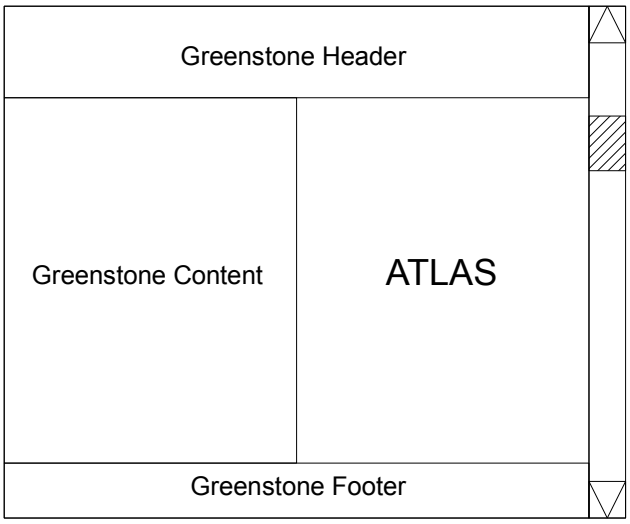


Figure 3.18: The full split layout with the Greenstone header and footer sections extending the full width of the page.

with by implementing it into the system. It had the potential to become visually quite distracting, as the page was scrolled the map would jump abruptly to its new position. This was due to their not being an *onScroll* event in Javascript to capture when the user scrolled the page up and down, the only alternative is to use a timer to check at regular intervals whether the page has been scrolled. Although this method works, using an *onScroll* event would allow this movement to be as smooth as scrolling itself. One way to work around the problem would be smoothly move the map to its new position rather than simply setting its position to its new value.

Rather than experiment with this motion smoothing idea, a different layout was conceived that would eventually become the final layout of the system. This layout was designed to fit all of the content on to the screen at once rather than having to use the main scroll bar (Figure 3.19 shows this design). This was achieved by using Javascript to manually adjust the height of the Greenstone content *div* and the ATLAS *div* so that they were the correct height in relation to the user's web browser. The ATLAS map was then scaled to fit its entire *div*. The Greenstone *div* had its CSS *overflow* property set to

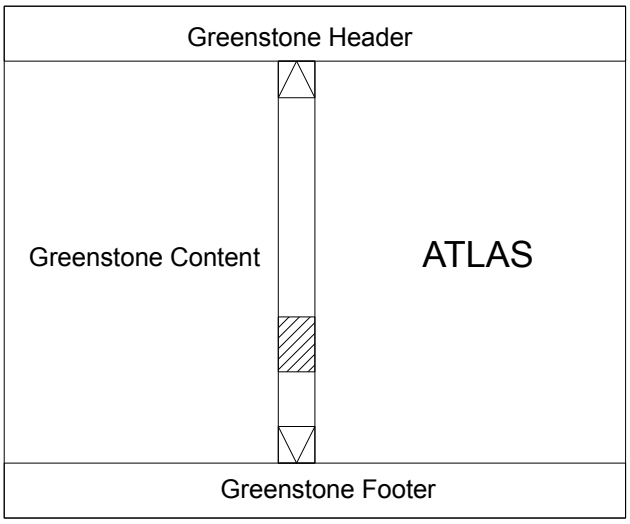


Figure 3.19: The final layout of the system.

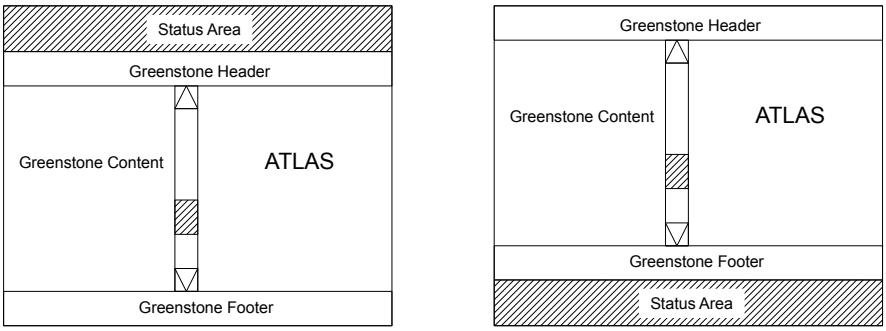
auto, meaning that if there was more content than could be shown in the *div* at its current size then a scroll bar would be added (rather than making the *div* larger). This solved both the problems of the distracting map movement (due to the map being stationary) and the document not having an obvious scroll bar as now the scroll bar would be alongside the text, rather than at the side of the web page.

3.4.2 Status Area

One of the most important aspects of usability is letting users know what's happening in the program as they're using it [Nie94]. With a large amount of the work of ATLAS being done on the server side of the system it is important to let users know when, for example, the server is busy loading the place recognition data structure or that the system is busy scanning pages for places, requiring the user to wait before they can continue using the system. Without letting the users know that this is happening they may think that the system has stopped working and may become frustrated. Also, because ATLAS loads pages differently than what users are accustomed to in a standard web site,

this becomes even more important as several aspects of the system are likely to behave differently than what the users expects. For example, normally when the user clicks on a standard link they expect the browser to change to the new page almost instantly and for the new page to then begin being built as the data arrives. In ATLAS however, there is a noticeable delay between when the user clicks on a link to when the new page is actually shown (due to the system waiting for the page to be loaded in the background). This could confuse users into thinking that either they did not click the link properly or that there is something wrong with the system. To help compensate for this change in standard behaviour ATLAS uses a status area that displays messages that let the user know what is happening. To continue the previous example, when a link is clicked on a message instantly appears in the status area saying "Loading new page" to let users know that everything is behaving normally. Once the page has finished loading the message and the status area disappear again (assuming that there is not more than one message being shown).

There were several aspects of the design of the status area that were important to consider. It is important that the status area is not distracting to users, but at the same time it is important that it is easily visible and that the user knows when it is displaying a new message. The placement of the status area has a strong influence on these two aspects; placing the status area at the top of the page (Figure 3.20(a)) would satisfy the need for it to not be distracting but has the disadvantage that the user may not notice it, or if they do they may not notice when a new notification is there to be read. If we take the opposite approach and the status area is placed at the bottom of the page (Figure 3.20(b)), it has the advantage that status bars are usually located at the bottom of interfaces and therefore would meet the user's expectations to an extent. Although this would naïvely seem like the best position to place the status area, it is more necessary in this case — more so than a standard status bar — that the status updates are easily noticeable due to the potential of user frustration being high if the status updates are missed. We instead



(a) (b)

Figure 3.20: Two of the status area position concepts.

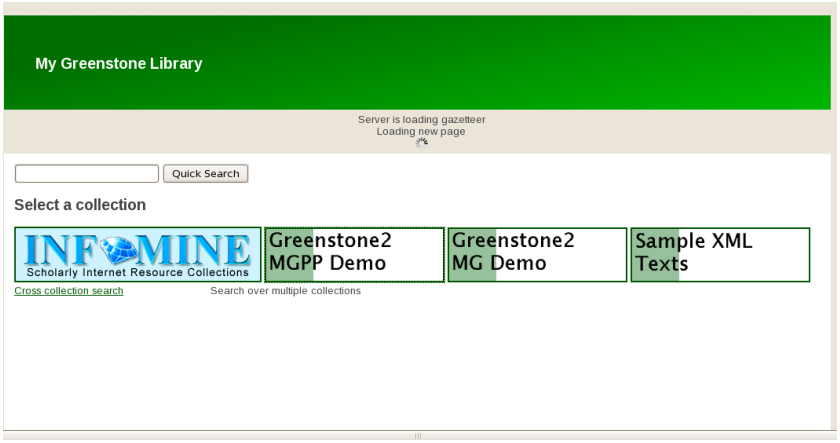


Figure 3.21: The status area showing two updates.

chose to place the status area between the header and content areas of the page (shown in Figure 3.21) so that it was easily visible at all times.

During the pilot tests of user study detailed in Section 4.1 it was discovered that, even with the status updates located in an easily visible place, people still tended to not notice them when they appeared. It was decided that something that was both eye-catching but not distracting needed to be added to the status area when there was a message so that it caught the user's attention but did not hold it unnecessarily. For this purpose we chose to add a small animated GIF file (shown in Figure 3.22) to the bottom of the status area when it was an update to be viewed. This animation also gives the user a feeling that the

Figure 3.22: The animated GIF used on the status area.

system is doing something and has not crashed, which is important if the user is forced to wait a long time. Importantly, this animation was not distracting due to its plain colours and simple movement but was still enough to catch the user's attention when it appeared. An animation like this is a common technique in AJAX applications to inform the user that the system is loading something.

3.4.3 Text presentation

With the layout of the page having been finalised to restrict the content area of Greenstone documents to half of the page, it meant that documents required more scrolling to read than what they had previously. To counteract this it was considered whether alternative ways to view documents — that would require less scrolling — would be beneficial to users. Three methods were tested, two of which have been implemented in the final system.

Fisheye view

The first method that was explored was a fisheye text visualisation method. This was implemented so that the text near the user's mouse cursor would be larger than the text of the rest of the document. This allowed the user to get an overview of the document whilst still being able to read and navigate it. This was especially useful when places in the text had been highlighted, as it was easy to see where they were even when the text was out of the focus of the fisheye. Figure 3.23 shows a screenshot of the fisheye view in action.

The disadvantage of the fisheye view however is that there is much wasted whitespace in the context area (i.e., above and below the focused area). In Section 5.2 we discuss a possible way of utilising this whitespace to help users

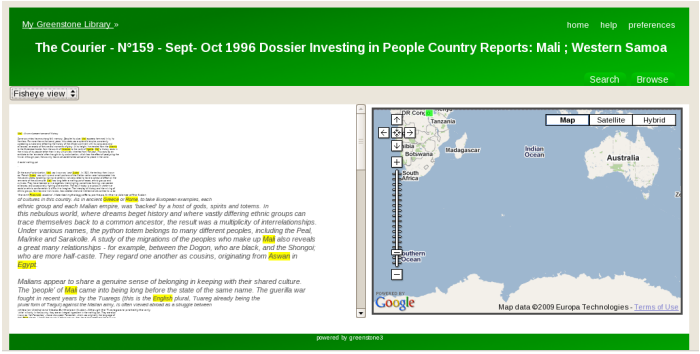


Figure 3.23: The document text in fisheye view.

quickly locate matching place names.

Several parameters were adjustable to modify different aspects of the fisheye system such as the maximum and minimum font sizes, the sharpness of the font size fall off, the width of the area around the cursor that remained at the maximum text size. A major factor that was considered in the process of choosing parameters was the size of the focus area. Having a focus area that is too small would mean that the text would be difficult to read. Having the focus area too large however would make the fisheye effect redundant. Finding a balance between these two ends of the spectrum is not a simple task. For instance, 12 of the 16 participants in the fisheye for source code user test performed by Jakobsen and Hornbæk felt that the context area was too small, yet if the context area had been made larger it is likely that their task completion performance would have been negatively effected due to the lack of extra context information.

