

May 2013

Social Media Visualization Tools and Techniques

Amalgamating Social & Factual data to provide a
comprehensive outlook on any geo-location.

Final Report



Steven Oakley
C1009109

Supervisor: Professor N.J. Avis

Moderator: Dr X. Sun

Abstract

My motivation for embarking on a project of this nature stems from a personal interest in finding new and unconventional methods of finding or creating new information. The interim report abstract outlined the overall philosophical framework of the project, whereby I stated how I planned to combine factual and social data from a variety of sources into elegantly constructed data sets that would help create new knowledge. I also explained that whilst there are numerous methods of collecting data similar to that which I utilise, there are currently few applications that effectively combine large quantities of this information in real time, with respect to a particular geo-location.

Data prioritisation is a fundamental principle of any project that handles large quantities of data, and so I made filtering and program efficiency a priority when developing the project from the very beginning. Having an effective means of displaying generated information to the end user is also critical, so as to ensure that the operator is not overwhelmed with unnecessary data. As such I have ensured that a significant amount of time was spent on the user interface design, which went through a series of revisions and improvements throughout the development lifecycle. Additionally, I understood the importance of ensuring that data could be both imported into and exported from the system in a variety of commonly-used formats, and so I certified through appropriate research that the formats I eventually chose were appropriate.

I believe that a system of this nature could be used in a multitude of widely varying situations, which should be made apparent in the report. The final recorded demonstrations detailed at the end of the report give a representation of how I personally feel the system could be used. I have however also received a number of suggestions from third parties (both individuals and organisations to whom I have shown the project), which indicates to me that so long as a person fully understands the capabilities of the system, along with any potential future capabilities, that there is a great range in the number of use-case possibilities.

Acknowledgements

There are a number of people without whom this project would not have been possible. I would like to thank my supervisor Professor N.J. Avis for his continual help and guidance throughout both semesters. I would also like to thank Professor O.F. Rana and Nicholas Horne, who helped in the very early stages of the project and provided me with a basis on which to develop my ideas and program.

Additionally, it is necessary to thank Weather Underground (a subsidiary of The Weather Channel), for their support and generous sponsorship of the project by allowing me high level access to their API and other geological information free of charge. Finally, I greatly value the constructive feedback I received from third parties, particularly regarding comments encouraging me to redesign the user interface – remarks which in hindsight I completely agree with.

Contents

Abstract	1
Acknowledgements	2
Contents	3
Introduction	5
Specification.....	7
Data Flow Overview	8
Security.....	10
Saving & Loading Files	12
Design	13
Main Interface	13
Tweet of Interest Interface	17
Design Modifications.....	22
<i>Main GUI Improvements</i>	<i>22</i>
<i>Tweet of Interest GUI Improvements</i>	<i>23</i>
<i>Miscellaneous Design Additions & Improvements</i>	<i>24</i>
Implementation	25
Main GUI	26
Analyse Tweets Function.....	27
Tweets of Interest	31
Miscellaneous Features.....	34
Security.....	38
Difficulties Encountered.....	41
Demonstration & Evaluation	43
Large-scale demonstration: Political Analysis.....	43
Medium-scale demonstration: Event Monitoring	50
Real-world Usage	59
Future Work	60
Conclusions	63
Reflections on Learning	64

Glossary	65
References	67
Bibliography.....	69
Appendix.....	71
Figure 1: Interim Report Future Development	71
Figure 2: Term 1 & 2 Gantt Charts	73
Figure 3: Interim Report System Overview	74

Introduction

The aim of the project, as outlined in the initial plan, originally consisted of collecting and analysing large amounts of social data and comparing this with factual data such as weather reports in order to produce a system with similar traits to a visual ‘early warning system’. The interim report explained in detail the specifics of what the system was capable of at the time, and what I believed would be possible in the future. With the support of my supervisor Professor N.J. Avis, I explained how the project was naturally beginning to move away from explicit social data ‘visualisation’, and was instead becoming more comparable to a ‘social data toolbox’, whereby from a single Tweet input, varying types of information could be created. Since the interim report I have continued to develop the project with this mind-set, and as such many new features have been implemented in addition to those originally proposed. I believe that this modified vision for the project has greatly enhanced the final capabilities of the system, and has turned the project into a system that I believe has genuine value in real-world situations.

The intended end-user has also changed throughout the lifetime of the project. Initially it would have been reasonable to assume that the user would have been a professional in their field of expertise (for example a meteorologist for a weather warning system), and so as a result I believed I could produce a system with complex and focused elements such as the GUI or the format of the final outputs. However, as the project progressed and evolved into the toolbox application it became clear that this method of thinking would not be sufficient, and as such in the later stages of development I took the decision to redesign the entire GUI system, along with making numerous programming alterations to the software in order to make the system accessible to new users who may not be familiar with this type of software. Further details as to the exact modifications I made are found later in the ‘Design’ and ‘Implementation’ sections.

Whilst both the focus of the project and the targeted end user have evolved since the project outset, the overall theme of the project has not. There is still a very strong emphasis on appropriately, efficiently and accurately filtering and analysing large amounts of data in order to produce information that is capable of producing new and usable knowledge when used appropriately. The system is capable of providing information relating to the weather, news, photos, textual sentiment, traffic/map reports, and is even applicable (and has actually been used) for event monitoring and political purposes. I believe that this clearly demonstrates the wide-ranging possibilities for the system, and genuinely believe that it has a tangible use even outside of academia.

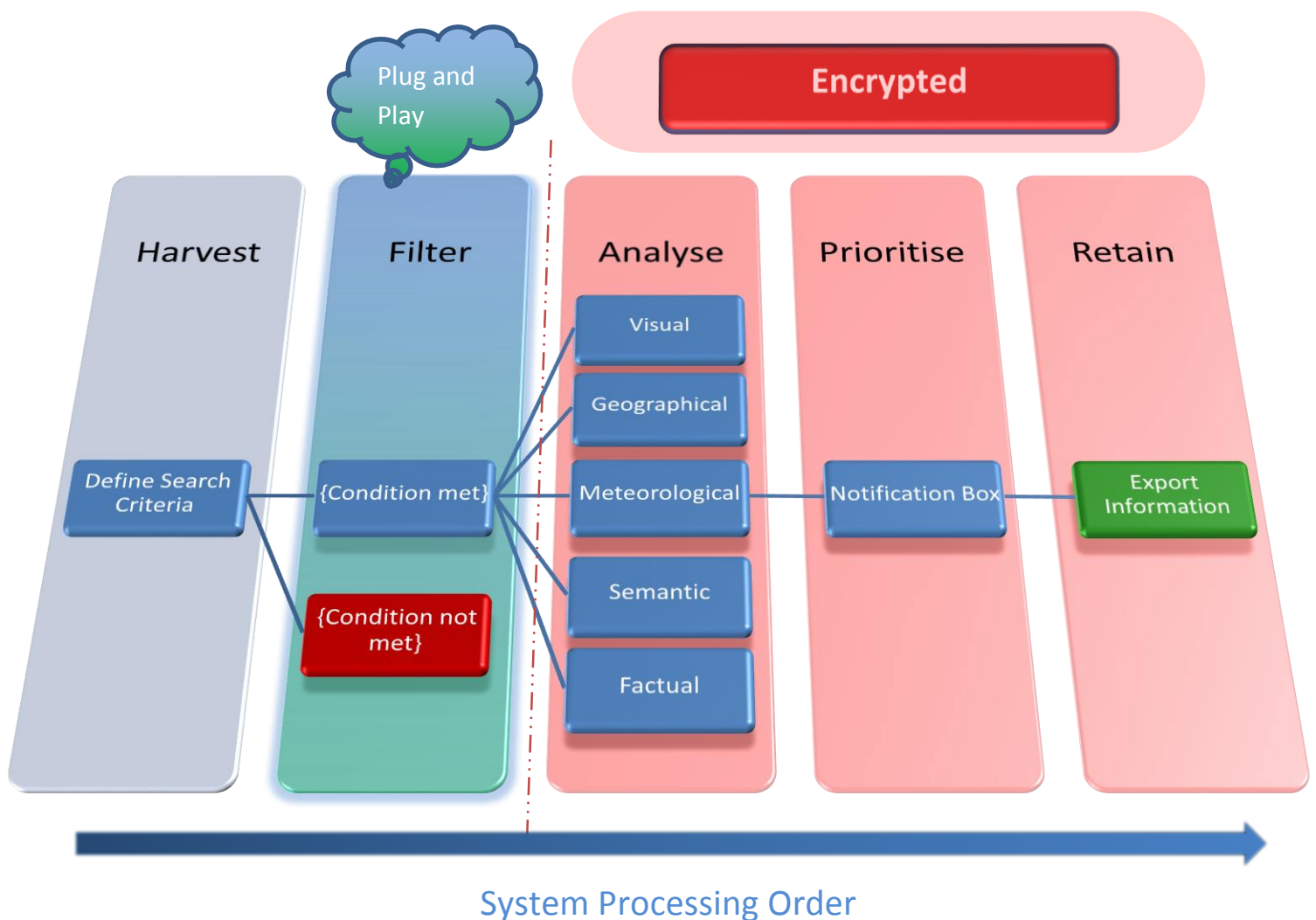
I outlined ten areas for improvement from the status of the system in the interim report (*see appendix figure 1*). Every one of these key deliverables has been addressed and successfully completed, and through regularly discussing the development of the system with Professor

N.J. Avis, many further refinements & additions have been made. This is in-line with the statements and Gantt charts presented in the interim report (*appendix figure 2*), whereby I specified that this term would I focus on system refinements and further analysis of currently-collected data.

Specification

The required final specification and program features have changed throughout the project, and as such the design of the system, both at the higher user-level and at the lower code-level, have also evolved. In this section I will discuss the lifecycle of the system - detailing matters such as the data-flow within the project, high-level descriptions of algorithms used and the overall design philosophy. A growing theme throughout the project life has been on creating a modular system, capable of having a 'plug-and-play' functionality that would ensure the product would be applicable to as many different subject-areas as possible. I will attempt to illustrate how I have achieved this, and will explain in much further detail particular sections of code of interest, later in the 'Implementation' section.

Figure 3 in the appendix contains the descriptive outline of how I felt the project could be broken down into at the time of the interim report. Below is the result of an update to the diagram, now representing the current system state. Whilst there have been functional additions to the 'Analyse' stage, and a security enhancement to the later stages collectively, I feel that the major philosophical change is in the modular nature of the 'Filter' stage.

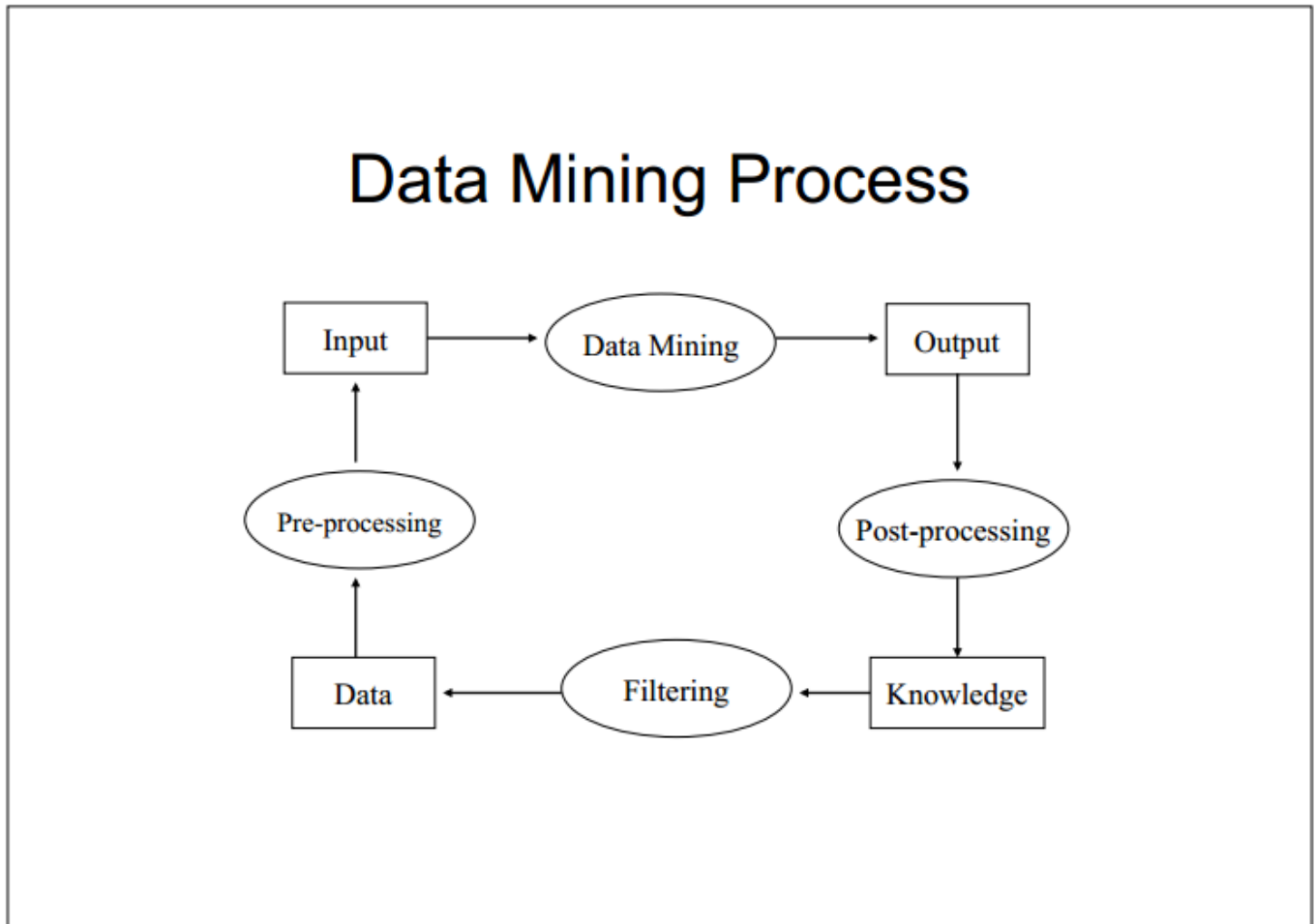


Data Flow Overview

To begin with, all data used within the system must have been harvested or 'mined' from Twitter. I achieve this by making use of a modified version of previous year student Nicholas Horne's Twitter harvesting tool, which I use with his, his tutor Professor O.F. Rana and my tutor Professor N.J. Avis's permission. This application uses the Twitter API Stream to collect Tweets and stores them in a local database – further details of this can be found in the interim report. The illustration below helps to provide an overview of how data flows through the system.