Another major consideration was whether or not to attempt to fit all of the document text into the content area so that a scroll bar was not needed. It was decided that this would not be a good decision as, with long documents, the font size would need to be at a size where it was impossible to read or see any highlighted place names in the document, which would obviously make the content area useless. Unfortunately several users that participated in the user study discussed in Section 4.1 clearly expected the whole document to be

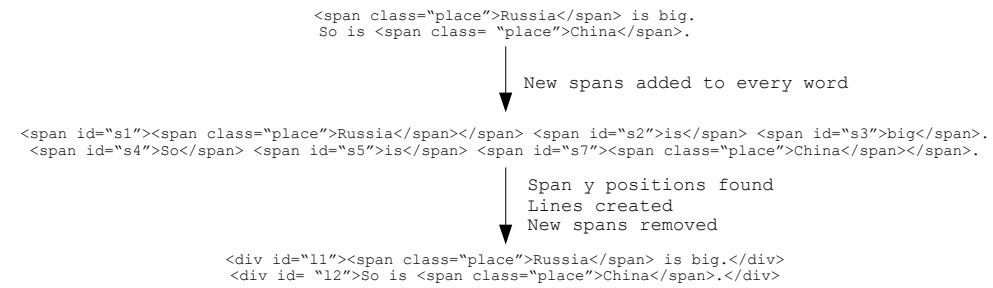


Figure 3.24: The line finding algorithm used to set up the fisheye view.

shown at once and did not realise that they needed to use the scroll bar to view the rest of the document.

Several novel techniques were used in the implementation of the fisheye view. Each line in the unmodified text needed to be located, as text resizing was done on a line by line basis. To do this the program initially places HTML *span* tags around each word in the text. Care is taken to exclude any other tags that may be already be present in the text. This is because treating tags as words will produce undesirable results. For example, a tag such as   will become   which is invalid HTML and will be displayed incorrectly in a web browser. Placing *spans* around each word can be used to locate the vertical height of each word in the text (through the use of the *getOffsetTop()* DOM method provided by the *span* tag) and in turn can be used to locate the lines in the text. Each line is then wrapped in a *div* tag and stored in an array, in order of their vertical location on the page. All the *span* tags used for line finding are then removed to increase performance as they are no longer needed. This process is demonstrated in Figure 3.24

Compaction view

The second text display method that was designed attempted to maximise the space in the content div by slowly decreasing the size of parts of the document

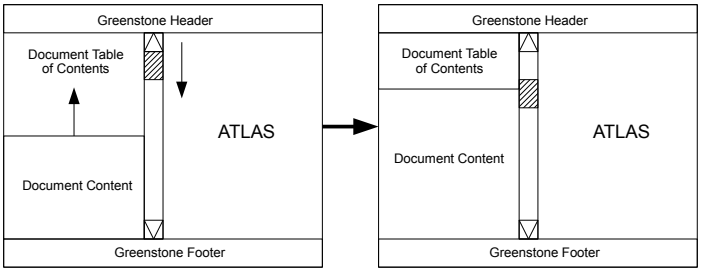


Figure 3.25: The compaction view design.

when they were no longer needed. This method was originally designed when the Greenstone skin that was being used was the default skin, rather than the development skin which is currently being used. In the default Greenstone skin, when a document is viewed it shows the table of contents at the top of the page. The idea of the compaction view was to slowly decrease the size of the table of contents as the user scrolled down the page until it was almost invisible at which stage it would turn into a button that would allow the user to display it again if they chose. This design is shown in Figure 3.25.

Several parameters were also adjustable with this method such as the speed at which the size of the non-important objects decreased relative to how far the page is scrolled. In the current version of ATLAS though this method was relatively ineffective as the table of contents of a document in the Greenstone development skin is already quite small and located on the side of the content area rather than at the top.

Column view

The third method used a similar concept as the fisheye method but was designed to better maximise the use of whitespace. It does this by decreasing the font size of the text and arranging it into several columns so that the whole of the document is displayed at once without the need for scroll bars. As the user moves their cursor over the text in the columns, a larger version of the text is displayed in a box centred above the position of the mouse cursor.

The advantage of this method is that it makes far better usage of whitespace than the fisheye method as almost none of it is wasted. In the example shown in Figure 3.26 the effective use of whitespace can easily be seen. A simple heuristic algorithm was used to quickly arrange the text into the appropriate number of columns necessary for the given amount of text. The full algorithm is shown in Figure 3.27.

The column view in ATLAS is implemented in a similar way to the Document Lens system [RM93] discussed in Chapter 2, however there are few key differences. Firstly, the ATLAS column view does not distort the parts of the document the lie outside of the "lens". The most obvious reason for this is that it is impossible to create this effect in a web browser without the use of Adobe Flash or other similar technology, so as the column view is coded in Javascript this functionality is currently out of reach. If this were not the case however, it would still be unlikely for this sort of effect to be implemented as the user may get lost in large documents. The benefit of not resizing content outside of the magnified area is that the user always knows where they are in the document regardless of its size.

The second — although minor — difference is that, being 3D based, the Document Lens system allows the user to move the lens in the Z direction (towards or away from the screen), effectively allowing the user to zoom out to a wider view or zoom in to a closer view. This is deemed unnecessary for the column view in ATLAS as there appears to be little benefit from implementing such functionality.

Several aspects of this text view required special consideration in order to provide good usability. One such consideration is where the box with the enlarged text should be in relation to the position of the cursor. ATLAS places the box 25 pixels above the cursor, unless the box's y position becomes less (higher up the page) than the top of the text frame, in which case the box is placed 25 pixels below the cursor. Having the box placed below the cursor by default was also considered but was quickly ruled out due to the fact that

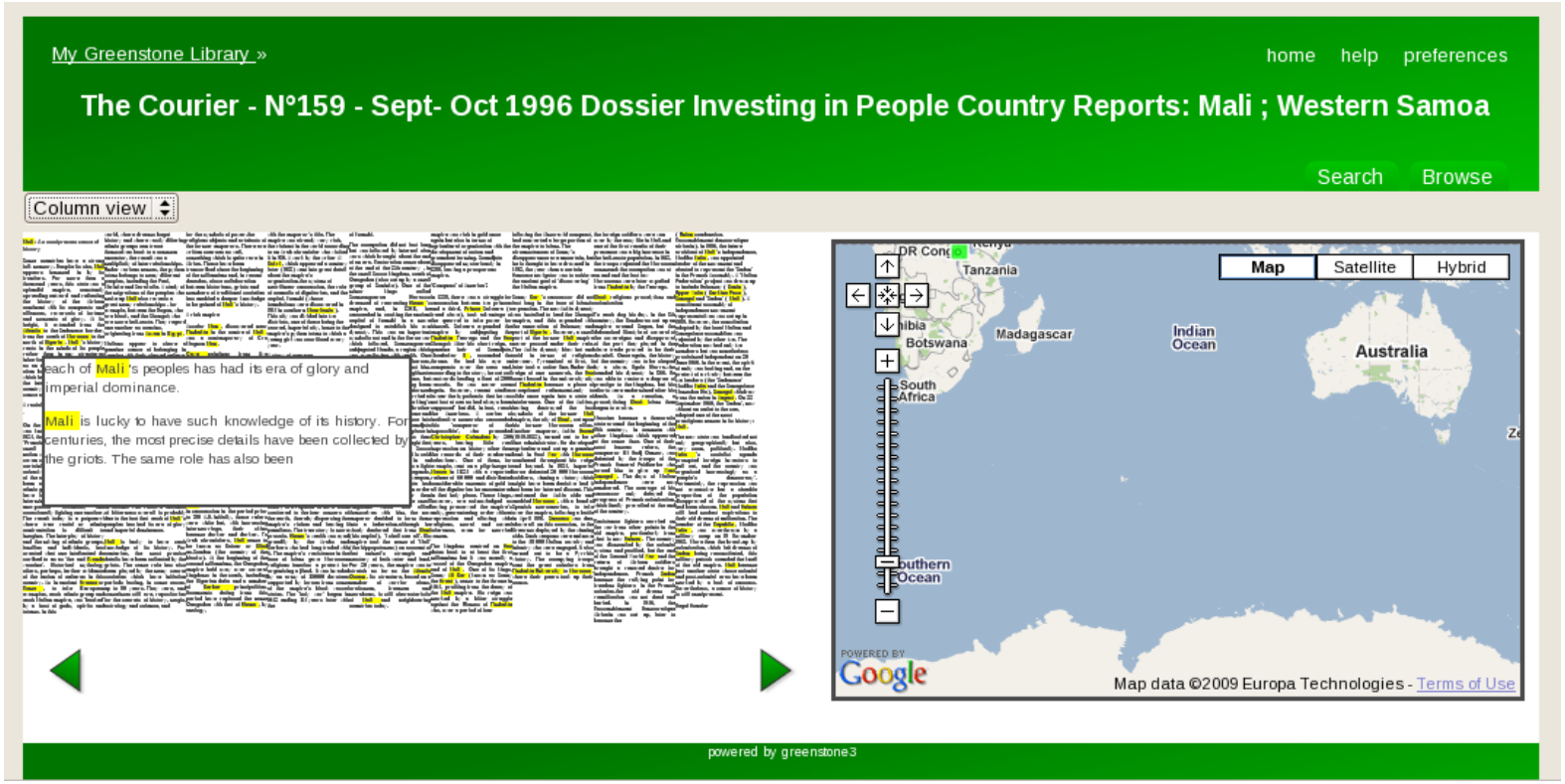


Figure 3.26: The document text in column view.

```
Let DY be the height of document <div> in pixels
Let TY be the height of the document text in pixels
Let FY be the font size of the document text
Let C be the number of columns
Begin loop
  If TY > DY then
    FY = FY * 0.85
    C = C + 1
  Else
    Stop looping
End loop
```

Figure 3.27: The heuristic algorithm used to arrange document text into columns.

the most common languages used in Greenstone collections read from top to bottom and having the box below the cursor would obscure the upcoming text.

Another design that was considered was having the box be centred on the cursor, the advantage of this method is that it is the most intuitive for the users, as it would act (to an extent) like the cursor was a magnifying glass. This method also has the disadvantage that it obscures upcoming text to a small extent due to magnification box being larger than the area it magnifies. The other disadvantage with this method is that it would be difficult to implement it the way that the user would probably expect it to work. The current system does not register movement in the x axis, it simply adjusts what is shown in the box based on the y position of the cursor on the current column. To create the full magnifying glass effect the user might expect, the box would have to take into account motion in the x axis which complicates the implementation to the point that it is out of the scope of this project.

Although research by Dyson and Kipping [DK97]found that users felt text laid out in columns appeared more organised and interesting, it is unlikely that these findings for layouts of two or three columns will extend to the greater column densities often present in the ATLAS column view. This prior research assumes that the text is readable in its multicolumn form. As the ATLAS

column view is designed to give a document overview, the text is usually too small to read for long documents as the text size is reduced to fit the whole document content on the page at one time. Although our research does not extend to measuring the reading performance of our text display method it is safe to assume that it will perform worse than the column views discussed in Dyson's research in both reading speed and in a preferred reading layout experiment. This is due mostly to the fact that the user must frequently move the mouse so that a new area of text is magnified, making it arduous to read a lengthy amount of text.

3.4.4 System interactions

The interactions between ATLAS and the Greenstone were an important aspect of the interface design. There are three places within the system where the interaction design needed particular consideration. These areas are the standard document view, the text search page and the spatial search page.

Document view interactions

Rather than just simply being able to see the locations of places mentioned in the text, we decided that it would be useful to be able to interact further with the system. With each document containing a previously unknown number of place names of an unknown concentration, interacting at an individual place name level requires special consideration. An interaction method needed to be chosen that was both intuitive for the user to use and could also provide easy access to several different features. It was decided that a menu system would be the best way for this to be achieved seeing as it is both intuitive — it is safe to assume most users will have used menus at some point in their computing experience — and can contain as many menu items as necessary. The next decision was how to best attach these menus to the places in the text.

Several methods were considered such as, having a permanent menu on the screen. The user could then click on a place names to select it and would

then be able to choose an option from the menu. We decided against this idea however due to the fact that this menu would take up space that could not be spared due to the current page layout already attempting to give the two page halves as much space as possible. It was decided that rollover menus would be the best option as the did not use up unnecessary space and were still intuitive to use.

When the mouse cursor is held over a highlighted place name, a menu is displayed showing options relevant to that place name (Figure 1.6 shows an example). Choosing the specific parameters of the rollover was also important to consider. It was decided that having the menu instantly appear when a place was moused over would have the potential to be frustrating for the user, as it may accidentally hide areas of the text they are trying to view. At the same time however, an interval that was too long could also be frustrating as the user would have to wait each time for the menu to appear. After some testing we decided that a one second interval was the best choice as it was long enough that the user would have to deliberately leave the mouse cursor in place but short enough to not be frustrating.

The menu items chosen to be available were: "Centre the map on this place", "Highlight this place on the map", "Highlight this place in the text", "Remove all highlights" and "Choose correct place" as can be seen in Figure 3.28 (Figure 1.6 shows an example of a menu in use). Being able to centre the map on a particular place is useful as it makes the specific place that is being searched for much easier to find, especially when there are many markers on the map. Another reason why this functionality is useful is because it takes care of moving the map for the user, so that when they zoom in a long way they will still be centred on that specific place rather than having to find it again in the zoomed-in view. In earlier stages of development, mousing over a place resulted in the place automatically being centred on the map. It was decided that this could become frustrating for users when they accidentally moused over a place name as they could lose the place they were currently examining

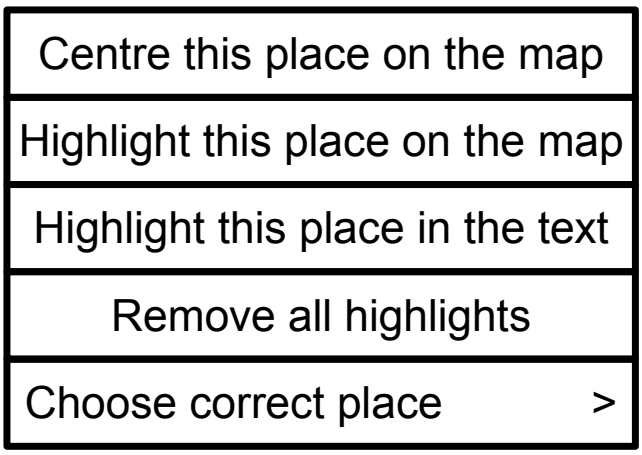


Figure 3.28: The document specific menu used in ATLAS.

for example. For this reason this functionality was moved to the menu so that the user could make a conscious decision about whether to centre this place on the map or not. Section 5.2 discusses possible future modifications to this functionality.

It was also decided that being able to highlight places in both the text and on the map would be useful functionality. Highlighting places in the text allows the user to quickly scroll through the document and see any more references to this particular place. Highlighting places on the map is potentially useful if the user is trying to locate a particular place marker amongst several other nearby place markers, or the user could highlight two places to easily see the distance between them for example.

Finally we chose to add the ability to change the place that a place name refers to. The place name disambiguation system performs adequately in most cases but there are certain situations where the system will make a mistake — such as when two places by the same name are referred to in the same document, the system will mark both of them as the same place. As it is rare for this to occur, a way to compensate for this has not been implemented in the system. It was decided that, rather than let the users become frustrated by an incorrect disambiguation, we would allow them the ability to change the

reference if they believed that it was incorrect. Changing the reference will also move the position of the marker on the map to the location of the new place the user has chosen.

Text search view interactions

To augment the standard Greenstone text search view we added a map to one half of the screen displaying the places located in each of the returned documents. Each document is given a colour and all of the places corresponding to that document are shown in that colour on the map (an example of this can be seen in Figure 1.4). To make sure colours are distinguishable from one another an algorithm is used to make sure that colours are distinct from each of the other colours. Colours were chosen to distinguish places of different documents — rather than another system such as adding numbers to the markers — because colours are an intuitive and natural method of visually mapping one set of information to another. Using numbers or another alternative method has the potential to confuse users, as they may question what the numbers mean. Colours are also much easier to locate amongst other colours than numbers are amongst other numbers.

The text search view has a high potential for clutter, especially if the content of the collection being searched is geographically oriented. When this occurs it can become difficult to clearly see the area (if any) that a document is centred around. To help in situations like this we decided that the ability to remove document place sets from the map would be helpful as it would allow the user to much more easily focus on the places contained within one or a few documents. For this functionality we considered using pop-up menus like those that were used in the document view as they would then provide a consistent method of interaction for users. We decided against this idea however due to the fact that each menu would have only contained two or three items in it ("Show/hide this document's markers" and "Show markers only from this document"). It was decided that using check boxes would provide a much more

intuitive method of turning on and off place markers relating to each document (the checkboxes can be seen to the left of the document links in Figure 1.4). This is because most users will have at least some experience with check boxes in the past — either on web pages or in other graphical user interfaces — and will therefore easily be able to understand what they do when they see them.

If in future development this view is ever extended then using menus may still be considered as they may help to group all of the functionality into one place. At this stage in development however this is not necessary or helpful.

Spatial search view interactions

As described in Section 3.3.7, ATLAS provides its own document search method in the form of spatial document searching (searching for documents by the locations of the places within them). The interaction design for this spatial search view provided an interesting challenge due to there being many different ways that each aspect of it could be designed. The first — and most important — aspect of the interaction was the designing how the users could select areas of the map to be searched. There were many possible ways for this to be implemented but a method was needed that was both simple and yet allowed the user to make complex area selections. Throughout this section we will use Switzerland as an example of a complex query to attempt to create. Switzerland's shape and neighbouring countries make it difficult to construct a query without including some of Italy, Austria, France, Liechtenstein and Germany. This can be seen in Figure 3.29.

One of the first designs was to simply allow the user to click and drag a box over the area they wished to use. In terms of simplicity this method is effective, as it is easy to use, fast and for many users would possibly be sufficient for specifying areas they wanted to query. The disadvantage of this method however is obviously its limited ability to construct complex queries. Figure 3.30(a) shows an attempt at creating a query for Switzerland; it can be seen that clearly this method is inadequate for a query of this complexity.

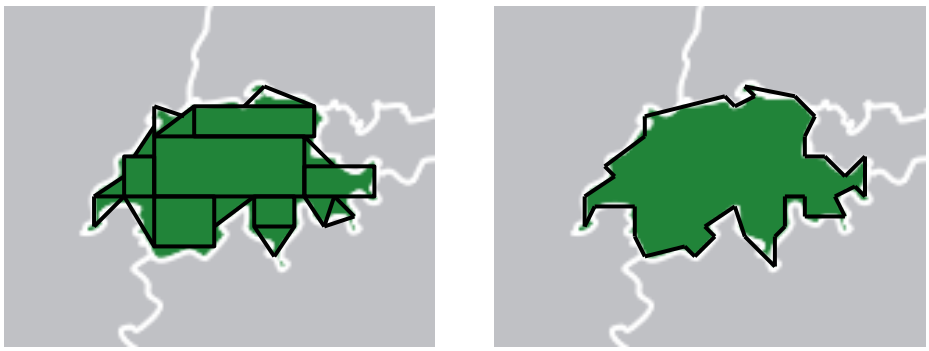
Figure 3.29: Switzerland with its neighbouring countries.

(a) (b)

Figure 3.30: Single box and multi-box spatial query examples.

A possible solution to this would be to allow the user to create multiple boxes to define a query. This would be adequate for low- to mid-level complexity queries but for advanced queries it becomes almost impossible. Figure 3.30(b) shows an attempt at creating the complex Switzerland query. Although this method is a significant improvement over the previous method it is still difficult to create such an intricate query.

Another similar method that was considered was allowing users to choose between several pre-defined shapes such as triangles, rectangles and circles and allowing them to click and drag the shape into place like in the previous



(a) (b)

Figure 3.31: Multi-shape and polygon query examples.

method. To fully meet the user's expectations of this area defining method it is likely that additional functionality, such as allowing the user to move, rotate and resize shapes once they are placed, would likely be needed. Although this method would make advanced queries significantly easier it would still be time consuming to construct them. This also has the potential to be too complicated for computer users with only a small amount of computing experience. Figure 3.31(a) shows that this method is much better for handling more intricate queries but is still overly complicated.

The method that we chose to implement was to allow users to construct queries by using points to create arbitrary polygons. This method is simple enough for all users to understand and is also capable of creating advanced queries with relative ease. Although this method is slightly slower than the three previous methods for simple queries it is faster and more accurate for queries that are more advanced. This is due to being able to simply click on the place where you want the polygon to extend to rather than having to find the correct shape to fill an area as well as possible. As can be seen in Figure 3.31(b) this method is both accurate and fast for an advanced query like Switzerland.

In future development of ATLAS it is possible that user could be given a choice as to which method they would prefer to use based on the complexity of their desired query. At this stage in development however we feel that the

method we have chosen provides a good combination of both simplicity and power. Section 5.2 discusses another possible spatial query defining method that will be considered for future development. The idea is to let users simply click on countries to select them as part of their query.

Chapter 4

Evaluation

In this chapter we will evaluate several specific areas of the ATLAS system including: its usability, its place name disambiguation accuracy and the efficiency of the trie structure used for place name recognition that has been specially designed for ATLAS.

4.1 User study

To evaluate the usability of the system a user study was designed. In the study we aimed to gather information about the experimental aspects of the program. With that information we hope to be able to make informed decisions about the effectiveness of the different aspects of the program.