To complement the above information, I believe that the following extract from one of Professor Alun Preece's lectures in his third year module 'Knowledge Management' ^{Reference 1}, helps to demonstrate the general method of thinking that systems similar to mine often seem to possess.



In the same lecture, data mining is defined as being “the non-trivial extraction of previously unknown and potentially useful patterns (i.e. information) from data”. As is apparent from the above diagram, data mining can be viewed as being a cyclic process that contributes to generating knowledge, and whilst my project appears at first to flow purely from data mining to filtering to processing to analysing, I wanted to try to remain true to the above cyclic principle. This was ultimately the reason I began to seriously consider the importance of having the ability to export and load data, which is explained in greater detail later in the report. This feature has allowed me to analyse much larger sets of data, evidence of which is covered later in the ‘Demonstration & Evaluation’ section, and so I feel that I do remain faithful to this cyclic process.

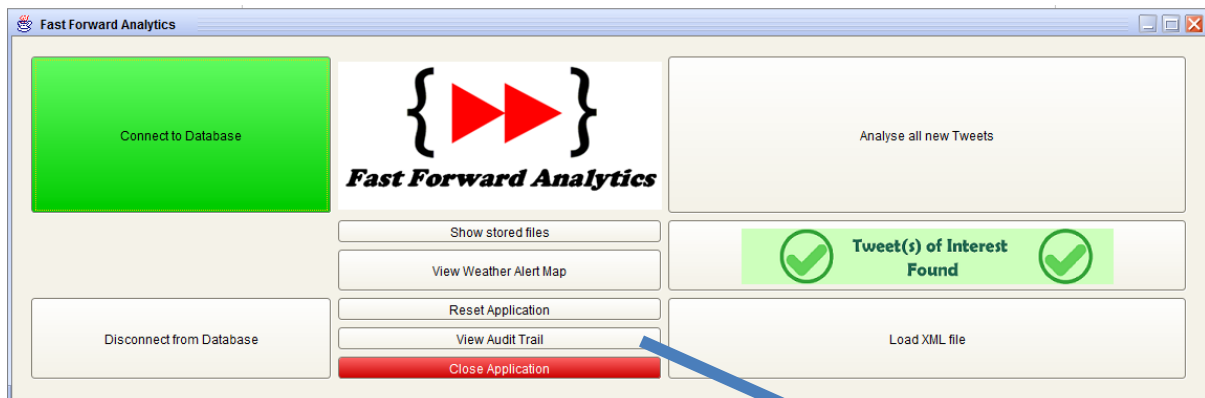
Security

As the project was progressing towards having the toolbox functionality, Professor N.J. Avis spoke with me about what had been discussed at his meeting with the metropolitan police force, involving systems of a similar nature to what mine was becoming. They were very interested in having a type of functionality where large amounts of social data could be analysed quickly, and particular Tweets of interest brought to a user's attention so that a more informed (human) decision could be made. The example provided was that if there were for instance 10 different protests taking place in London, then the system could help determine which protest groups were more likely to require a greater police presence through utilizing features such as sentiment analysis, immediate access to live cameras and human inspection of highlighted information. Additionally, a robust and accurate audit trail would be necessary to ensure that any potentially controversial decisions made with the help of the software could be backed up with evidence of what information an operator had access to at the precise point a decision was made.

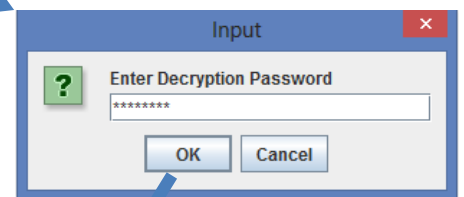
This inspired me to implement a complete and encrypted audit trail feature that would provide the necessary functionality for a use case such as this. Detail on how exactly this was implemented is documented later in the report, however I believe that I have met this requirement fully, as the table below represents all events recorded by the system that is stored in an encrypted file, only accessible with a password.

Action	Information Stored	Information Stored	Information Stored	Information Stored
'Analyse all new Tweets' button clicked	Timestamp			
File loaded into application	Timestamp	Filename	File location	
Tweet of Interest found	Timestamp	Tweet ID	Tweet geographical location	
'Click for more Information' button clicked	Timestamp	Tweet ID	Tweet geographical location	
Webcams requested	Timestamp	Webcam title/description	Web URL address of webcam source	Frame closure
Satellite Imagery requested	Timestamp	Web URL address of imagery source	Frame closure	
Mapping Imagery requested	Timestamp	Web URL address of both	Frame closure	

		Google and Bing maps		
Flickr Photo requested	Timestamp	Web URL address of Flickr Photo		
News story requested	Timestamp	News headline	Web URL address of news article	Frame closure
User note added	Timestamp	Note contents		
Backup made	Timestamp	Backup name	Backup location	MD5 checksum of backup
European Weather information requested	Timestamp	Web address of image	Frame closure	
Access to Audit trail attempted	Timestamp	Password used		
System Reset	Timestamp			
Application Closed	Timestamp			



Process of accessing the encrypted audit trail file.



AuditTrailDecrypted - Notepad

```

File Edit Format View Help
2012-12-28 22:43:26.787 - European Weather Alert Image viewed. Source available at: http://icons-sf.wunderground.com/data/640x480/2xeu_severe.gif
2012-12-28 22:43:32.55 - Frame closed : European Weather Alerts Map
2012-12-28 22:43:36.605 - Tweet Analysis Initiated
2012-12-28 22:44:07.784 - New Tweet - Detailed information for a Tweet originating from Bristol with Tweet ID: 284791581071011840 accessed
2012-12-28 22:44:20.667 - 284791581071011840 - Satellite Imagery viewed. Source available at: http://api.wunderground.com/api/205f2bb905cf8536/an:
2012-12-28 22:44:22.784 - Frame closed : Satellite Imagery
2012-12-28 22:44:40.598 - New Tweet - Detailed information for a Tweet originating from Bridgend with Tweet ID: 284791611202891776 accessed
2012-12-28 22:44:43.251 - 284791611202891776 - Map images: http://dev.virtualearth.net/REST/v1/Imagery/Map/Road/51.669,-4.0163/12?pp=47.6156352,-1
2012-12-28 22:44:45.466 - Frame closed : Bing & Google Maps
2012-12-28 22:44:48.237 - 284786326950256640 - Flickr photo: http://farm9.staticflickr.com/8499/8319794692_254704fb08.jpg accessed
2012-12-28 22:44:52.786 - Frame closed : Flickr Photos
2012-12-28 22:44:57.865 - 284791611202891776 - Note added: "this is a note"
2012-12-28 22:45:04.97 - 284791611202891776 - Backup: Bridgend.txt successfully exported
MD5 Hash of Bridgend_backup.txt = 97a87669c835ef631d19a498ae147354
2012-12-28 22:49:52.407 - System Reset

```

Saving & Loading Files

Having the ability to both save and load files into the system has greatly improved the potential widespread appeal of the application. Not only is this a feature that would be required if it was to be used by an organisation such as the police for aforementioned reasons, but it also opens up many new possibilities. One use case I have envisioned for the system would be that of a training simulator or test-centre device for candidates applying for a position within a company. To test particular candidate competencies, a number of premade test cases could be developed by the operator and then subsequently loaded onto all candidates machines to see how the users react and prioritise data cases that are presented to them. This would of course also make extensive use of the previously described auditing system, whereby user actions could be analysed in extensive detail if deemed necessary.

I have provided the user with two means of saving an output file – either as a plain text file, or as an xml file. Initially I developed the plain text saving method, and while this was functional, it is not a format that is used regularly as a saved file format due to a lack of advanced formatting methods. My meeting & presentation with the Social Science COSMOS team on 30th January 2013 (*presentation uploaded to 'additional files' project area*) lead to comments reiterating this from Dr Pete Burnap who suggested that another choice of format would be more suitable for interoperability with other applications. As such, I began implementing the xml saving option, which I was able to develop relatively quickly due to my experience from reading xml files previously in the project.

Whilst it is possible to view all information by directly viewing the exported .txt and .xml files, I decided that it would be beneficial to allow a user to directly load files into the system and have the GUI display the information exactly as it would have originally been viewed. This continued to advance the project toward having the specification required by the police instance detailed previously of being able to justify decisions made by showing exactly what information was available at the time of saving (and verifiable by the recorded MD5 checksum of the file, saved in the encrypted Audit Trail file). This functionality also allowed me to speed up other parts of the development process as I could export a good example case which made use of all features, and simply load this file into the system each time to test new features/revisions, as opposed to having to search for new Tweets each time and populate the database. This reduced an arbitrary system development overhead, which allowed future development and bug fixes to proceed at an improved rate.

Design

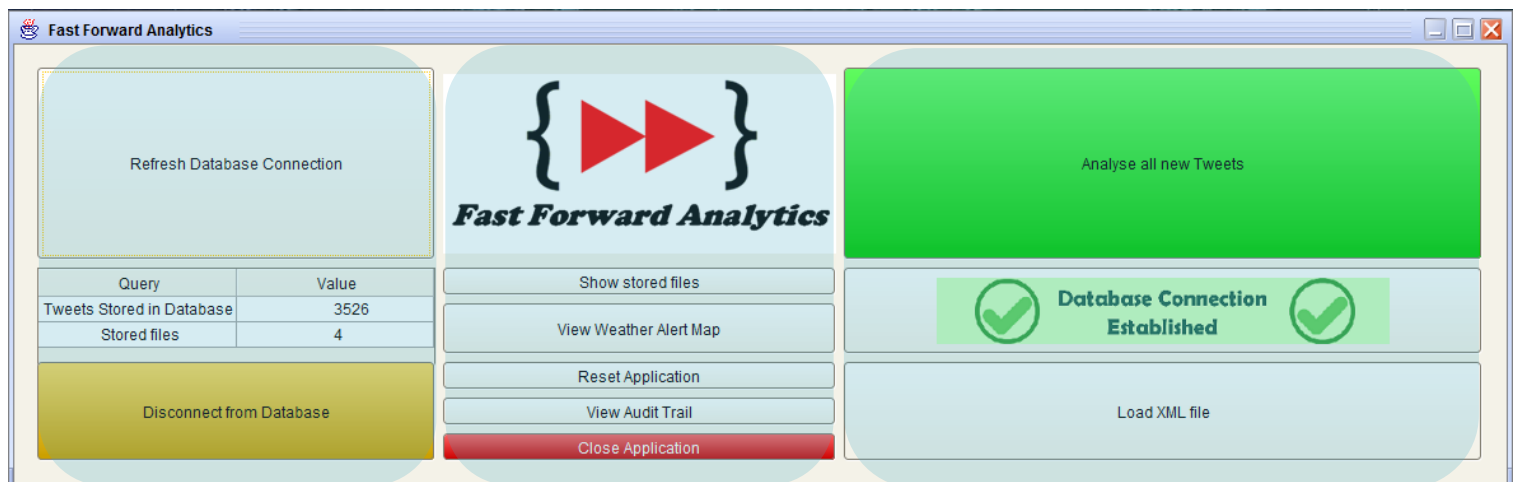
The user interface for the system has continued to develop in line with the evolving use case scenarios, and in this section I will identify these changes and provide justification for the modifications.

The increasing complexity of the software, coupled with the changing targeted end-user, made it necessary to review the current user interface and attempt to implement a new and more intuitive GUI that could provide more detailed user feedback. Additionally, as the project was progressing towards becoming a product in its own right rather than an extension to a previous project aimed solely at solving very specific academic questions, I felt it appropriate to create a 'brand name' and accompanying logo. The logo below is displayed prominently on the main user interface, and is also the executable file's 'splash screen' image that is displayed when the application is first launched.



Main Interface



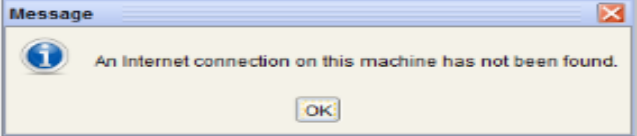


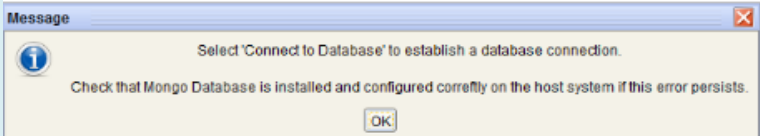


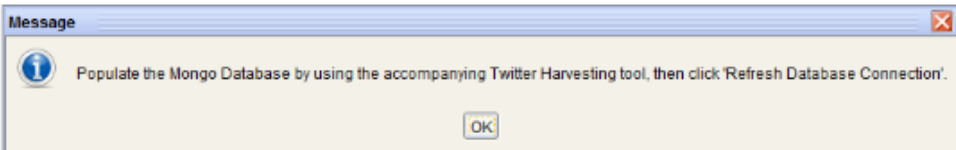






I trialled various forms of GUI layouts using a multitude of different Java layout solutions, before I settled on the final solution displayed on the following page that makes use of the more advanced Java GridBagLayout options. I feel that this interface is a significant improvement on the original, offering a substantial number of improvements such as improved user feedback & help regarding the current system state, a logical button layout in three discreet sections with differing button sizes depending on importance, and improved coding logic to help minimise the risk of user error causing a problem to system stability. Further details of these coding features are present in the Implementation section.