In this section we first discuss the pilot tests that were run to test for any obvious problems that could effect the success of the full user test. We then present the results of the full user test.

4.1.1 Pilot tests results

The pilot tests were designed as short, informal tests of aspects of the system that were to be tested during the user study. Several minor problems were found during this process. Firstly, as mentioned in Section 3.4.2 we discovered that, despite the status area being clearly visible on the page, it did not draw

the attention of users as it had been designed to do and was often overlooked. It was suggested that an animation be added to the status area to both attract the user's attention when it was needed and also let the user know that the system was still working even if there was no other visual feedback that made this clear.

Another potential usability problem that was discovered was that it is sometimes difficult to differentiate between the rectangle place markers of similar colours in the text search view. It was discovered that one contributing factor was the transparency used to make it easier to see the place that had been marked through the marker. If the marker is too transparent it made the colour much harder to distinguish due to it being partially combined with the colour of the map beneath it. To compensate for this the transparency was changed so that it increased when the map was zoomed closer and decreased when the map was zoomed out, as both the opaqueness and transparency are important depending on how close the map has been zoomed.

This transparency modification function partially solved the problem of markers being difficult to distinguish from each other, but even with the colours being more easily distinguishable there is a limited amount of times that the colour spectrum can be divided before the colours are no longer easy to tell apart. Although cases where the spectrum will need to be divided to this point are likely to be rare, it was still important that some functionality be added to help handle such cases. It was decided that a good solution to this problem would be to add extra information to the marker pop-up information that is displayed when the marker is clicked. This extra information states what documents this specific place was referenced in so that if the user was having trouble differentiating the colours of the markers they could make sure by clicking on the marker to see this extra information.

4.1.2 Full user test results

The full user test was designed so that participants were to make full use of all of the features of ATLAS (the user study material is shown in Appendix A). Ten participants (four females) took part in the user test with their ages ranging from 20 to 52. Levels of computing experience also varied from frequent computer users with computer related educations to infrequent users with limited computing experience. Such a wide variety of participants were used to test the system because ATLAS is aimed at the same audience as digital libraries — many of which are founded on the principle that the information contained with them should be easily accessible to all who wish to view it.

The Greenstone collection used for the user study was the MGPP Demo collection which is a small subset of the Humanity Development Library. The documents in this library contain useful information for developing countries, such as how to farm butterflies or snails. This collection was adequate for the user study as most documents contained several place names.

Participants were first given a brief introduction to the system, explaining about digital libraries and what ATLAS does to enhance them. They were also given an example of how to browse for a document, how to perform a text search and how to perform a spatial search. After this introduction participants were asked to complete a series of tasks, each with one or two questions to answer. Tasks were deliberately designed to encourage users to use each of the features that are provided by ATLAS. For example, one question asked the participants to use the features available to find Costa Rica on the map (which is likely to be a difficult task for people with limited geographical knowledge) to encourage participants to explore the menu features. Once the tasks were completed the participants were then asked to complete a brief questionnaire about how they felt about the system and what they felt could be improved. The copy of the task sheet and questionnaire is shown in Appendix A.

The user test was designed to help answer the following questions:

- Is the functionality provided by ATLAS useful?
- How easy/intuitive is it to use the features provided by ATLAS?
- How useful were the fisheye and column views?
- What changes could be made to improve the system?

Did participants find ATLAS useful?

Almost all participants stated that they believed having a map that shows the places mentioned in the documents they read was something that they would find helpful. The first question on the survey given to each participant was "Do you think that having a map to show where places are located is helpful when reading documents? Why?". Some examples of answers given by participants include: "Yes, because I like to know where in the world places are in what I am reading. It gives more background", "Yes, often you've heard of the place but don't know where it is" and "Yes, many people, including myself, don't have a good idea of world geography and therefore the context of information".

Clearly the main reason that participants approve of the idea of marking places in documents on a map is that it gives more background context by allowing users to actually visualise the locations that are mentioned. This in turn gives the user further information such as the distance between places and how they are arranged. Some users are likely to find these simple pieces of information interesting and useful, depending on the specific type of document being read.

Did participants find the system intuitive and easy to use?

There were three main aspects of the program that need to be considered to answer this question as, from the user's point of view, the system is made up of three components: ATLAS, Greenstone and Google Maps. The way these components are combined means that having a weakness in one of these three components affects the usability of the system as a whole. For example, two

users had not used Google Maps before and therefore struggled to complete several of the tasks despite being able to adequately use the rest of the system. Overall however, from both the comments of the participants and the fact that most participants completed the tasks without difficulty, the usability of the system has shown to be good.

To test the standard document view participants were given several tasks that used each of the features available in the place-specific menu. Most users had no trouble using these features and were able to complete these tasks quickly. These participants also made positive comments about the system. One of the questions asked in the survey section of the task sheet was "How easy was it to use the features available to find the place you were looking for on the map?", two examples of answers written by participants are "Very easy - great!" and "Easy, exactly as I'd expect".

Although users found the features ATLAS provides in the standard document view easy enough to use, there were several aspects of its functionality that caused minor problems. Several users expected to be able to use the right mouse button to bring up the menu as is usually possible in other programs. Several other uses expected to be able to left click on the place name to be able to bring up the menu. The current implementation requires the user to hover the cursor over the place name for small amount of time before the menu is shown, however this is likely to be modified after this information provided by the user test. As it is common for menus to be attached to right mouse button clicks and users expected it, it is likely the the menu implementation will be changed in future development so that either hovering over the place name or right clicking the place name will bring up the menu. For the left click functionality two participants suggested that left clicking on a place name should centre the map on the corresponding place rather than needing to choose "Centre this place on the map" from the menu. As this was the most common menu item selected in the tests it makes sense for it to have a shortcut and using left clicks provides an efficient way of doing this.

The ATLAS-enhanced text search view was evaluated by having the participants search for the term "water" and then looking for places on the displayed map that came from the document "35 Bees". The goal was that users would look at the colour that was given to the "35 Bees" document and then look for markers of that colour on the map. Of all of the features provided by ATLAS, the text search view takes the longest to load, taking several minutes to scan all of the returned documents for place names. Several users proceeded to click on the link to the "35 Bees" document rather than wait for this map to load. One of the improvements planned for ATLAS in the future is the use of spatial indexes. Spatial indexes will allow the system to know in advance the places that are present in each document and therefore will not need to scan each document that is retrieved and therefore there will no longer be such a large delay for the map to be shown.

The spatial document search view was tested by having each participant create a simple query of Tasmania, Australia by drawing a polygon around it. They then clicked the "Search Area" button and were taken to a list of result documents of which they were asked to write down the first 3. All participants completed this task successfully, although those that had not used Google Maps before required a small amount of assistance.

There was one minor aspect of the Greenstone system that proved to have poor usability and confused several participants. After the user selects "Browse" on the collection home page they are taken to a page that gives the user options as to how they wish to browse the collection (shown in Figure 4.1). Several users were confused at the sight of this page, most of these confused users did not realise the page was loaded and one user thought a table was being loaded and so waited. There are two major contributing factors to this problem, the first problem is that the page is mostly whitespace with only a small strip of green along the top of the content area meant that several users did not notice the options at the top of the page. The other contributing factor is that the browsing options do not appear like links that can be clicked

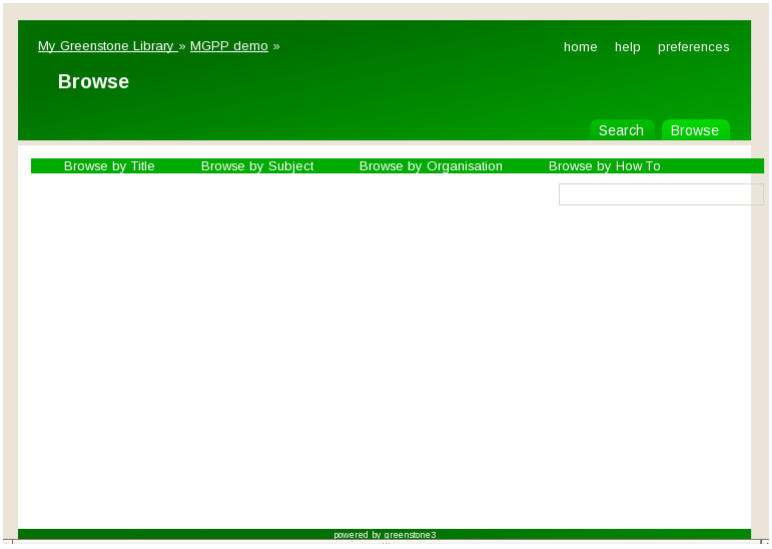


Figure 4.1: The browsing method selection page.

on due to the fact the links are not underlined like regular links. As the rest of the links in the system are underlined this could be considered to be violating Web-Wide conventions which is listed as one of the top 8 usability problems [NL06].

How helpful did participants find the fisheye and column views?

The fisheye view was tested by having the participants navigate to a long document the activate the fisheye view. Users were then asked to give a rough estimate of how many highlighted place names there were in the document. Although the system had found 49 place names in the text, user's answers varied significantly. The lowest estimate was 12 — with the next lowest estimate being 20 — and the highest estimate was 60, this large variation is a clear indication that there were some usability problems with the fisheye view.

The most obvious reason that there was such a varied array of estimates was that several participants did not realise that there was more to the document than what was initially shown when the fisheye view was presented to them. They counted the places they could currently see and assumed that it was the

whole document. This is a fair assumption for these participants to make as it is common for user interfaces that make use of the fisheye concept to attempt to display all of the content at once (such as the FishNet system discussed in Section 2.2.1). Whether or not to attempt to fit all of the text into the one area was something that was considered when the fisheye view was first developed. It was decided against due to the size of the text needing to be so small for long documents that it was almost invisible due to the smaller font size causing smaller line lengths as the number of characters per line remained the same.

Another reason contributing to the poor usability of the fisheye was that users easily got lost when scrolling the document as the document and the fisheye focus moved at the same time, making it difficult for them to remember where they were up to so that they could continue counting place names. One of the questions asked in the survey was "How easy did you find the fisheye view to use compared to viewing the text normally? What did you like/dislike about it?", the responses to this question included remarks such as "Made scrolling confusing and ineffective ...", "I liked it because it was different but disliked it because it confused me" and "Had trouble with the Fisheye, I couldn't get it to do what I wanted it to do".

Participants getting lost in the text when scrolling the fisheye view was also reported by both Jakobsen and Hornbæk in their evaluation of fisheye for source code [JH06] and Baudisch et al. with their evaluation of their FishNet system which is a fisheye internet browser [BLH04]. This finding is particularly problematic for the ATLAS system as, because documents are designed for reading sequentially, having users able to get easily lost in viewing method that was designed to increase document readability is counterproductive.