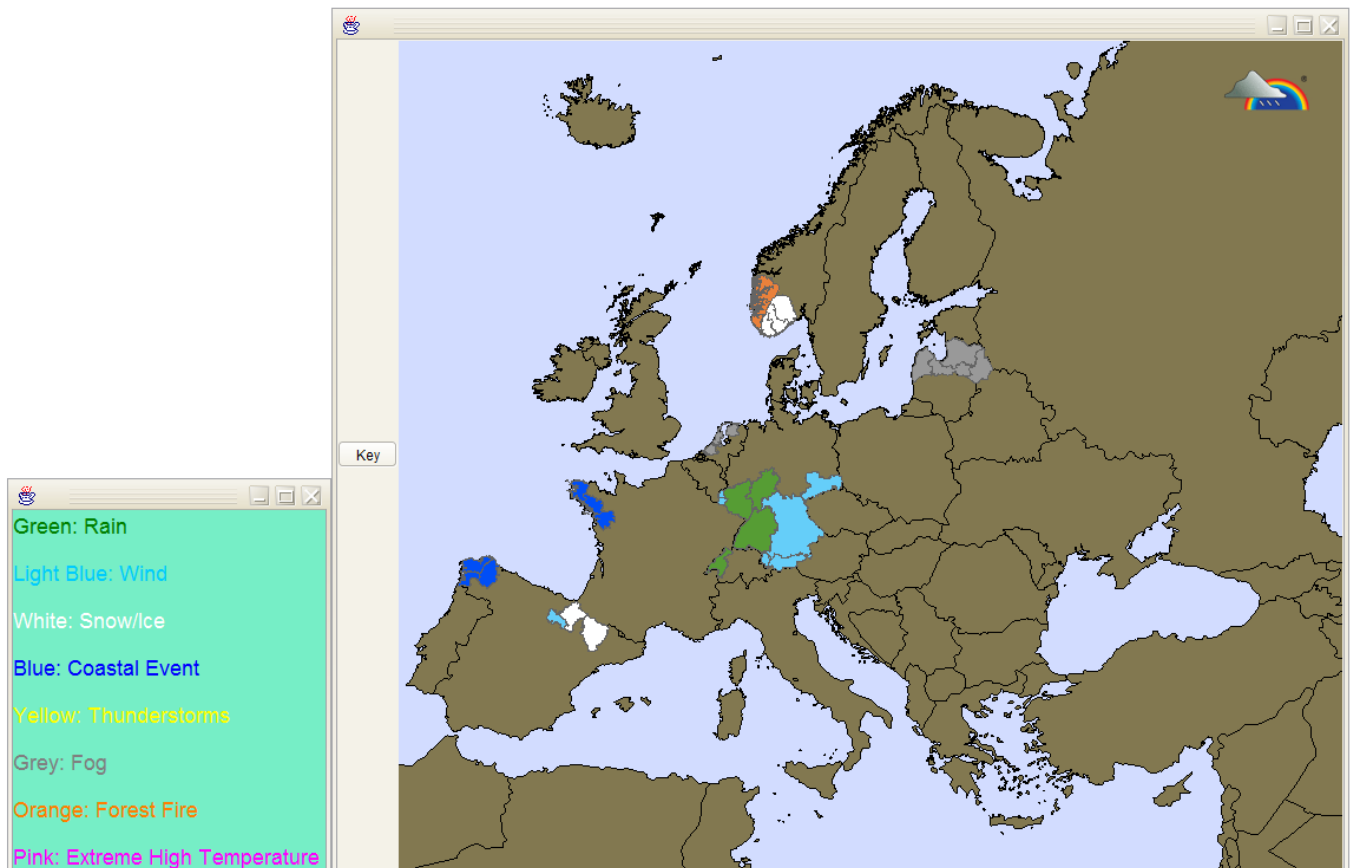
Database Options	Pre/Post Analysis Options	Start / Load Analysis
Connect / Reconnect to database.	Logo – could be a clickable link image to possible future website.	Begins searching through database for Tweets of Interest.
Dynamic table information – only shown if connected to database.	Opens operating system default explorer to saved files location.	Interactive & Dynamic status button.
Disconnect from database.	Loads and displays a weather alert map from Wunderground.com.	Opens operating system file chooser in saved files location.
	Prompts user for password in order to view audit trail.	
	Closes Application.	

Above is a table describing the GUI features in sequential order for each of the three button columns.

The image on the previous page showed the GUI when the system is fully operational, as made clear by the green 'Database Connection Established' button with two tick symbols in the right hand column. This status button is dynamic and will update automatically as necessary to quickly alert the user to any detected problems with the system. Additionally, clicking on the button will create a popup window giving further detail, unique for each different system status. The table below show the various status possibilities and their accompanying additional information windows.

 No Internet Connection 	
 No Database Connection 	
 Database Empty 	
 Database Connection Established 	<p>No dialogue box necessary</p>
 Analysing Tweets 	<p>No dialogue box necessary</p>
 Tweet(s) of Interest Found 	<p>No dialogue box necessary</p>

The button 'View Weather Alert Map' in the middle column of the main GUI will load and display a snapshot of weather alerts in Europe, as recorded by Meteoalarm ^{Reference 1}. According to the UK Met Office, Meteoalarm is *"a co-operative initiative involving more than 20 European national weather services. The web site has been developed by Eumetnet, the Network of European Meteorological Services, and is supported by the World Meteorological Organization (WMO)."* ^{Reference 2} The user interface generated is displayed below:



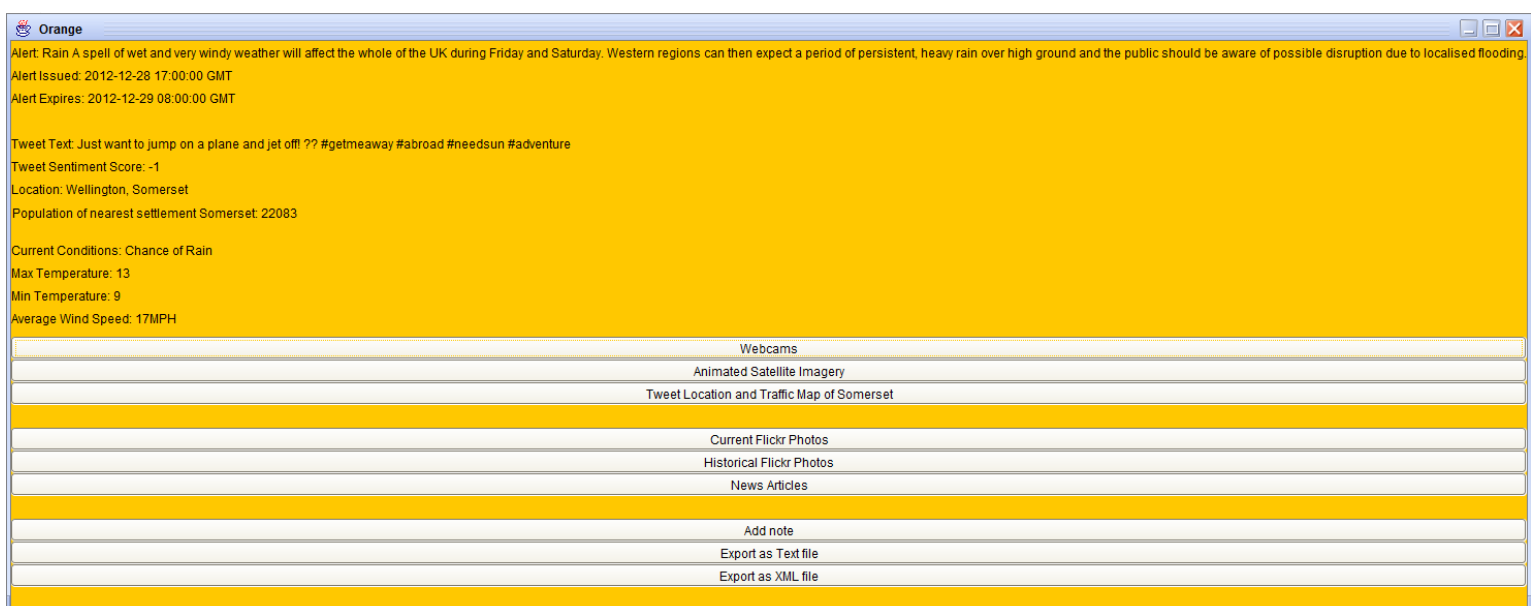
The key on the left of the above image is only displayed when the user clicks on the respective 'Key' button on the map image itself. One challenge that I encountered when designing this aspect of the interface was the choice of background shade for the 'Key' window. I tried many different variations of colours in order to find one in which the correct colours could be used to represent the alert, but still ensured that all text remained legible. I finally settled on the above solution (background colour: "Aquamarine 2" – RGB [118,238,198]) after consulting the 'W3C Guidelines for Colour Readability' document ^{Reference 3}.

Tweet of Interest Interface

The interface shown to the right is what is generated when a Tweet of Interest is discovered. As the current filter focuses on the weather, the colour of the GUI box matches the Tweet location's respective Alert rating (as determined by the Met Office or other country

equivalent), which is also displayed in the window header. The four information points below display the specific alert, the Tweet text, the overall sentiment of the Tweet and the originating Tweet location. The colour of the 'Tweet Sentiment Score' text changes as appropriate – red for a negative value, black for neutral and green for positive.

Selecting 'Click for more Information' will display the interface shown below, and close the above basic information window. This advanced window displays all information found in the basic one, in addition to further alert, location & weather information, along with a multitude of further analysis buttons. The colour of the interface is again defined by the alert rating colour.



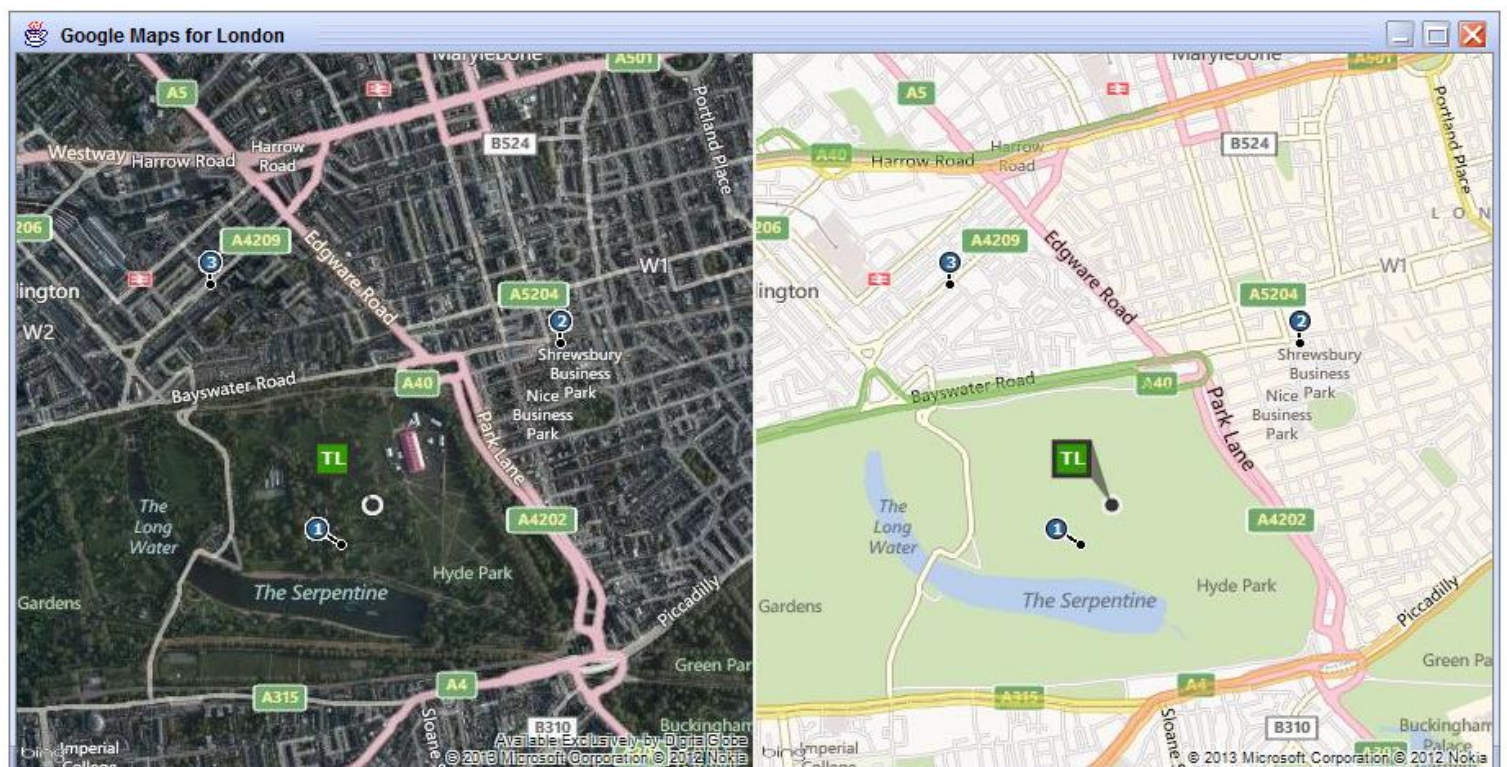
On the following page is a descriptive list of what each of the above buttons do, along with accompanying images of the results.