The column view was tested by having the participants choose column view in same long document that the users had used the fisheye view. Participants were then asked to locate three different place names in the text and write them down. All participants completed this task easily after a brief explanation of

how it works. This view also received more positive remarks such as "Easy, its logical for me to use", "... I like to be able to see all the text at the same size so I can scan over it easier" and "Column view was good. You could see how much there was to read, and I found it easy to scroll through ...". The only negative comments were that it was "a bit daunting" and "difficult to skim read".

Whether or not the two alternative text display methods will actually be used in practice remains unknown. Although this could have been asked as a question in the questionnaire its results would not have been helpful due to the limited amount of time each participant spent using each view. Ideally participants would have spent at least a few hours using each view so that they have a chance to fully realise the benefits and/or problems of each text display method.

What improvements were suggested by participants?

One of the questions asked in the survey given to participants was "What improvements would you suggest could be made to the system?". Several basic improvements were suggested such as:

- Centering the map on a place when its name is clicked on in the text.
- As the user browses through the "Choose correct place" move the marker and centre the map on each place as they mouse over it.
- Put a "Loading" box over top of the document text when either the column view or fisheye view is being loaded.
- Darker highlighting in fisheye (highlights are hard to see).
- Different colours for cities and countries.
- Smaller map size in comparison to the text.
- A place-specific menu option that takes the user to a page about that place (i.e., a Wikipedia page).

- In the "Choose correct place" submenu have the currently selected place as a disabled option the top of the list.

One of the more advanced suggestions was extending the gazetteer to use historical place names, such as those used by Smith and Crane in the Perseus project's historical digital library [SC01] or biblical place names so that places can be found in documents relating to those eras. This would not be too difficult to achieve in ATLAS, if the resources were available then they could be easily added to both the place name recognition system and disambiguation system and the place information retrieval system.

It was clear that some participants felt that dividing the screen in half for the two sections was not the best way for the system to be arranged. Several participants suggested either decreasing the size of the map (so that the text area would be larger) or offering some way of resizing the map at will. Besides the alternative text viewing methods — which are designed to make better use of the small text area than the standard document view — another feature that was considered during the development of the current ATLAS system was the ability to turn either the map view or the text view on or off with buttons, allowing either view to take up the full screen if necessary. This feature was not implemented however as it was hypothesised that the alternative text viewing methods would be adequate for resolving the problem of having a reduced area for text. This research however suggests that this hypothesis may be false and it is likely that either allowing users to hide one of the two main sections or allowing the sections to be resized will be implemented into a future version of ATLAS.

4.2 Place name disambiguation accuracy

As described in Section 3.3.5, ATLAS uses a scoring system to assist in the disambiguation of places. As there are many place names in the world that refer to more than one place it is necessary to calculate which of these places

is the best match for each ambiguous place name if it is to be marked on a map.

Here we will experiment with the scoring procedure and evaluate its precision. We will not be evaluating the recall of the system as we have designed the experiment so that the recall is 1.0 (meaning that the system will always find all of the relevant place names). When the test data was prepared it was set to only include places that were in the gazetteer, all places that were not in the gazetteer were not included. This was done so that the results were not negatively skewed due to gazetteer being insufficient. Also, all places that had only one match in the gazetteer were also not included, this was so that the results would not be positively skewed by these places that were impossible for the system to define incorrectly. As test data we manually tagged five Wikipedia pages, specifically the Australia (123 ambiguous places), Canada (189 ambiguous places), Ireland (60 ambiguous places), South Africa (59 ambiguous places) and the state of Washington (76 ambiguous places) pages.

There are four main parameters available to adjust in the scoring system, each of which can be adjusted between 0 and 1. They are as follows:

- **Partial child score percentage (PCS):** Each time a place is located in the text X is added to its score. This parameter adjusts how much of X is also given to the parents of this place. For example, if "Hamilton" is found in the text then all Hamiltons get X score added to their score, part of X is then added to the parents of each Hamilton. If this parameter is 0 then none of X is added to the parent's scores, if this parameter is 1 then all of X is added to the parent's scores.

- **Parent bonus limiting percentage (PBL):** This parameter controls what places are allowed to add part of their score to their children once the initial scoring process has been completed. For a place to be allowed to do this it must have a score close to that of the top scoring place. If this parameter is 0 then only the top scoring place can add part of its score to its children, if this parameter is 1 then places with a score

within 50% of the top scoring place are allowed to add part of their score to their children.

- **Parent bonus percentage (PB):** This parameter determines how much of a parent's score is added to its children. If this parameter is 0 then none of its score is added, if this parameter is 1 then all of its score is added.

- **Indirect reference penalty percentage (IRP):** This parameter controls how much of a place's score is removed if it is not directly referenced in the text. For example, if "Hamilton" is mentioned in document text but "Waikato" and "New Zealand" are not, then part of their scores are removed. If this parameter is 0 then the place's score is reduced to 0, if the parameter is 1 then no score is removed.

The goal of this test was to ascertain what combination of parameters gave the most accurate results. Each parameter was divided into six possible values (0, 0.2, 0.4, 0.6, 0.8 and 1), each possible combination of these parameters was tested for each document (1296 tests for each document) for a total of 6480 tests.

Of the 1296 possible combinations 56 produced the highest result of 98.62%, meaning that it correctly disambiguated 500 of the 507 place names in the five documents. The top scoring combinations are shown in Table 4.1. Three combinations got the lowest score of 73.37% (meaning they correctly disambiguated 372 of the 507 place names), these are shown in Table 4.2.

The most obvious factor contributing to whether or not a combination is successful is the value of the IRP parameter. As can be seen in Table 4.1, all but one of the combinations has its IRP parameter set to 0, which means that places that are not directly referenced in the text get their scores set to 0 in the post-scoring adjustment procedure. On the other hand, the three lowest scoring combinations had their IRP parameter set to 1.0, meaning that no score was removed from places that were not directly mentioned in the adjustment

#	IRP	PB	PCS	PBL		#	IRP	PB	PCS	PBL
1	0.0	1.0	0.4	1.0		29	0.0	1.0	0.4	0.8
2	0.0	0.6	0.6	1.0		30	0.0	0.2	0.2	0.8
3	0.0	0.8	0.2	0.8		31	0.0	0.8	0.2	1.0
4	0.0	0.2	0.2	0.6		32	0.0	0.2	0.2	0.0
5	0.0	0.4	0.6	1.0		33	0.0	0.4	0.6	0.8
6	0.0	0.6	0.2	0.8		34	0.0	1.0	0.6	0.8
7	0.0	0.8	0.6	1.0		35	0.0	0.8	0.4	0.6
8	0.0	0.4	0.2	1.0		36	0.0	0.8	0.6	0.8
9	0.0	0.6	0.4	0.6		37	0.0	0.6	0.2	0.4
10	0.0	0.6	0.4	1.0		38	0.0	1.0	0.2	0.4
11	0.0	0.6	0.6	0.8		39	0.0	0.8	0.2	0.0
12	0.0	0.8	0.4	1.0		40	0.0	0.2	0.4	0.6
13	0.0	0.4	0.2	0.2		41	0.0	0.8	0.4	0.8
14	0.0	0.4	0.2	0.6		42	0.0	0.4	0.4	1.0
15	0.0	0.6	0.2	0.2		43	0.0	0.4	0.2	0.4
16	0.0	0.8	0.2	0.4		44	0.0	0.2	0.6	0.8
17	0.0	0.6	0.2	1.0		45	0.0	0.8	0.2	0.2
18	0.0	0.4	0.2	0.8		46	0.0	0.4	0.4	0.6
19	0.0	0.2	0.6	1.0		47	0.0	0.2	0.2	0.2
20	0.0	0.4	0.2	0.0		48	0.0	1.0	0.2	1.0
21	0.0	0.2	0.2	0.4		49	0.0	1.0	0.2	0.2
22	0.0	0.2	0.4	1.0		50	0.0	0.6	0.2	0.6
23	0.0	0.4	0.4	0.8		51	0.0	0.2	0.2	1.0
24	0.0	1.0	0.2	0.0		52	0.0	1.0	0.2	0.8
25	0.0	1.0	0.6	1.0		53	0.0	1.0	0.2	0.6
26	0.0	0.6	0.2	0.0		54	0.2	0.2	0.2	1.0
27	0.0	0.6	0.4	0.8		55	0.0	0.8	0.2	0.6
28	0.0	0.2	0.4	0.8		56	0.0	1.0	0.4	0.6

Table 4.1: The 56 top scoring combinations of the disambiguation system scoring parameters (with 98.62% precision).

phase. This meant that the scores of these places could skew the results away from the actual focus of the document, especially if the PBL parameter was high (which it is in the three lowest scoring combinations).

The data suggests that the next most important parameter is the PCS parameter. In the top scoring combinations it had an average value of 0.325 with a standard deviation of 0.155 and its values varied between 0.2 and 0.6. With the lowest scoring combinations having an average PCS value of 1.0 it is clear that having a low value for the PCS parameter that is above 20% is important. A low PCS parameter means that only a small amount of the score given to the places that match the located place name is given to its parents

#	IRP	PB	PCS	PBL
1	1.0	0.4	1.0	1.0
2	1.0	0.2	1.0	0.8
3	1.0	0.2	1.0	1.0

Table 4.2: The three lowest scoring combinations of the disambiguation system scoring parameters (with 73.37% precision).

	PB	PCS	PBL
Average	0.593	0.325	0.661
Standard Deviation	0.288	0.155	0.323
Lowest value	0.2	0.2	0.0
Highest value	1.0	0.6	1.0

Table 4.3: Summary statistics of the PB, PCS and PBL parameters.

(a high value means that the same score is given to both the places and their parents).

It is difficult to analyse the effect of the other two parameters due to there being no clear pattern in their values. The average PB parameter value is 0.593 with a standard deviation of 0.288 and its values varied between 0.2 and 1.0. The average PBL parameter value was 0.661 with standard deviation of 0.323 and its values varied between 0.0 and 1.0. With such central values combined with high standard deviations it is difficult to infer anything from these values on their own (these values are summarised in Table 4.3).

To see if some correlation could be found between the PB and PBL parameters and the PCS parameters we grouped the combinations into three groups by their PCS values (0.2, 0.4 and 0.6). The PB parameter continued to vary across the groupings with averages of 0.587, 0.600 and 0.600 and standard deviations of 0.292, 0.293 and 0.298 respectively (summarised in Table 4.4). The PBL parameter however, showed a strong correlation between it and the PCS parameter with averages of 0.516, 0.800 and 0.900 and standard deviations of 0.353, 0.169 and 0.105 respectively (summarised in Table 4.5). It clear that, as the PCS parameter is increased, the PBL parameter should also be increased. If however the value of the PCS parameter is lower then the specific value of the PBL parameter is less important. This data implies that as the parents are

PCS value	**0.2**	**0.4**	**0.6**
Number of values	31	15	10
Average	0.587	0.600	0.600
Standard deviation	0.292	0.293	0.298
Lowest value	0.2	0.2	0.2
Highest value	1.0	1.0	1.0

Table 4.4: The statistics for the PB parameter when grouped by the PCS parameter.

PCS value	**0.2**	**0.4**	**0.6**
Number of values	31	15	10
Average	0.516	0.800	0.900
Standard deviation	0.353	0.169	0.105
Lowest value	0.0	0.6	0.8
Highest value	1.0	1.0	1.0

Table 4.5: The statistics for the PBL parameter when grouped by the PCS parameter.

given a larger portion of their children's score that the effect that the parents have on the overall results should be increased.

In summary, it has been found that the most important factor contributing to the precision of the place name disambiguation system used in ATLAS is the value of the IRP (Indirect Reference Penalty) parameter. If the parameter was its lowest possible value of 0 then the system performed at its best. If, on the other hand, the value was 1 then the system performed at its worst. The IRP parameter controls how much of a places score is removed if it has not been explicitly mentioned in the document. This finding suggests that places that are not explicitly mentioned should have their score completely removed.

The next most important parameter is the PCS (Partial Child Score) parameter which — along with the IRP parameter being set to 0 — must be set to a low value such as 30%. This means that, for example, if "Christchurch" is mentioned in a document then Canterbury will recieve 30% of the score given to Christchurch and New Zealand will recieve 30% of the score given to Canterbury.

The other two parameters have no clear direct relation to the precision of

the system, however the PBL (Parent Bonus Limiting) parameter showed a strong correlation between it and the PCS parameter, revealing that, if the PCS parameter was high (i.e., around 60%) then PBL parameter should also be high (i.e., around 90%, which results in places that are within 45% of the top scoring place adding a percentage of their scores to their children) to maintain high precision in the system.

4.3 Gazetteer trie efficiency

In this section we evaluate the efficiency of the gazetteer trie structure discussed in Section 3.3.4. The top priority of the gazetteer is achieving fast place name recognition with a secondary goal of maintaining an acceptable level of memory usage. Four methods of organising the trie structure are evaluated, the are as follows:

- **Full node trie:** Every node in this structure has a full set of about 65,000 children nodes (one for each possible Unicode character). This structure is designed to trade memory usage for speed.

- **Hash table trie:** Every node contains a hash table that maps the input characters to their corresponding child nodes. This is effectively the opposite of the full node trie as it trades speed for memory usage.

- **Hybrid node trie:** This trie uses a full set of nodes for English characters and a hash table for other characters.

- **Modified hybrid trie:** This trie uses a full set of nodes for all characters at the root node and uses hybrid nodes for all other nodes. This modification was made upon analysing the distribution of Unicode characters in the hybrid node trie. It was noticed that the root node had a significantly higher concentration of Unicode nodes that any other node(858 Unicode nodes, with the next most concentrated node having 42 Unicode nodes).

Method	Memory (bytes)
Full node	N/A
Hash table	208,349,680
Hybrid node	247,465,472
Modified hybrid node	247,682,968

Table 4.6: Memory usage of the various trie structures.

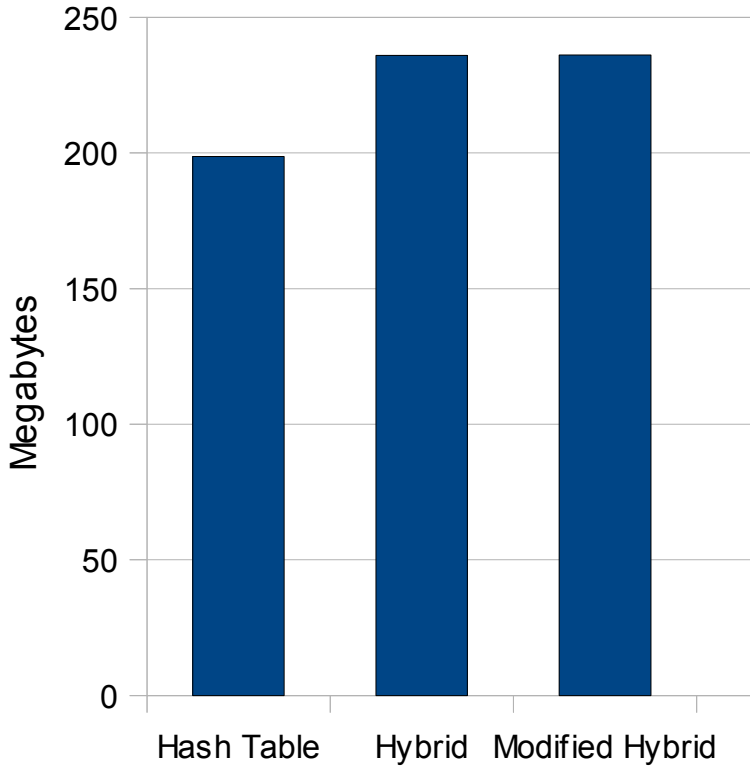


Figure 4.2: Data structure memory usage in bytes.

The memory consumed by each of the structures is shown in Table 4.6 and is graphed in Figure 4.2. We were unable to get a memory reading for the full node method as the Java Virtual Machine (JVM) threw an exception indicating that it was out of memory — despite us increasing the maximum heap size of the JVM from 64MB to 2048MB. The exception was thrown before the structure had even loaded 10,000 entries (out of the required 300,000+ entries), confirming our earlier hypothesis that building a full trie structure for Unicode was infeasible.

To test the speed of the different structures we created a test file from the names of all of the places in the gazetteer. This file was then fed through each

Method	Time (ms)
Full node	N/A
Hash table	532
Hybrid node	493
Modified hybrid node	491

Table 4.7: Task completion time for each of the data structures.

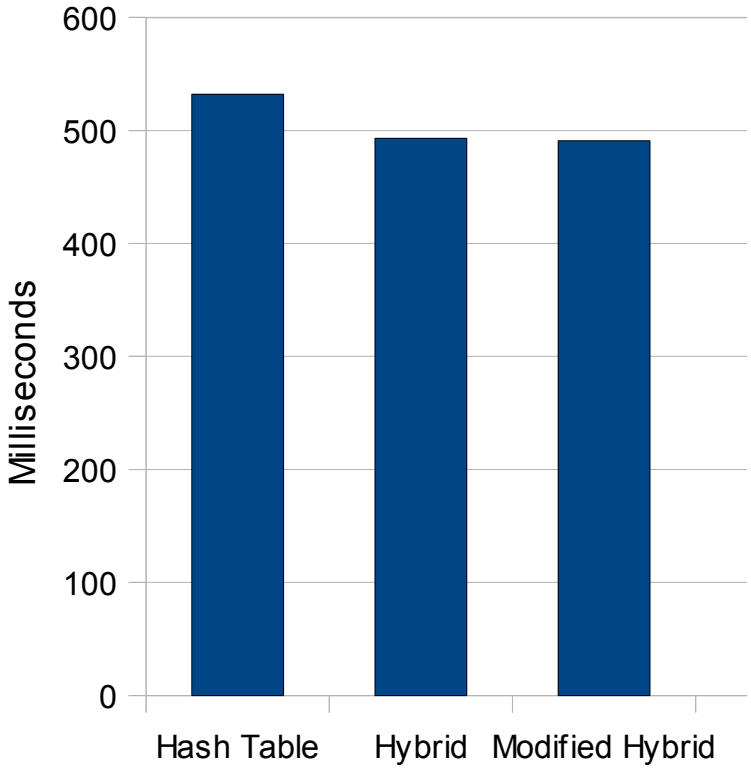


Figure 4.3: The speed of each of the structures in milliseconds.

data structure as if it were a document, the time it took the structure to locate all of the places was recorded. Places were not scored or disambiguated as this is not the purpose of this data structure. The test was repeated 100 times for each data structure and then averaged, the results of this experiment are shown in Table 4.7 and illustrated in Figure 4.3. The experiment was run on a Linux machine with a 3.0GHz Intel dual-core CPU and 2.0GB of RAM.

Again we were unable to measure the speed of the full trie structure due to its inefficient use of resources causing an out of memory exception to be thrown by the JVM. It is likely that it would have been the fastest of all of these methods but far too inefficient in its usage of memory to make it worth

Method	Time (ms)
Hash table	1297
Hybrid node	1346
Modified hybrid node	1267

Table 4.8: Time taken for each of the data structures to parse the Unicode-oriented data set.

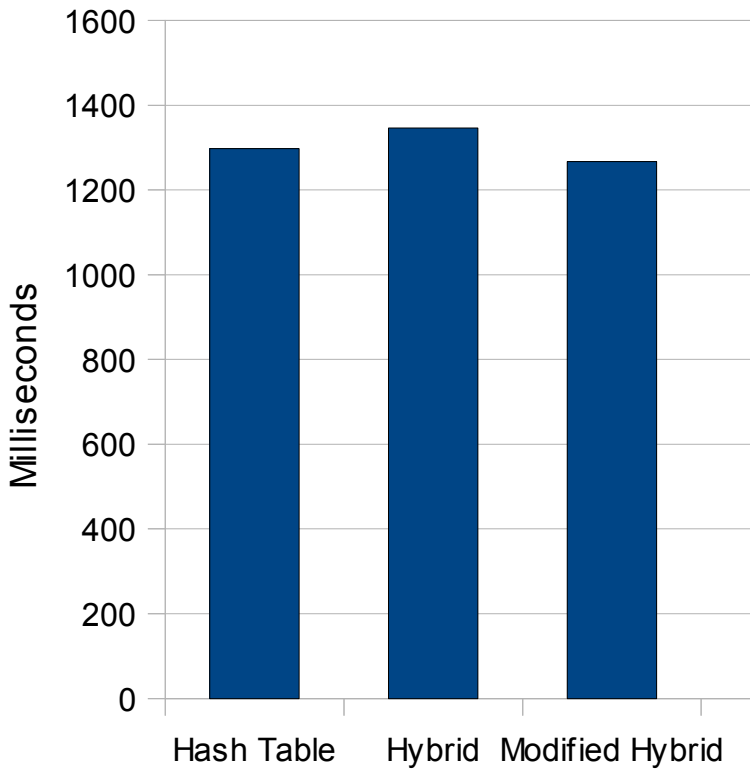


Figure 4.4: The time taken for each structure to parse the Unicode-oriented data set (in milliseconds).

considering.

Interestingly the hybrid structure and the modified hybrid structure performed equally well which was unexpected. We suspected that the data set being used did not contain enough Unicode characters to truly highlight the potential difference in the two structures. For this reason we created a second data set but this time only the names that contained at least one Unicode character were included. To get a more accurate result we also duplicated the data set three times. The experiment was repeated and the new results are shown in Table 4.8 and illustrated in Figure 4.4.

As expected, this new data set shows more clearly the benefits of the modified hybrid structure. The modified hybrid structure performs almost 6% better than the unmodified hybrid structure. Surprisingly the hash table structure performed better than the unmodified hybrid structure. This performance difference is likely due to the extra calculations required by the unmodified hybrid structure to compensate for when a Unicode character is recieved. These calculations are not necessary in the hash table structure as it effectively treats all characters as Unicode characters and does not need to make a distinction.