1. Webcams

This will display up to three of the nearest webcams found in relation to the initial Tweet location. It will also display any assigned webcam titles, such as in the case below of 'London, Hyde Park', 'Oxford Street Webcam, London', and 'Paddington and Hyde Park, London'.

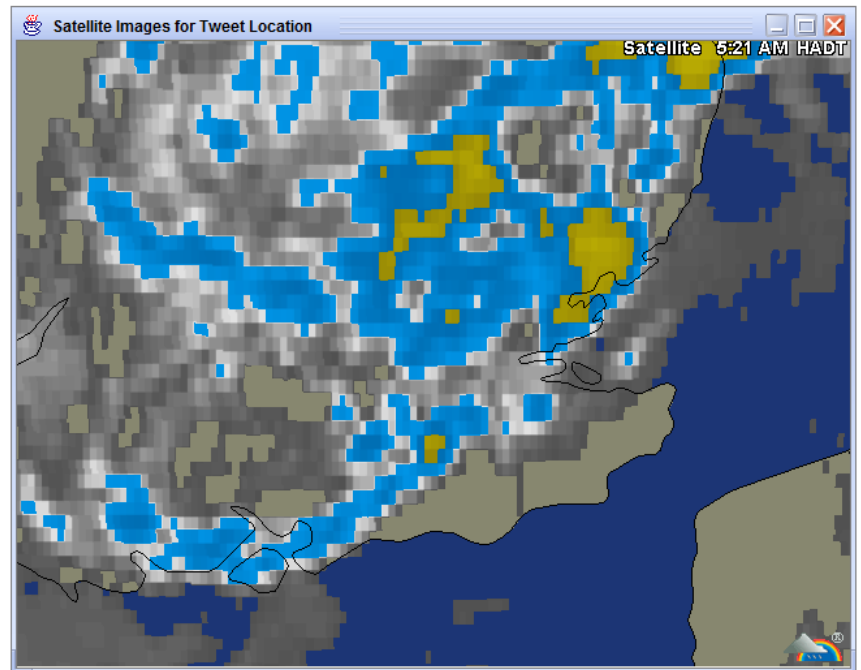


Additionally, the following window will be displayed specifying exactly where the three webcams are (numbered 1 – 3, with respect to the webcams above numbered 1 – 3 from left to right), in relation to the Tweet location labelled 'TL'. Both satellite and road views are displayed, with the images being returned from Google Maps ^{Reference 4} and Bing Maps ^{Reference 5}. The map zoom level automatically scales as appropriate depending on the proximity of webcams to the Tweet location.



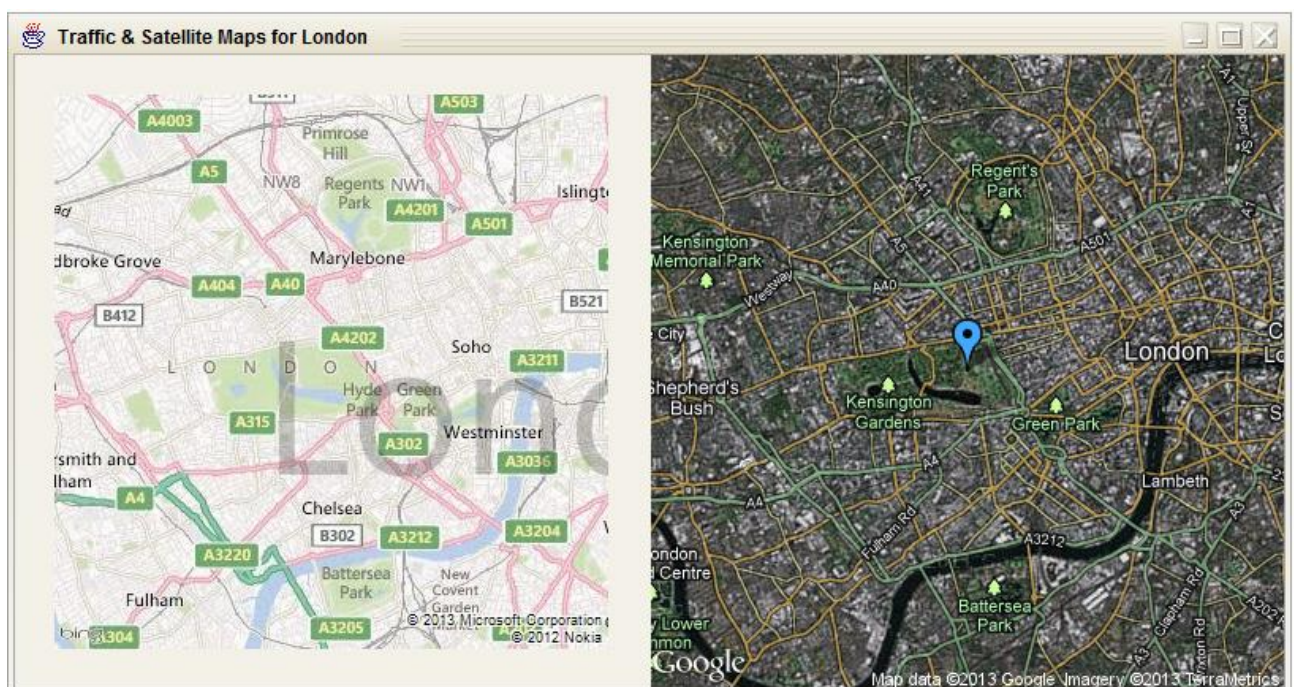
2. Animated Satellite Imagery

This will display an animated moving image of current and predicted future weather, centred on the tweet's latitude & longitude coordinates. The image updates within the Java frame, with a clock at the top right of the image refreshing with the image in hourly increments.



3. Tweet Location and Traffic Map of [Location Name]

The specific location text on the button itself will change depending on the location of the Tweet. When clicked, two images will be displayed in the same Java frame - the first showing a Microsoft Bing traffic map, and the second a Google maps version that shows more of the surrounding area. The green lines on the Bing map show low road congestion, while the red lines indicate that current congestion rates are high.



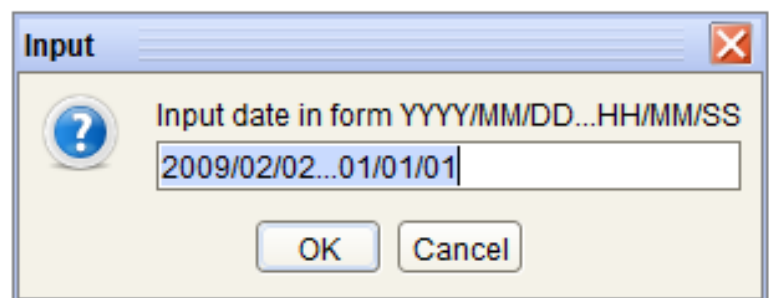
4. Current Flickr Photos

Selecting this option will load the most recently upload public Flickr image that was taken within the same geographical area as the tweet location. Images with EXIF data specifying the original photo taken location are prioritised over those that contain any other kind of coordinates, such as upload location.



5. Historical Flickr Photos

Clicking on this button will prompt the user to input the desired time and date that they wish to search a Flickr photo for. The location of the Tweet of Interest interface that they select the button from will determine the coordinates used in the search. A default date is automatically inserted into the text field and highlighted, to make the process of conforming to the required format easier. Clicking OK will then search for a fitting image.



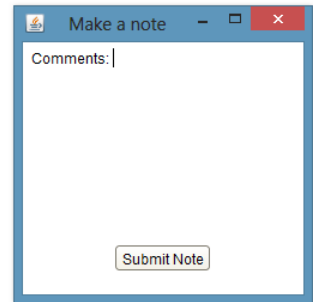
6. News Articles

This will search through Feedzilla.com's news archive ^{Reference 6} for the most recent news article containing the Tweet location's name anywhere in the title or content, and displays the returned information. Clicking the link at the bottom of the window will open a user's default browser and navigate to the news article's original source location online.



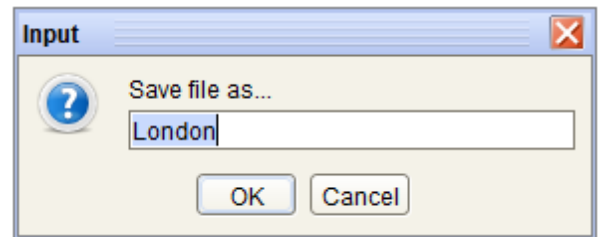
7. Add / Edit Note

The button text will change from 'Add note' to 'Edit note' depending on whether a note has previously been created for a particular Tweet. Clicking on the button will generate a Java frame containing an editable text box and a submit button. Submitting the note will add it to the audit trail, and additionally append it to the Tweet save file if the user goes on to save the Tweet file.



8. Export as Text file

This will prompt the user for a file save name (automatically suggesting the tweet location as a save name option), and then save all information, including user added notes, as a text file into the directory ...\\storedDataSets .



9. Export as XML file

This will perform the same action as described above, but as an XML file.

Design Modifications

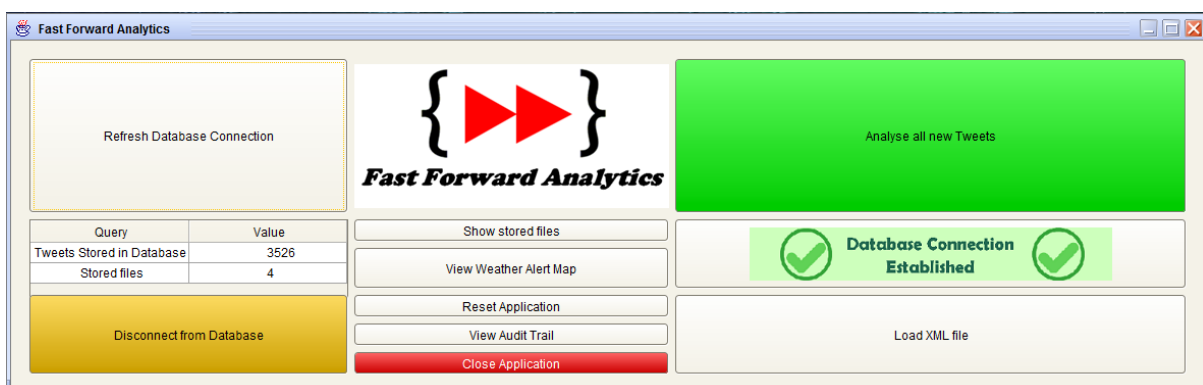
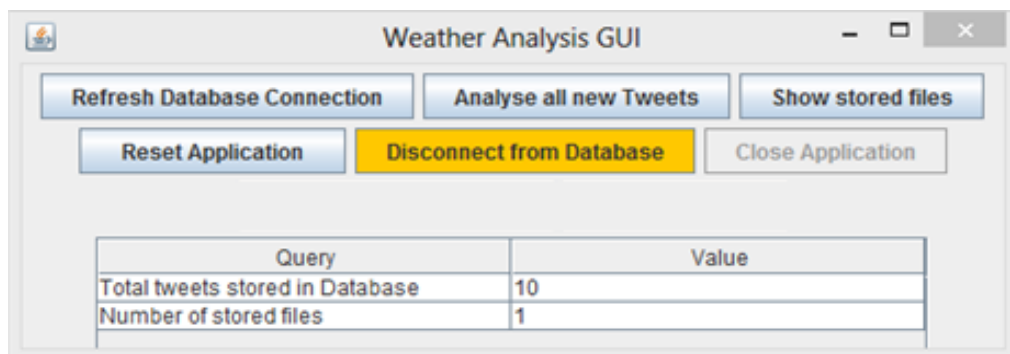
Main GUI Improvements

Alongside implementing a number of new features in to the system since the time of the Interim report, I have also revisited a number of user-interface areas that I felt could use some improvement. Often these areas have been sections that I initially created as a prototype and had planned to revisit later, but some improvements were made after receiving user feedback, both from planned feedback sessions and from actually having the project used by a real organisation – further details of this are detailed later in the report. In this section I will outline the improvements that have been made in the last few months.

To help with these refinements on the appearance of the system, I decided to make use of the JTattoo LookAndFeel software ^{Reference 7} which provides numerous options to help modify the overall GUI theme. I settled on using the 'SmartLookAndFeel' option as I felt it offered the most professional-looking theme, and helped to differentiate the application from other standard Java user interfaces. This effect is achieved by applying the following code in my GUI Java class, after ensuring the correct JTattoo Jar file is installed.

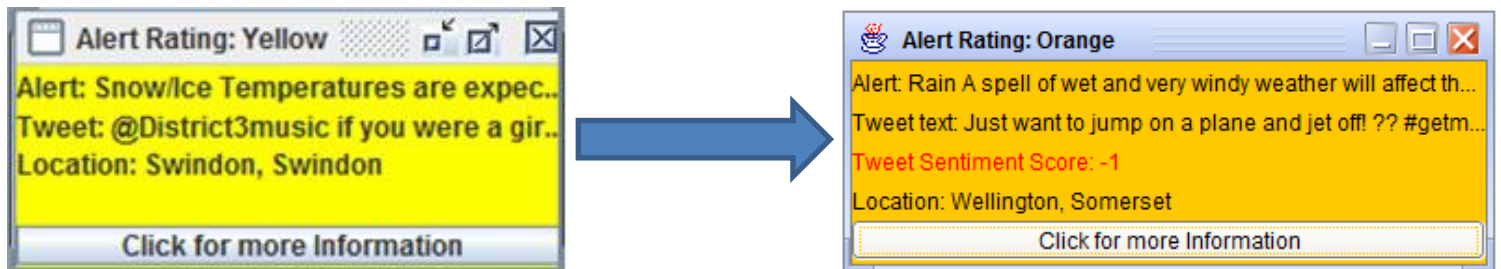
```
UIManager.setLookAndFeel("com.jtattoo.plaf.smart.SmartLookAndFeel")
```

The image below shows the evolution of the GUI design. Whilst the original was functional, little thought was made to the layout of the design, and the only form of user feedback was in the colouring of a few of the buttons and the table of information at the bottom.