Chapter 5

Conclusion

In this chapter we summarise the development of ATLAS and discuss the results of the evaluations that were performed. We then discuss several aspects of the future development of the system including: the minor improvements that will be made to the system after the user study evaluation, the improvements that could be made to the system if the appropriate resources were made available and finally, the areas of research that are still to be explored.

5.1 Contributions

The usefulness of both digital libraries and digital maps is unquestionable, and in this thesis we have explored the combination of the two technologies with a goal of using digital maps to enhance the content provided by digital libraries. As part of this research we designed the mAp-inTegrated digitaL librAry System or ATLAS. The contributions that are made by this research include:

- A user friendly system to enhance the content provided by digital libraries by automatically providing a map for their content.
- A real-time place recognition and disambiguation system to mark places in documents on digital maps.
- A fast and relatively accurate place name disambiguation method.

- Exploration into the usefulness and effectiveness of digital maps combined with a digital library.

- Evaluating the efficiency of several data structures used to quickly locate terms in text.

- An investigation into the effectiveness of two alternative methods for displaying text in a limited space, as well as replicating the findings of two separate research projects.

ATLAS was designed to provide additional functionality to digital libraries through the use of an automatic place recognition and disambiguation system combined with digital maps. With these two technologies ATLAS is able to automatically and efficiently scan documents for references to cities, geographical regions and countries and then mark those places in the document and also on a digital map.

ATLAS also provides functionality that allows users to interact with the places marked on the map and the places marked in the text. This is achieved through the use of place-specific menus that allow users to perform tasks such as centering the digital map viewport over the selected place, highlighting references to a place in the text or change the actual place that a place name refers to (e.g., Changing "Cambridge" to refer to Cambridge, England rather than Cambridge, New Zealand), which in turn changes the location of the marker on the map.

As well as extending the functionality of the standard document view, ATLAS is also capable of displaying the places from multiple documents at once. Differentiating between places from different documents is achieved by giving each document a colour and then displaying the places present in that document in that colour on the map. Each document's set of places can be turned on and off at will to make it easier to see the results from a particular document.

Finally, ATLAS also provides its own method for searching for documents by allowing users to query an area on a map to find documents relating to that area. This search method currently uses the text indexes already present in the digital library to perform this search by creating a text query made of all of the places in the area. Therefore it judges the relevance of documents based on factors such as search term frequency and clustering which is not ideal for spatial searching. Although we have not performed any formal research on the accuracy of this spatial searching method it appears to provide reasonably accurate results.

Integrated into ATLAS are several free and/or open source resources. The digital library used for development was Greenstone 3, which is an open source system developed at the University of Waikato in New Zealand. As ATLAS is designed to be a digital library add-on — rather than a complete system — most of the user interface is provided by Greenstone. Google Maps was used to provide the digital mapping services for the system as it is both powerful and familiar to many users. The World Gazetteer, which is a free gazetteer containing over 300,000 entries, was used to provide the geographic information for the system. The GATE ANNIE module was used to improve the accuracy of the place name recognition system by tagging each word with its part-of-speech classification. A PostGIS-enabled PostgreSQL database was used as the place information retrieval system and the spatial searching capabilities of ATLAS.

The place name recognition and disambiguation system used a customised trie data structure for place name recognition and a scoring system was used for place name disambiguation. These systems were designed to work fast enough to parse documents in real-time.

Two alternative text displaying methods were explored as a way to maximise the small amount of space allowed for the document text. The fisheye view allowed users to see more text at once by shrinking the font size in areas that the user was not focusing on. The column view arranges the text into

multiple columns as well as reducing the font size to maintain a reasonable number of characters per line.

Results from the usability testing were generally positive, however, several participants found the fisheye view difficult to use due to issues with scrolling. This finding is consistent with other research done on fisheye interfaces in different contexts. The column view was received more positively than the fisheye view with all participants able to complete the task in which it was used in the user test. Participant's comments indicated that the column view was logical to use and was easy to scroll through.

To evaluate the accuracy of the system we manually tagged five pages from Wikipedia. We then experimented with combinations of four parameters that are adjustable in the disambiguation system of ATLAS to find what combination(s) gave the best results. Upon analysis of these results it was found that the most important factor in correct place disambiguation with the ATLAS system is the removal of places that are not directly referenced in the document. Another important factor is to give the parents of places that are directly referenced in the document a small amount of the score given to their children.

Although the system has produced a high precision value it is important to realise that many documents are not as place-oriented as the Wikipedia articles used in the experiment and that, as the disambiguation scoring system relies on there being several place references in the text it will not perform as well if these references are more limited. Further testing is needed to evaluate the performance of the disambiguation system on smaller articles and to find out if the ideal combination of parameters is different for documents of this nature.

The final area of the system that was evaluated was the efficiency of the trie structure used as part of the place name recognition system. Three different structures were evaluated in terms of their memory usage and speed. Although differences between the structures was minor it was decided that the modified hybrid trie structure was the best option for ATLAS 4.3.

5.2 Future work

Although the ATLAS system has proven itself to be a useful addition to the Greenstone system there are still many aspects of it that are still under development. Features such as the spatial document search, the extended text searching functionality and the alternative text viewing methods are still in their infancy and, although they accomplish what is required of them, there are still many avenues of research yet to be explored. Other parts of the system are further developed and as a result are likely to only receive minor adjustments in future development. These include the place name recognition and disambiguation system and the system's page layout and user interface. These aspects of the system were deemed more important and therefore have been the main focus of the research.

Future research into the spatial document searching capabilities of ATLAS is likely to focus on two main areas: making spatial searching easier and more intuitive, and increasing the accuracy of the results that are returned. Researching into either of these two areas will first require research into spatially indexing documents. As described in Section 3.3.7 ATLAS makes use of the text indexes in Greenstone to perform spatial document searching. Although this method performs reasonably well, using spatial indexes rather than text indexes would allow the returned results to be more accurate. For example, if a user creates a spatial query that includes Cambridge, New Zealand then the text indexes will find documents that frequently mention "Cambridge" which is likely to be undesirable as most documents mentioning "Cambridge" will be referring to Cambridge in England. Spatial indexes will resolve this problem as it will be possible to only find documents that mention Cambridge in New Zealand.

Creating spatial indexes will also allow improvements in the user interface of the spatial document searching. For example, as discussed in Section 2 one of the features that would likely improve how intuitive the spatial searching interface is would be to allow users to actually click the countries, regions or

cities that they wish to find documents about. This is not efficiently possible without using spatial indexes as, if the user chooses to make their query region the United States for example, then the only way to search for related documents using text indexes is to create a text query that contains every place in the USA (over 25,000 places) which is impractical.

The text search view will also benefit from creating spatial indexes. Currently the text search view extension is not very effective due to the time the system requires to scan each of the documents that are returned from the text search to find what places are mentioned within them (depending on the number of the returned documents it can take several minutes). Spatial indexes will remove the need to scan these documents and therefore the map displaying the places found in the search result documents will be able to be displayed almost instantly.

One of the disadvantages of the fisheye text display method is that there is a large amount of unused whitespace above and below the focus area. In the fisheye source code editor discussed in Section 2.2.1 this whitespace was used to display information relevant to the user's current context. Using the context view for locating terms related to the current term of interest was proven to be a useful feature in the fisheye source code editor study. Taking this concept and applying it to the ATLAS fisheye view could provide some useful functionality for the users. For example, when a user selects a place name, all other lines in the text that mention that place within the document could have their font size increased (if they are in the context area), allowing the user to easily see where else in the document this particular place is mentioned.

Another area that may be explored in future development is using the disambiguation calculation performed by the ATLAS system to add metadata to documents stating what geographic areas are prominent in the document. This would be done in a similar manner to the Web-a-Where system [AHSS04]. The current disambiguation calculation effectively works out what are the main regions that are being discussed in a document so that it can effectively

disambiguate the place names, but this information could also be used for metadata.

References

[AHRW76] J. R. Anderson, E. E. Hardy, J. T. Roach, and R. E. Witmer. *A land use and land cover classification system for use with remote sensor data*. Government Printing Office, Washington, DC US Geological Survey, 1976.

[AHSS04] Einat Amitay, Nadav Har'El, Ron Sivan, and Aya Soffer. Web-a-where: geotagging web content. In *SIGIR '04: Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 273–280, New York, NY, USA, 2004. ACM.

[Bed00] Benjamin B. Bederson. Fisheye menus. In *UIST '00: Proceedings of the 13th annual ACM symposium on User interface software and technology*, pages 217–225, New York, NY, USA, 2000. ACM.

[BLH04] Patrick Baudisch, Bongshin Lee, and Libby Hanna. Fishnet, a fisheye web browser with search term popouts: a comparative evaluation with overview and linear view. In *AVI '04: Proceedings of the working conference on Advanced visual interfaces*, pages 133–140, New York, NY, USA, 2004. ACM.

[COH99] Michael G. Christel, Andreas M. Olligschlaeger, and Chang Huang. Interactive maps for a digital video library. *Multimedia Computing and Systems, International Conference on*, 1:9381, 1999.

[DK97] Mary C. Dyson and Gary J. Kipping. The legibility of screen formats: Are three columns better than one? *Computers & Graphics*, 21(6):703–712, 1997. Graphics in Electronic Printing and Publishing.

[Dys04] Mary Dyson. How physical text layout affects reading from screen. *Behaviour and Information Technology*, 23:377–393(17), 2004.

[Fur86] G. W. Furnas. Generalized fisheye views. *SIGCHI Bull.*, 17(4):16–23, 1986.

[Fur91] George W. Furnas. New graphical reasoning models for understanding graphical interfaces. In *CHI '91: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 71–78, New York, NY, USA, 1991. ACM.

[Fur99] G. W. Furnas. The fisheye view: a new look at structured files. *Readings in information visualization: using vision to think*, pages 312–330, 1999.

[GCY92] William A. Gale, Kenneth W. Church, and David Yarowsky. One sense per discourse. In *HLT '91: Proceedings of the workshop on Speech and Natural Language*, pages 233–237, Morristown, NJ, USA, 1992. Association for Computational Linguistics.

[Gra93] R. Scott Grabinger. Computer screen designs: Viewer judgments. *Educational Technology Research and Development*, 41(2):35–73, 1993.

[HAHR93] Xuedong Huang, Fileno Alleva, Mei-Yuh Hwang, and Ronald Rosenfeld. An overview of the sphinx-ii speech recognition system. In *HLT '93: Proceedings of the workshop on Human Language Technology*, pages 81–86, Morristown, NJ, USA, 1993. Association for Computational Linguistics.

[JH06] Mikkel Rønne Jakobsen and Kasper Hornbæk. Evaluating a fisheye view of source code. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 377–386, New York, NY, USA, 2006. ACM.