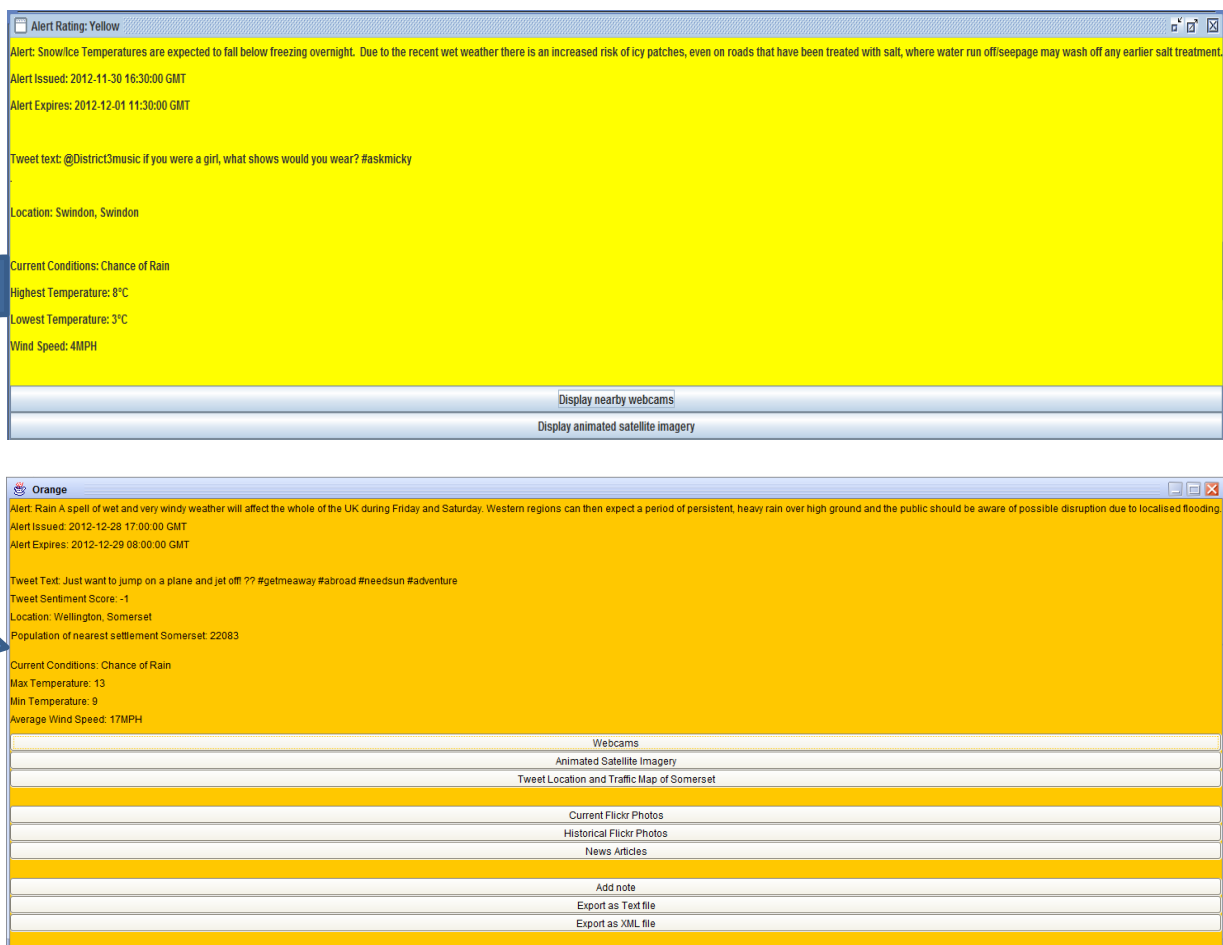


Tweet of Interest GUI Improvements

The effects of the modified 'look and feel' of the overall GUI theme can again be found in the appearance of the Tweet of Interest windows. I believe that the design of the window would now seem more familiar with most users as it has the three classic minimise, maximise and close buttons present – these are not the default Java icons, however through using the Jtattoo package I have changed it as shown below.

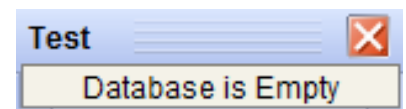


A number of functional improvements have also been made to the advanced tweet window, as can be seen below with the increase from two analysis operation buttons to nine, grouped into three appropriate sections (exact location analysis, wider location analysis, and the notes & saving section). Furthermore, Tweet sentiment and location population information is now present in the updated version.



Miscellaneous Design Additions & Improvements

- As a result of user feedback, I modified some of the current GUI colours in order to increase ease of use & readability. For example, the alert-coloured Tweet of interest windows have potential colours of Green, Yellow, Orange or Red. Whilst having black text on top of these backgrounds was suitable in most cases, it made the information difficult to read when a red alert tweet had been found. As such, I modified the code and so now when a red alert box is created, the text colour is white, resulting in a more user-friendly reading experience.
- I have exported the application into an executable file, with the 'Fast Forward Analytics' logo used for both the icon image and initial loading splash screen. This was achieved by first exporting as a Jar file, and then using the application 'Jar2EXE'. Reference 8
- Previously all user and system status notifications were presented to the user in a popup box, such as that displayed to the right. User feedback & testing resulted in comments explaining how these felt unnecessarily intrusive, and that a more seamless and integrated solution should be considered. This is what led to the aforementioned dynamic button implementation, which displays information to the user directly from within the main GUI, and also has the added benefit of providing additional detailed information upon a press of the button.
- In the first GUI implementation I required the user to disconnect from the database before exiting the application. I have now removed this minor additional user task, and simply added the disconnect code to the exit button itself.



Implementation

I developed the Java application using the Eclipse IDE (Integrated Development Environment), which is an application designed to increase developer productivity through offering a wide variety of features and plugins. In this section I will detail some of the critical and interesting sections of the code used to produce the system, however the rest of the code in the fourteen Java classes are commented and should be relatively easy to comprehend. Some elements of the code extracts to follow have been edited for ease of understanding or presentation – full unedited code is included in the accompanying supporting files.

I manually coded every GUI in the Java application, using a variety of different frame layouts. I decided against using a form of a Swing GUI builder such as NetBeans' 'Project Matisse' for a personal development reason, as I generally find that if I first learn the intricacies of a process such as GUI designing by hand, then I should have a solid foundation of understanding for if I encounter a problem in future projects when developing with such a tool. As a result I feel that I now have the confidence to explain what the precise role of each element and variable is, and whilst it is not necessary to explicitly describe every line of code in this section, I took care to ensure that I commented any unconventional and uncommon code for my own reference when I referred back to previously written classes.

I should state that my decision to use Java as the system's programming language was not always a certainty. I had previously found some of the Java syntax from previous modules difficult to understand, and as such carefully considered the pros and cons of using another language. The two other languages that I had strongly considered using were Matlab, which I had experience with from the Multimedia modules, and C Sharp (C#). Whilst I had not had any previous experience with C Sharp, I had heard positive comments about its ease of use, and did actually began developing the system using the language.

After developing with C Sharp for a few days, I decided that it would be in the project's best interests to develop with Java for a number of reasons. Firstly, I had researched and encountered a number of limitations of C Sharp that I felt could have the potential to hamper the project in later stages, and did not want to lock myself into a programming language that could limit future progress. Secondly, I found a noticeably smaller amount of documentation and active user-forums discussing C Sharp than I did for Java, which again could have led to problems further into the development process. Finally, previous student Nicholas Horne's project was written in Java, and so I feared that creating interfaces between the two projects using different languages could become an otherwise unnecessary problem that could damage both efficiency and functionality. I finally came to the decision that it would be best to concentrate on Java for now whilst keeping the option of using Matlab open later, and so proceeded to begin reacquainting myself with the previous Java material and examples that I had experienced in previous modules.

Main GUI

I utilised the Java GridBagLayout manager to produce the final GUI design as I felt that it offered me the most flexibility when it came to precise positioning options. My first user interface made use of the more simplistic but restrictive GridLayout option, which I found useful in the development phase due to the relatively simplistic options - it was very easy to simply add another button and let the system manage issues such as button order and size. Once I had decided on how I wanted the system to look and had determined roughly the number of buttons I needed, I began to experiment with using the FlowLayout manager. Whilst I found this more adaptable than GridLayout, I was not satisfied with the final result and so eventually turned to the more capable GridBagLayout option, which I will now explain in further detail.

Sets the frame title as Fast Forward Analytics.

Will exit the entire application if this Main interface is closed, closing all other open frames.

```
super("Fast Forward Analytics");
setSize(1150,378);
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
final Container pane = getContentPane();
pane.setLayout(new GridBagLayout());
((JComponent) pane).setBorder(new EmptyBorder(15, 15, 15, 15));
((GridBagLayout)pane.getLayout()).rowHeights = new int[] {0, 0, 10, 0, 0, 0};
```

Sets a border of 15 pixels between the edge of the window and the elements inside on every side.


Sets the default size of frame in pixels. The frame still remains user-resizable.

Sets the separation distance between each element within the frame. Six values here indicates that there are six rows of elements in this frame, with a separation of 10 pixels applied to the third element.

On the following pages I will explain how I implemented the features present on the main GUI frame. The method in which all actions are triggered is through an event listener with appropriate 'try' and 'catch' statements, which I will not include in the code extracts in order to keep the explanations as focused as possible.

Analyse Tweets Function

This is the most important and complex task in the system, and its operational code spans a number of java classes that I have tried to separate into discreet files for code clarity.

The first action that occurs is the program will change the user's mouse cursor to the operating system's "defaultWait" cursor:  This is achieved with the following code.

```
pane.setCursor(Cursor.getPredefinedCursor(Cursor.WAIT_CURSOR));
```

It will then proceed to delete all files stored in the temporary files directory, to both increase program efficiency and also make any form of debugging easier as the developer is not presented with files from multiple searches. A comment stating that a search was initiated is added to the audit trail, and then the actual searching algorithm begins.

```
weatherFilter.main(new String[0]);
```

The first class that is executed is 'weatherFilter.java' – this is the modifiable class that can be used to change the filtering requirements of the application. This particular filter searches through all Tweets and alerts the user to a Tweet of Interest if the tweet location is defined as originating from an area with a current weather alert active. A prerequisite of meeting this criteria is the requirement that the Tweet also has associated geo-location metadata, and so any Tweets without this information are discarded before further analysis is performed.

The system will then search through all Tweets, returning the tweet ID, tweet text and latitude & longitude values if present, and assigning them to Java variables. Information such as the tweet text and data & time information is returned in the same element, and so I use the following code in order to return only the information I desire:

```
int startPosition = tweetText1.indexOf(": \""") + ": \"".length();  
int endPosition = tweetText1.indexOf("\"", "\"date\"", startPosition);  
String tweetText = tweetText1.substring(startPosition, endPosition);
```

This will 'clean' the string, and store only the required information.

The system will then query The Weather Channel's API partner Wunderground.com, using the collected latitude & longitude values as parameters for each Tweet with geo-coordinates:

```
URL WundergroundWeatherXml = new  
URL("http://api.wunderground.com/api/205f2bb905cf8536/forecast/geolookup/q/" + latitude  
+ "," + longitude + ".xml");
```

My unique wunderground.com API key.

This request will generate data relating to both meteorological weather information such as alerts and current conditions, and additionally returns information such as the nearest weather station that is used in a future query. I search through this data by locating specified parent nodes and navigating through the XML tree through child nodes until the system finds the correct information. All elements that are useful for functionality of the program are assigned as variables and optimised if necessary (such as in the previous example of removing unnecessary information from a string).

Searches for node 'avewind' within the parent node, as defined earlier within the previous NodeList.

```
NodeList avewind = forecastdayElement.getElementsByTagName("avewind");  
for (int h = 0; h < avewind.getLength(); h++) {  
  
    Node avewindNode = avewind.item(h);  
    if (avewindNode.getNodeType() == Node.ELEMENT_NODE) {  
  
        Element avewindElement = (Element) avewindNode;  
        NodeList mph = avewindElement.getElementsByTagName("mph");  
        Element mp = (Element) mph.item(0);  
        if(z == 1){  
            z = z + 1;  
            System.out.println(mp.getTextContent());  
        }  
    }  
}
```

The wunderground API returns a weather forecast for the future seven days. This 'z counter' line ensures only the first (i.e. current) information is displayed, however can be removed as necessary if the entire forecast is required.

mp.getTextContent() contains the average wind speed generated by Wunderground.

The code's execution rate is heavily dependent on both the number of elements in the array and the internet connection speed. When this section has been completed, the program proceeds to the 'getAlert.java' class which is used in determining if a weather alert is active for a given Tweet location. Whilst it may seem inefficient to search for this after gathering the previous data, it is a necessary step in the system process due to the fact that alerts are only capable of being determined by querying the nearest weather station directly (data which is gathered from the previous filtering.java class at the same time as the other queries), rather than by coordinates directly.

The first task that the getAlerts.java works on is ensuring that all tweet information is 'clean' by performing conformity checks and performing changes to the data format if required. This is necessary due to data being sourced from many different organisations, depending on the data's country of origin – the UK Met Office's data layout is different to that of Météo-France's for example.

The algorithm will then perform the following URL query, with the variable 'location' being the unique identifier of a Tweet's nearest meteorological weather station.

```
URL AlertXml = new URL("http://api.wunderground.com/api/205f2bb905cf8536/alerts" +  
location + ".xml");
```

This station ID is often that of an airport or large governmental/military complex, and takes the following 16-digit form: zmw:00000.283.03779. This is known as a World Meteorological Organization (WMO) identifier – a universally adopted standard that helps to provide world-wide interoperability for this filter.

If a tweet is found to be of interest, the program outputs a system 'beep' to signify the finding, and the tweet's elements are stored in a unique temporary text file through appending the appropriate variables to a file created through executing the following code:

```
Toolkit.getDefaultToolkit().beep();  
PrintWriter writer = new  
PrintWriter("C:\\...\\temporaryFiles\\outputTweetsWeatherAlertsRefined" + unique +  
".txt");
```

Additionally, all database JSON data stored on that particular file is copied from the database and into the Tweet file through the 'readFile.java' class. I decided to include this data so that in the event of any kind of database error, the potentially crucial information that had been recorded could still be found in the temporary files directory - until a system reset is performed which removes these files.

After this operation has been completed for each tweet, the system will search the temporary files directory for the file 'forpopUp1.txt'. If a tweet of interest has not been found then this file will not be present, and so the system will display a notification informing the user that no tweets of interest have been found. If the file is found, then the class 'popupBox.java' is executed, and the system passes necessary arguments, such as the number of tweets found, to this next class.

If file doesn't exist, show popup box via a new thread informing the user that no tweets of interest were found.

```
File file = new File("C:\\...\\temporaryFiles\\forpopUp1.txt");
boolean exists = file.exists();
if (!exists) {
    nothingToReport.add(new JLabel("      No Tweets of interest found      "));
    ScheduledExecutorService s = Executors.newSingleThreadScheduledExecutor();
    s.schedule(new Runnable() {
        public void run() {
            nothingToReport.setVisible(false);
            nothingToReport.dispose();
        }
    }, 3, TimeUnit.SECONDS);
} else {
    popupBox notify = new popupBox(counterForpopupBox, 400, 100, 1);
}
```

Display popup for three seconds before removing.

If file does exist then proceed to popupBox.java class.

The class 'popupBox.java' then displays the retrieved information in a new window, as described in the 'Design' section and again displayed below.



Tweets of Interest

The class `popupBox.java` is responsible for displaying notable data in an easy to read format, querying other APIs with the stored data, and recording any new data into the existing database. It has to be modular in the sense that there will often be multiple Tweets being displayed simultaneously, and if the user clicks to view information such as location info, a map for the location of that particular Tweet must be presented, whilst keeping other Tweet processes in view and ready to be processed as well.

The majority of the GUI layout and presentation code was covered in the interim report for the basic and advanced views, and whilst a number of amendments have been made since then, the overall operation has not changed and so will only be briefly commented on in this section.

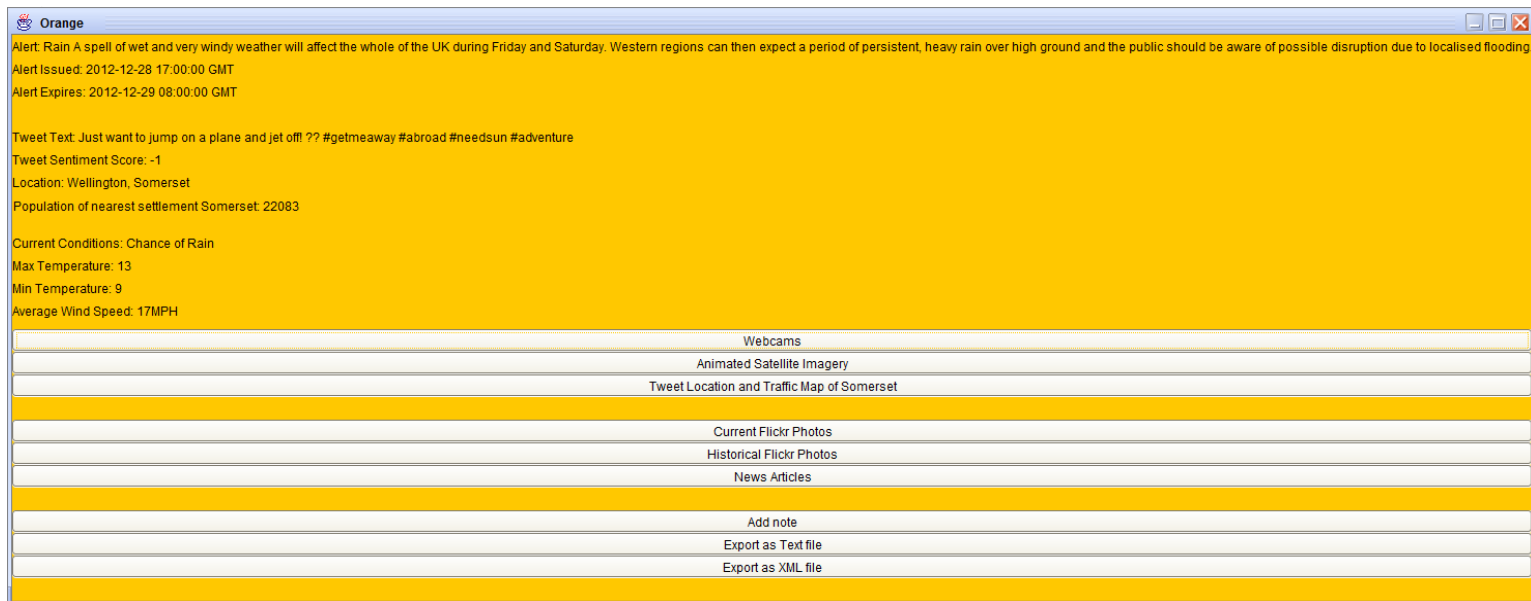
The basic user interface is generated with the following code, which should be relatively easy to comprehend.

```
frame.setTitle(alertRatingLine);
frame.setLayout(new GridLayout(5, 1));
TextArea textArea = new TextArea("", 10, 20);
frame.add(new JLabel(alert));
frame.add(new JLabel("Tweet: " + tweetText));
frame.add(new JLabel("Tweet Sentiment Score: " + sentimentScore));
frame.add(new JLabel("Location: " + locationText));
frame.getContentPane();
frame.add(infoButton);
frame.pack();
```

The code will then set the background colour of the JFrame, depending on the alert rating for the Tweet location. Due to the alert data being sourced from multiple countries it is necessary for the program to translate alerts from different languages, as shown below with the system checking for a 'Green' alert with the German equivalent 'Grün'.

```
if (alertRatingLine.equals("Alert Rating: Green") || alertRatingLine.equals("Alert Rating: Grün")) {
    frame.getContentPane().setBackground(Color.GREEN);
    frame2.getContentPane().setBackground(Color.GREEN);
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    frame.pack();
    frame2.pack();
    frame.setVisible(true);
} else if (alertRatingLine.equals("Alert Rating: Yellow") || alertRatingLine.equals("Alert Rating: Gelb")) {
    {
        ...
    }
}
```


The code that performs the webcam search and satellite imagery was covered in the interim report. I will now summarise the additional analysis options added since then to the advanced view shown below. The actual code for these operations (found in the `popupBox.java` class) is adequately commented for a developer to follow if required.



- Tweet Location and Traffic Map

The system will query Google and Microsoft Bing maps, using the latitude and longitude values as variables within the URL request address.

- Current Flickr Photos

This feature operates by utilising the Yahoo Flickr API ^{Reference 9}, using the Tweet location as a variable (gained through the previous Wunderground.com query), and returning the most recently uploaded image. The API is however more complex in its operation, as only the photo ID value is initially returned. In order to retrieve the image automatically without requiring the user to manually search Twitter, the system will use this photo ID as a key in another API request in order to return the server farm that the image is stored on in Flickr's database of images, and gain a variable titled by Flickr as 'secret'. Only with this 'secret' variable can the image actually be returned through a final third API call.

- Historical Flickr Photos

This is a similar operation to the summary above, but using a user-defined date in the original query instead of searching for the most recently uploaded.

- News Articles

This operation queries Feedzilla.com via their developer API and returns news articles that contain the tweet location anywhere in either the title or the text of the article.

- Add / Edit Note

This code works by firstly creating a new JFrame window, and then (depending on whether a note had previously been added), will fill the text edit area with the previously-made comments.

```
final JTextArea textArea = new JTextArea("Comments:", 10, 20);
JButton submitTextButton = new JButton("Submit Note");

...
submitTextButton.addActionListener(new ActionListener() {
    ...
    dialog.add(new JLabel("        Note stored successfully        "));
})
```

Confirmation of the note save command is displayed after clicking 'Submit' within the note window.

Default columns & rows value for text area.

- Export as Text / XML file

This saving function utilises the Java Document Builder feature, and appends variables directly to a file for the text version, and via a more sophisticated element & node layout for the XML version.

Checks to ensure the user has entered a filename.

Defaults to tweet location as a save name suggestion.

```
String userInput = JOptionPane.showInputDialog( "Save file as...", locationForNews);
if ((userInput != null) && (userInput.length() > 0)) {
    xmlSaveName = userInput;
}
Document doc = docBuilder.newDocument();
Element rootElement = doc.createElement("Tweet");
doc.appendChild(rootElement);
```

Saves element stored in 'rootElement' as a child to its parent's root node.

Miscellaneous Features

- Main GUI – Show Stored Files

The code required for this operation is relatively easy to understand, and results in the operating system's default file explorer opening at the specified directory:

```
Desktop.getDesktop().open(new File("C: ... \\storedDataSets"));
```

- Main GUI - Close Application

Along with simply closing the application, this button will also disconnect from the database and 'kill' that system process in order to save resources for the user.

```
Runtime.getRuntime().exec("TASKKILL /F /IM mongod.exe")  
System.exit(0);
```

The 0 in the brackets is used for debugging purposes, and indicates a successful and planned system exit.

Will search for the process 'mongod.exe' as a running process, and 'kill' it if it is found. This is the process that connects the application to the database.

- Main GUI - Connect to Database

This will ensure that the processes 'mongod' and 'mongo' are running – both are required for a connection to the database.

Connects to Mongo Database, located on the local host (port 27017), and then finds the collection 'tweets' within the database 'test'.

```
Process proc = Runtime.getRuntime().exec("C:\\mongodb\\bin\\mongod.exe");  
Thread.sleep(100);  
proc2 = Runtime.getRuntime().exec("C:\\mongodb\\bin\\mongo.exe");  
Mongo = new Mongo("localhost", 27017);  
DB db = mongo.getDB("test");  
DBCollection collection = db.getCollection("tweets");  
analyseButton.setBackground(Color.GREEN);  
disconnectButton.setEnabled(true);  
analyseButton.setEnabled(true);
```

These commands will enable the 'analyse' and 'disconnect' buttons only when the processes are running, as both require an active database connection.

- Main GUI - Weather Alert Map

Stores the image URL as a URL variable.

The WindowListener allows for later detection of window closure – for appending details to the audit trail.

```
final JFrame mapFrame = new JFrame();
mapFrame.addWindowListener(new WeatherAlertFrameClose());
mapFrame.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
URL europeMapURL = new URL("http://icons-
sf.wunderground.com/data/640x480/2xeu_severe.gif");
ImageIcon image = new ImageIcon(europeMapURL);
JButton keyButton = new JButton("  Key  ");
mapFrame.add(keyButton);
mapFrame.add(new JLabel(image));
mapFrame.pack();
```

Adds the 'Key' button and alert map image to the window.

Converts the URL variable into an image.

- Main GUI - Dynamic Status Indicator

The code for the status indicator that details information such as if a database connection has been established, or if a tweet of interest has been found, operates under a process of elimination. The system first checks if a Tweet has been found – if it has, then it displays the appropriate status. If not, then it will check for another status and only stop checking when a status has been determined. There is no possible system status that is not represented by one of the conditions below in the system hierarchy check.

Tweet
Found

Searching
for Tweets

System
Ready

No
Internet
Connection

No
Database
Connection

Database
Empty

The program checks for an internet connection by executing the following code by attempting to access <http://www.google.com>. If an error is returned then an internet connection is deemed to be unavailable, else the code will return true. This code is clearly dependant on Google's homepage being active, however if in the unlikely event that Google was unreachable, the system would still operate as normal - it would just produce the 'No Internet Connection' system status error.

```
URL url = new URL("http://www.google.com");
HttpURLConnection urlConnect = (HttpURLConnection)url.openConnection();
Object objData = urlConnect.getContent();
} catch (UnknownHostException e) {
    e.printStackTrace();
    return false;
}
catch (IOException e) {
    e.printStackTrace();
    return false;
}
return true;
```

The string in the popup box triggered by the user clicking on the system status box (as explained in the *Design* section), is also created and changed at the same time through the use of a simple `addListener`.