[JH09] Mikkel Rønne Jakobsen and Kasper Hornbæk. Fisheyes in the field: using method triangulation to study the adoption and use of a source code visualization. In *CHI '09: Proceedings of the 27th international conference on Human factors in computing systems*, pages 1579–1588, New York, NY, USA, 2009. ACM.

[Lar96] Ray R. Larson. Geographic information retrieval and spatial browsing. In *Geographic information systems and libraries: patrons, maps, and spatial information [papers presented at the 1995 Clinic on Library Applications of Data Processing, April 10-12, 1995]*, pages 81–124. Graduate School of Library and Information Science, University of Illinois at Urbana-Champaign, 1996.

[MMG99] Andrei Mikheev, Marc Moens, and Claire Grover. Named entity recognition without gazetteers. In *Proceedings of the ninth conference on European chapter of the Association for Computational*

Linguistics, pages 1–8, Morristown, NJ, USA, 1999. Association for Computational Linguistics.

[Nie94] Jakob Nielsen. Enhancing the explanatory power of usability heuristics. In *CHI '94: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 152–158, New York, NY, USA, 1994. ACM.

[NL06] Jakob Nielsen and Hoa Loranger. *Prioritizing Web Usability*. New Riders, Berkeley, California, 2006.

[PKS+06] Bruno Pouliquen, Marco Kimler, Ralf Steinberger, Camelia Ignat, Tamara Oellinger, Ken Blackler, Flavio Fuart, Wajdi Zaghouani, Anna Widiger, Ann-Charlotte Forslund, and Clive Best. Geocoding multilingual texts: Recognition, disambiguation and visualisation. *CoRR*, abs/cs/0609065, 2006.

[PSIDG04] Bruno Pouliquen, Ralf Steinberger, Camelia Ignat, and Tom De Groeve. Geographical information recognition and visualization in texts written in various languages. In *SAC '04: Proceedings of the 2004 ACM symposium on Applied computing*, pages 1051–1058, New York, NY, USA, 2004. ACM.

[RM93] George G. Robertson and Jock D. Mackinlay. The document lens. In *UIST '93: Proceedings of the 6th annual ACM symposium on User interface software and technology*, pages 101–108, New York, NY, USA, 1993. ACM.

[SB92] Manojit Sarkar and Marc H. Brown. Graphical fisheye views of graphs. In *CHI '92: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 83–91, New York, NY, USA, 1992. ACM.

[SC01] David A. Smith and Gregory Crane. Disambiguating geographic names in a historical digital library. In *ECDL '01: Proceedings of the 5th European Conference on Research and Advanced Technology for Digital Libraries*, pages 127–136, London, UK, 2001. Springer-Verlag.

[SWRG02] Bongwon Suh, Allison Woodruff, Ruth Rosenholtz, and Alyssa Glass. Popout prism: adding perceptual principles to overview+detail document interfaces. In *CHI '02: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 251–258, New York, NY, USA, 2002. ACM.

[WP94] Allison Gyle Woodruff and Christian Plaunt. Gipsy: automated geographic indexing of text documents. *J. Am. Soc. Inf. Sci.*, 45(9):645–655, 1994.

[ZWS+05] Wenbo Zong, Dan Wu, Aixin Sun, Ee-Peng Lim, and Dion Hoe-Lian Goh. On assigning place names to geography related web pages. In *JCDL '05: Proceedings of the 5th ACM/IEEE-CS joint conference on Digital libraries*, pages 354–362, New York, NY, USA, 2005. ACM.

Appendix A

User study material

This Appendix contains material relating to the user study as described in 4.1, including:

- The Task Sheet given to participants.
- The Participant Information Sheet.
- The Participant Research Consent Form.
- The letter from the School of Computing and Mathematical Sciences Ethics Committee granting permission for the study.

Task Sheet

Task 1

In the MGPP demo collection use the browsing features to navigate to the document called "Farming snails 1: Learning about snails; Building a pen; Food and shelter plants".

a. What is the only highlighted place name in the text?

b. Put your mouse cursor over the place name and click "Choose correct place", how many other places have this place name?

c. In what country has the marker been placed on the map?

Task 2

Navigate back to the document selection page and choose the document called "Butterfly Farming in Papua New Guinea". Go to the section labelled "5 Application to Other Nations".

a. Costa Rica is highlighted near the top of the page, use the tools available to locate Costa Rica. What two countries border Costa Rica?

b. Near the bottom of the document San Francisco is highlighted. The scoring system has made a mistake and scored San Francisco in Costa Rica higher than San Francisco in the United States of America. Use the "Choose correct place" menu option to choose the correct San Francisco. What state is San Francisco in? Is San Francisco on the east or west side of the USA?

c. Locate Australia on the map. Click on it's marker (located roughly at the centre). What is the population shown in the bubble? What are the coordinates shown?

Task 3

Navigate back to the document selection page and choose the document called "Little Known Asian Animals With a Promising Economic Future". Go to the section labelled "Introduction".

a. Indonesia is highlighted several times near the top of the page. Bring up the menu for one of them and choose "Highlight this place in text". How many times does "Indonesia" appear in the text?

Task 4

Navigate back to the home page of the MGPP Demo collection and choose "Spatial Search".

a. Draw a box around Tasmania (below Australia) and then click "Perform Search". What are the first three documents listed in the results?

Task 5

Navigate back to the home page of the MGPP Demo collection and choose "Text Search".

a. Enter "Water" into the query area and click the search button. Name a country that has a reference in the document "35 bees".

Task 6

Navigate back to the document selection page and choose the document called "The Courier - N°158 - July - August 1996 Dossier Communication and the media - Country report Cape Verde". Click "Meeting Point" in the table of contents and then click "Robert Ménard, Director of 'Reporters sans frontières".

a. Choose Fisheye from the text view selector at the top of the document. Move up and down the document to understand how it works. Using the fisheye view give a rough estimate of the number of highlighted place names in the text.

b. Choose Column View from the text view selector at the top of the document. Browse for a while to understand how it works. What are the names of three of places in the document?

Task 7

Please complete the following survey:

a. Do you think that having a map to show where places are located is helpful when reading documents? Why?

b. How easy was it to use the features available to find the place you were looking for on the map?

c. How easy did you find the fisheye view to use compared to viewing the text normally? What did you like/dislike about it?

d. How easy did you find the column view to use compared to viewing the text normally? What did you like/dislike about it?

e. What improvements would you suggest could be made to the system?

Participant Information Sheet

Ethics Committee, School of Computing and Mathematical Sciences

Project Title
Integrating Interactive Digital Maps into a Digital Library

Purpose
This research is designed to test the design and usefulness of the digital-map-integrated digital library program created by the researcher. This research will be used to evaluate the software in the Master's thesis of the researcher.

What is this research project about?
This research is about exploring the combination of digital library software and digital mapping software. It aims to enhance digital library content by allowing users to better visualize the places mentioned in the digital documents and to perform tasks such as:

- Using spatial searching to find documents relating to a particular area.
- Use digital-map-enhanced searching to see at a glance the places related to each search result.

What will you have to do and how long will it take?
The researcher will require you to complete a series of tasks using the program that the researcher has designed. These tasks are designed to test all aspects of it's functionality and should take about 10 to 15 minutes to complete. You may be video recorded while you complete the tasks. You will then be asked to complete a short survey questionnaire that will ask your opinions on what you have tested, this is aimed at discovering how effective the program is at doing what it is designed to do and how useful it is. The survey should take about 5 to 10 minutes to complete.

What will happen to the information collected?
The information collected will be used by the researcher to evaluate the software as part of the Master's in Science thesis being written by the researcher. It is possible that articles and presentations may be the outcome of the research. Only the researcher and the supervisor of the researcher will be privy to the notes, documents and recordings. Afterwards, notes, documents and recordings will be transferred to the Waikato SCMS Data Archives to be destroyed on the 1st of March 2020. No participants will be named in the publications and every effort will be made to disguise their identity.

Declaration to participants
If you take part in the study, you have the right to:

- Refuse to perform a task.
- Refuse to answer any questions in the questionnaire.
- Ask any further questions about the study that occurs to you during your participation.

· Be given access to a summary of findings from the study when it is concluded.

Who's responsible?
If you have any questions or concerns about the project, either now or in the future, please feel free to contact either:

Researcher:
Samuel John McIntosh
Address: 18 Morrinsville Road, Hillcrest, Hamilton
Mobile: (027) 343 3587
Email: sjm84@waikato.ac.nz

Supervisor:
David Bainbridge
Phone: (07) 838 4407
Email: davidb@cs.waikato.ac.nz
Address: Department of Computer Science
University of Waikato
Private Bag 3105, Hamilton

Research Consent Form

Ethics Committee, School of Computing and Mathematical Sciences

Integrating Interactive Maps into a Digital Library

Consent Form for Participants

I have read the **Participant Information Sheet** for this study and have had the details of the study explained to me. My questions about the study have been answered to my satisfaction, and I understand that I may ask further questions at any time.

I also understand that I am free to withdraw from the study at any time, or to decline to answer any particular questions or complete any tasks in the study. I understand I can withdraw any information I have provided up until the researcher has commenced analysis on my data. I agree to provide information to the researchers under the conditions of confidentiality set out on the **Participant Information Sheet**.

I agree to participate in this study under the conditions set out in the **Participant Information Sheet**.

Signed: _____

Name: _____

Date: _____

Additional Consent as Required

I agree / do not agree to being video recorded while completing the given tasks.

Signed: _____

Name: _____

Date: _____

Researcher's Name and contact information:
Samuel John McIntosh
Address: 18 Morrinsville Road, Hillcrest, Hamilton
Mobile: (027) 343 3587
Email: sjm84@waikato.ac.nz

Supervisor's Name and contact information:
David Bainbridge
Phone: (07) 838 4407
Email: davidb@cs.waikato.ac.nz
Address: Department of Computer Science, University of Waikato, Private Bag 3105, Hamilton

Computing and Mathematical Sciences
Pōrohiko me ngā Pūtaiao Pāngarau
The University of Waikato
Private Bag 3105
Hamilton
New Zealand

Phone +64 7 838 4021
www.scms.waikato.ac.nz

THE UNIVERSITY OF WAIKATO
Te Whare Wānanga o Waikato

21 January 2010

COPY

Samuel McIntosh
C/- Department of Computer Science
THE UNIVERSITY OF WAIKATO

Dear Samuel

Request for approval to perform a user test involving human participants

I have considered your request to perform an experiment to test the overall usability of the digital-map-integrated digital library program and survey questionnaire for your research project "Integrating Interactive Digital Maps into a Digital Library".

The procedure described in your request is acceptable.

I note your statements that participants will not be named in the final report or any other publications arising from this research and every effort will be made to disguise their identity. The information collected will be kept securely in the SCMS Data Archive.

The research participants' information sheet and consent form meets the requirements of the University's human research ethics policies and procedures.

I therefore approve your application to perform the experiment.

Yours sincerely,

Dave Nichols
Human Research Ethics Committee
School of Computing and Mathematical Sciences