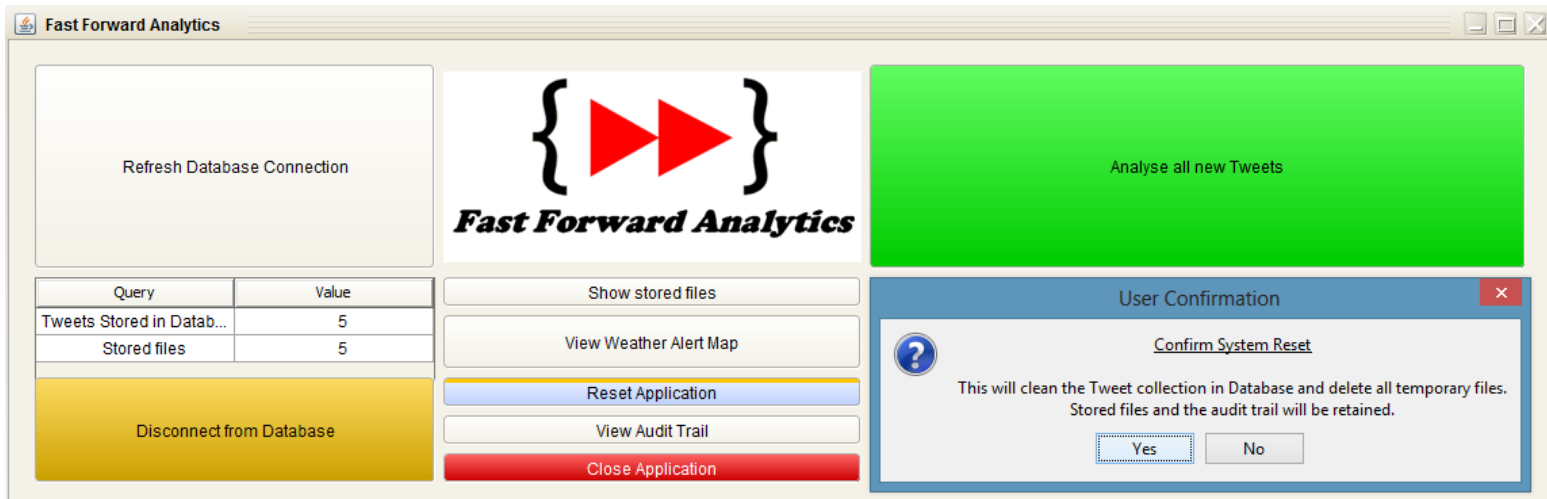
- Saving

The save feature will combine all collected information, including user added notes, and store them in a correctly-formatted file such as the XML example output below.

```
<?xml version="1.0" encoding="UTF-8"?>
<Tweet>
  - <Tweet_Information>
    <Tweet_ID>284786336509091840</Tweet_ID>
    <Tweet_Text>Another day another $$$ #work</Tweet_Text>
    <Latitude>51.508360</Latitude>
    <Longitude>-0.162952</Longitude>
    <Location>London, London</Location>
  </Tweet_Information>
  - <Weather_Information>
    <Temp_High>13</Temp_High>
    <Temp_Low>9</Temp_Low>
    <Wind_Speed>17MPH</Wind_Speed>
    <Conditions>Chance of Rain</Conditions>
    <Weather_Station_Identifier>/q/zmw:00000.283.03779</Weather_Station_Identifier>
  </Weather_Information>
  - <Alert_Information>
    <Alert>Rain A spell of wet and very windy weather will affect the whole of the UK during Friday and Saturday. Western regions can then expect a period of persistent, heavy rain over high ground and the public should be aware of possible disruption due to localised flooding.</Alert>
    <Alert_Rating>Orange</Alert_Rating>
    <Alert_Description>The weather is potentially dangerous. The weather phenomena that have been forecast are not unusual, but be attentive if you intend to practice activities exposed to meteorological risks. Keep informed about the expected meteorological conditions and do not take any avoidable risk.</Alert_Description>
    <Alert_Issued>2012-12-28 17:00:00 GMT</Alert_Issued>
    <Alert_Expired>2012-12-29 08:00:00 GMT</Alert_Expired>
  </Alert_Information>
  - <Other_Information>
    <User_Notes>Cardiff Tweet file saved</User_Notes>
    <Semantic_Analysis_Score>-1</Semantic_Analysis_Score>
  </Other_Information>
</Tweet>
```

- Main GUI - Reset Button

The reset button will (after a user 'confirmation' popup box is displayed and accepted), remove all Tweet entries from the database, and delete all stored temporary files. The following code displays the key elements necessary in completing this action.



Displays confirmation box – only proceed if 'Yes' option selected.

```
int confirmed = JOptionPane.showConfirmDialog(null,
"<html><p align=center><u>title</u><br><br>text1<br>text2 </html>", "User
Confirmation", JOptionPane.YES_NO_OPTION);

if (confirmed == JOptionPane.YES_OPTION) {
    File f2 = new File("C:\\Users\\Steven\\Documents\\University\\Year 3\\Project
Code\\Tests\\temporaryFiles");
    String[] myFiles;

    if(f2.isDirectory()){
        myFiles = f2.list();
        for (int i=0; i<myFiles.length; i++) {
            File myFile = new File(f2, myFiles[i]);
            myFile.delete();
        }
    }
}
```

Navigates to the directory 'temporaryFiles', and sequentially deletes each file.

Security

Each time an event occurs that is recorded in the audit trail, the following code is executed (following the action itself is performed, for efficiency purposes).

```
PrintStream auditOut = new PrintStream(new
FileOutputStream("../security\\AuditTrail.txt",true));
System.setOut(auditOut);
java.util.Date date2 = new java.util.Date();
System.out.println(new Timestamp(date2.getTime())+ " - System Reset");
auditOut.close();
Security.main(null);
```

Executes Security.java class.

Generates times stamp.

Text describing event.

The (unencrypted) file AuditTrail.txt file is created and populated with the above text, and is then subjected to the following code, within the Security.java class.

Take previously created AuditTrail.txt as input file.

Set encrypted_Audit_Trail.txt as new output file. 'true' indicates that new data will be appended to the file, rather than replacing it.

```
FileInputStream fis = new FileInputStream("../security\\AuditTrail.txt");
FileOutputStream fos = new FileOutputStream("../security\\encrypted_Audit_Trail.txt",
true);

encrypt(key, fis, fos);

File file = new File("../security\\AuditTrail.txt");
file.delete();

public static void encrypt(String key, InputStream is, OutputStream os) throws Throwable
{
    encryptOrDecrypt(key, Cipher.ENCRYPT_MODE, is, os);
}
```

Deletes previously created unencrypted file.

This function computes the encryption, utilising DES (Data Encryption Standard). See full code upload for further details.

I believe that the DES encryption method used is currently sufficient for this version of the software. Should the system be developed further and rolled out to real-world organisations, a more secure encryption method – for example one of DES’s successors Triple DES, should be used. I used DES as it was relatively simple to implement as a means of showing that the application is capable of such features, and whilst not as sophisticated as some other methods, is still relatively secure for most use case scenarios.

The outcome of the security element of the code is that there is one file (encrypted_audit_trail.txt) that remains on the system within the project ‘security’ directory. The only way to legibly access this file is via the application through entering the correct decryption password (as shown in the *Design – Security* section). The code for this user input box is displayed below.

The ‘userInput’ variable is set to the password that the user entered, and is sent to the SecurityDecrypt.java class to be used as the decryption key. If the correct key is sent, the file ‘AuditTrailDecrypted.txt’ will be decrypted, else it will remain encrypted.

The third line simply ensures that the operating system’s default .txt application is used to open the text file.

```
String userInput = JOptionPane.showInputDialog("Enter Decryption Password");
SecurityDecrypt.main(null, userInput);

Desktop d = Desktop.getDesktop();
d.open(new File("C:\\...\\security\\AuditTrailDecrypted.txt"));
Thread.sleep(1000);
File file = new File("C:\\...\\security\\AuditTrailDecrypted.txt");
file.delete();
```

File.delete() will delete the file completely, bypassing the operating system’s recycle bin.

The decrypted file is only present on the machine for a short period (to allow time for the operating system to open the file), and is then immediately deleted. This ensures that the only way the decrypted file can be retained is through a direct user saving action within the default text file reader application.

If a user tries to access the file outside of the application, they will be shown the following illegible encrypted document.



If the user enters the correct decryption password, the audit trail will be displayed to the user, as shown below.

AuditTrailDecrypted - Notepad	
File Edit Format View Help	
2012-12-28 22:43:26.787 - European Weather Alert Image viewed. Source available at: http://icons-sf.wunderground.com/data/640x480/2xeu_severe.gif	
2012-12-28 22:43:32.55 - Frame closed : European Weather Alerts Map	
2012-12-28 22:43:36.605 - Tweet Analysis Initiated	
2012-12-28 22:44:07.784 - New Tweet - Detailed information for a Tweet originating from Bristol with Tweet ID: 284791581071011840 accessed	
2012-12-28 22:44:20.667 - 284791581071011840 - Satellite Imagery viewed. Source available at: http://api.wunderground.com/api/205f2bb905cf8536/api	
2012-12-28 22:44:22.784 - Frame closed : Satellite Imagery	
2012-12-28 22:44:40.598 - New Tweet - Detailed information for a Tweet originating from Bridgend with Tweet ID: 284791611202891776 accessed	
2012-12-28 22:44:43.251 - 284791611202891776 - Map images: http://dev.virtualearth.net/REST/v1/Imagery/Map/Road/51.669,-4.0163/12?pp=47.6156352,-1	
2012-12-28 22:44:45.466 - Frame closed : Bing & Google Maps	
2012-12-28 22:44:48.237 - 284786326950256640 - Flickr photo: http://farm9.staticflickr.com/8499/8319794692_254704fb08.jpg accessed	
2012-12-28 22:44:52.786 - Frame closed : Flickr Photos	
2012-12-28 22:44:57.865 - 284791611202891776 - Note added: "this is a note"	
2012-12-28 22:45:04.97 - 284791611202891776 - Backup: Bridgend.txt successfully exported	
MD5 Hash of Bridgend_backup.txt = 97a87669c835ef631d19a498ae14735a4	
2012-12-28 22:49:52.407 - System Reset	

Another security feature that I have included in the application is the generation of an MD5 hash of every saved file, and saving this checksum value to the audit trail. This would be useful if for example the authenticity of a saved file was to come into question – cross referencing the file's MD5 hash value with the value in the encrypted audit trail would confirm or deny any allegation that a file had been modified after the initial save. The Java class 'MD5.Java' is the class in which the computation is performed, and the key operations are performed using MessageDigest. Reference 10

Difficulties Encountered

In this subsection I will detail a number of problems I encountered throughout the development phase, and attempt to convey how I overcame these issues. This could either be through using an alternate method of thinking in order to solve the problem directly, or an attempt to change the problem itself into a more easily manageable one.

The primary and reoccurring problem that I encountered with regard to the user interface, was that of the Java GUI Event Dispatcher. The Event Dispatcher Thread is the thread on which non thread-safe Java Swing Events are managed and executed on, and without correct planning can result in a variety of irregular glitches, crashes and other unexpected behaviour.

I faced problems with this aspect of Java whilst I was designing the updated GUI, when I wanted to implement a number of advanced features such as the dynamically updating system-status images, along with retrieving and displaying information from a live and changing database. To further complicate matters I intended for these actions to be occurring at the same time as when the user was performing other tasks in a multitude of other GUI windows, each containing unique actions and variables which the system would have to process simultaneously.

Implementing a reliable GUI with as many different elements as this, all integrated so closely together, was a difficult and time consuming challenge. After a large amount of research and trial & error I eventually began to make progress by utilising Java executors, scheduling, 'runnable' commands, Swing workers and event queues that are only invoked when explicitly specified. Some of the code extracts below demonstrate how I used these relatively complex methods in order to achieve the current GUI, which did eventually solve the problems I had been encountering.

```
ScheduledExecutorService s = Executors.newSingleThreadScheduledExecutor();
s.schedule(new Runnable() {
    public void run() {
        ...
    }
})
```

This code extract guarantees that the task to be executed will not use more than one thread, and so helps to contain certain elements and keep them away from other aspects of code which it could otherwise interfere or be interfered with.

```
Runnable r = new Runnable()
{
    public void run()
    {
        ...
    }
}
EventQueue.invokeLater(r);
```

This simplified code extract keeps all JFrames displaying correctly simultaneously. Without this code, when updating one JFrame all others would become inactive, unresponsive and 'greyed out' which was both aesthetically displeasing and functionally inefficient.

Another problem I encountered that was very time consuming to solve, was an error in which occasionally and seemingly randomly the GUI would simply not return a requested image such as the weather map or Flickr images via a URL request. If I closed and re-launched the application, the button would then work. This issue persisted for a number of months during development, as I found it impossible to recreate the problem to properly debug it. Eventually, through extensive research, I discovered that the problem was related to Java trying to connect to an IPv4 Internet IP via IPv6, and would occasionally fail to connect and subsequently 'hang' on returning a URL ImageIcon. Fortunately there was a simple one-line solution, displayed below.

```
System.setProperty("java.net.preferIPv4Stack", "true");
```

This is equivalent to inputting the following text directly into a command window, and forces Java to use IPv4 rather than IPv6.

```
- Djava.net.preferIPv4Stack=true
```

Demonstration & Evaluation

As there is a wide range of potential uses for a project of this nature, I have decided to demonstrate two different use case scenarios of varying scale. Firstly, I will exhibit the large-scale analytical possibilities that the system is capable of on a national and even international level. I will then perform a medium-scale analysis in order to showcase the type of functionality that enables extensive monitoring of a situation, event or even a particular person. I believe that this evaluation method will help to validate the relevance of the system, with particular emphasis on illustrating how versatile the application is.

Large-scale demonstration: Political Analysis

I was aware from the project's beginning that recording large amounts of Twitter data at times of national or global interest could be beneficial for later analysis. As such, I decided to capture a significant amount of data on the US Presidential Election on November 6, 2012. I believed that this could provide me with interesting analysis opportunities in the future, and I will now discuss my findings after post-processing the results.

Background Information

The two major US presidential candidates were the incumbent Democrat Barack Obama, and the Republican contender Mitt Romney. It was largely accepted that the US state of Ohio would be a key voting state that would likely indicate who would win the election across the rest of the USA. The final results are displayed below, and they show that Ohio was indeed representative of the rest of the USA's overall vote, as the differences are within 0.6% of the national average for both candidates.

<i>Candidate</i>	National Vote Share	Ohio Vote Share
<i>Barack Obama</i>	51.1%	50.67%
<i>Mitt Romney</i>	47.2%	47.69%

In total I harvested 5584 tweets from 8:46pm – 10:54pm EDT local time (01:46am – 03:54am GMT), applying four different filters, as displayed below.

<i>Time (EDT)</i>	<i>Location</i>	<i>Criteria</i>	<i>Tweets recorded</i>
20:46 – 21:01	Ohio	All tweets	1409
21:05 – 21:10	Ohio	Keywords ‘Mitt’, ‘Barack’, ‘Romney’, ‘Obama’	472
21:22:05 – 21:22:55	All states	Keyword ‘Election’	786
21:35 – 22:54	Ohio	All Tweets	2917

My hypothesis was that the number of Tweets for the Republican & Democratic candidates would **not** be in line with the final result. My justification for this prediction is based upon the statistically founded differing characteristics of the two political party’s different voting groups. I have gathered a number of statistical comparisons on voter demographics and displayed them below.

Republican & Democratic voter Demographics

<i>Characteristic</i>	<i>Republican</i>	<i>Democrat</i>	<i>Difference (Republican to Democrat)</i>
<i>Average Age</i>	50 years	47 years	+ 3 years
<i>Male</i>	51%	41%	+ 10%
<i>Female</i>	49%	59%	- 10%
<i>Ethnicity: White</i>	87%	55%	+ 32%
<i>Ethnicity: Black</i>	2%	24%	- 22%

Reference 11

Twitter User Demographics

<i>Characteristic</i>	<i>Users</i>
<i>Age: 15 – 25</i>	73.7%
<i>Age: 26 - 35</i>	14.9%
<i>Age: 36 - 45</i>	5.5%
<i>Age: 46 +</i>	5.9%
<i>Male</i>	47%
<i>Female</i>	53%
<i>Ethnicity: White</i>	39%
<i>Ethnicity: Black</i>	61%

Twitter Usage Demographics

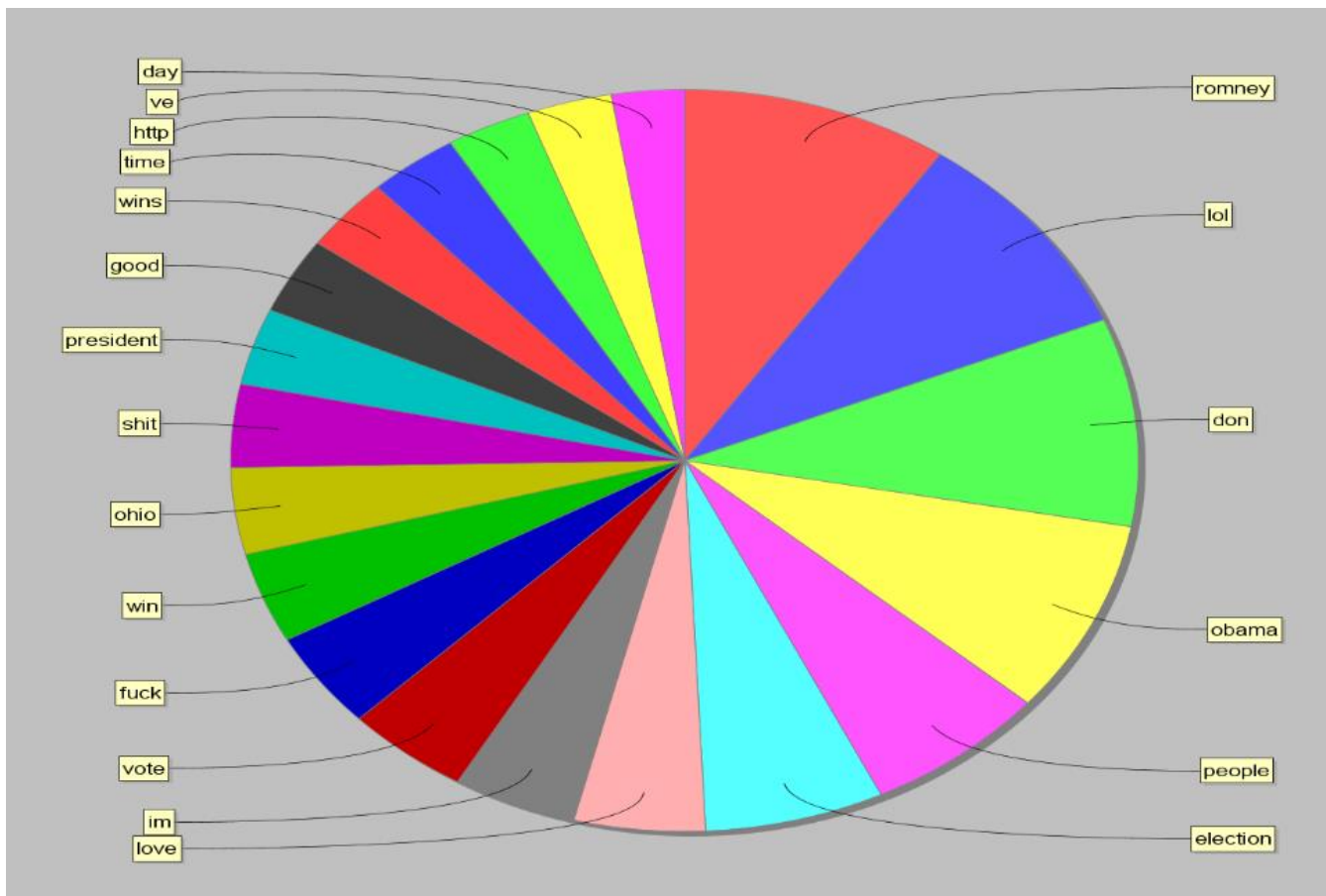
	<i>Male</i>	<i>Female</i>
<i>Total average Tweets per user</i>	567 Tweets	610 Tweets

Reference 12

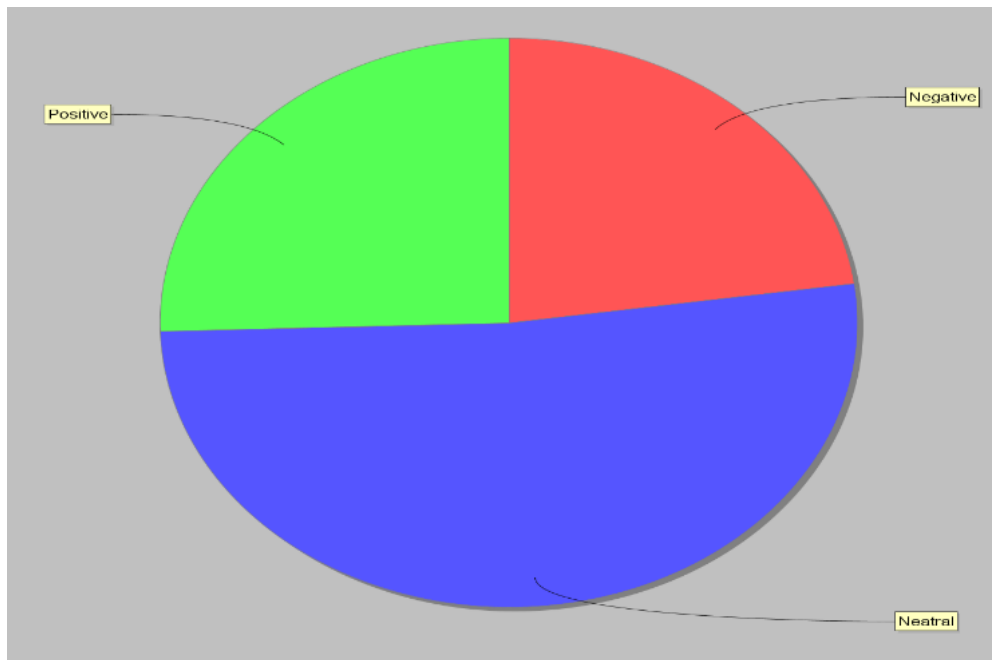
Analysis

I believed that the statistics above would surely result in an overrepresentation of pro-democrat supporters. Every category detailed, including age, gender and ethnicity, suggests that there would be a strong link between Democrat voters and their presence on Twitter. For example, it is reported that there are 18% more female Democrat voters than there are female Republican voters. This, coupled with the statistic that 53% of Twitter users are female, and that on average female users tweet more than their male counterparts, would suggest a more vocal Democrat collective Twitter 'voice'.

The pie chart below is displaying the keywords found in all 4326 Tweets harvested from Ohio, which had no keyword filter applied to the initial data gathering. Whilst it was encouraging to see that even without any keyword filtering the election was still clearly the major topic of conversation on Twitter, I was surprised that 'Romney' was the most tweeted keyword for this 94 minute harvesting period.

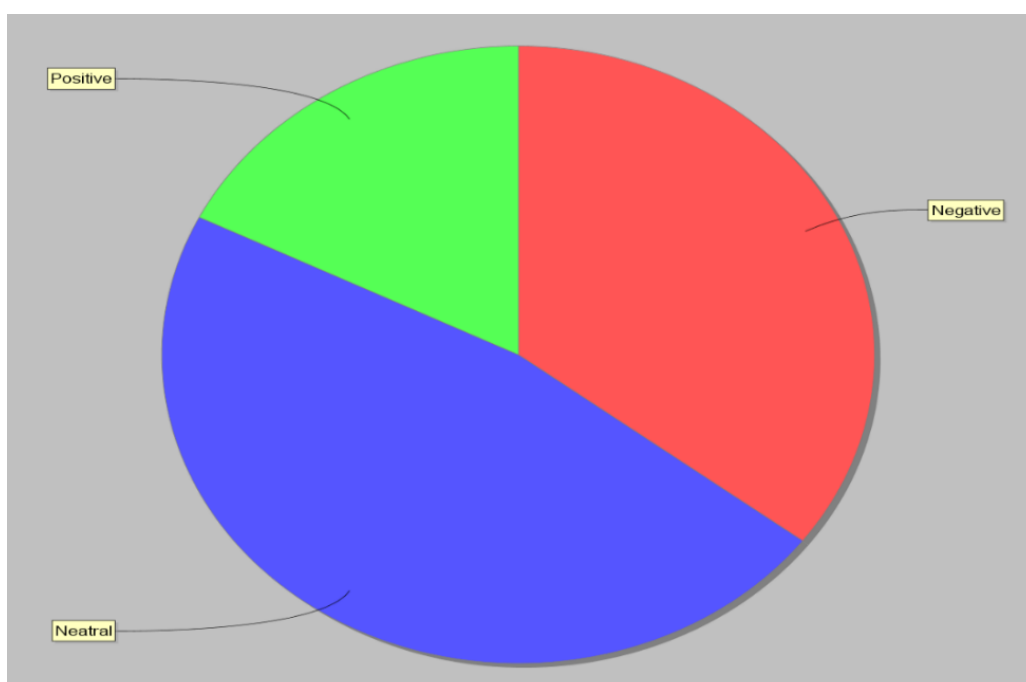


As this was not the result I had expected, nor one I could explain, I decided to analyse the data sample further to try to find a logical explanation. Through performing sentiment analysis on each of the Tweets via the Tweet harvesting program, I produced the chart on the following page showing the average overall positive, negative and neutral connotations for the harvested Tweets.



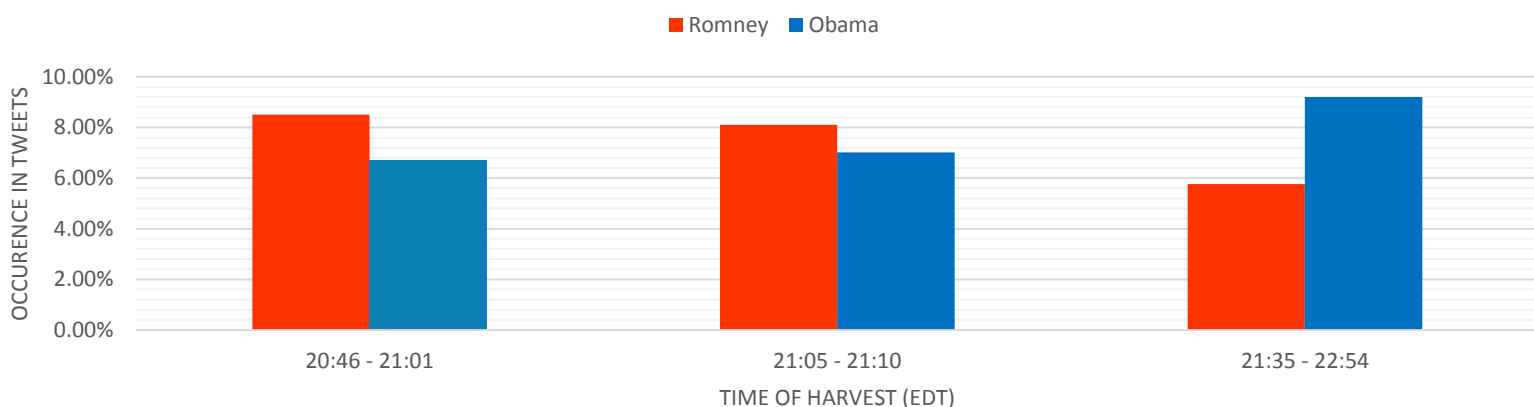
This still however did not explain why the results were not in line with both my initial hypothesis, and the actual Ohio election result. Through inspection of the sentiment analysis computation however, I realised that the election-related words 'win', 'wins' and 'winner' were improperly skewing the result and making the average Tweet connotations appear more positive than they really were. As such, I removed these keywords from the AFINN list used for the semantic score computation, which resulted in the graph below.

This result logically complemented the first keyword pie chart, and was the first scientific step that lead towards my final conclusions detailed later in this section.



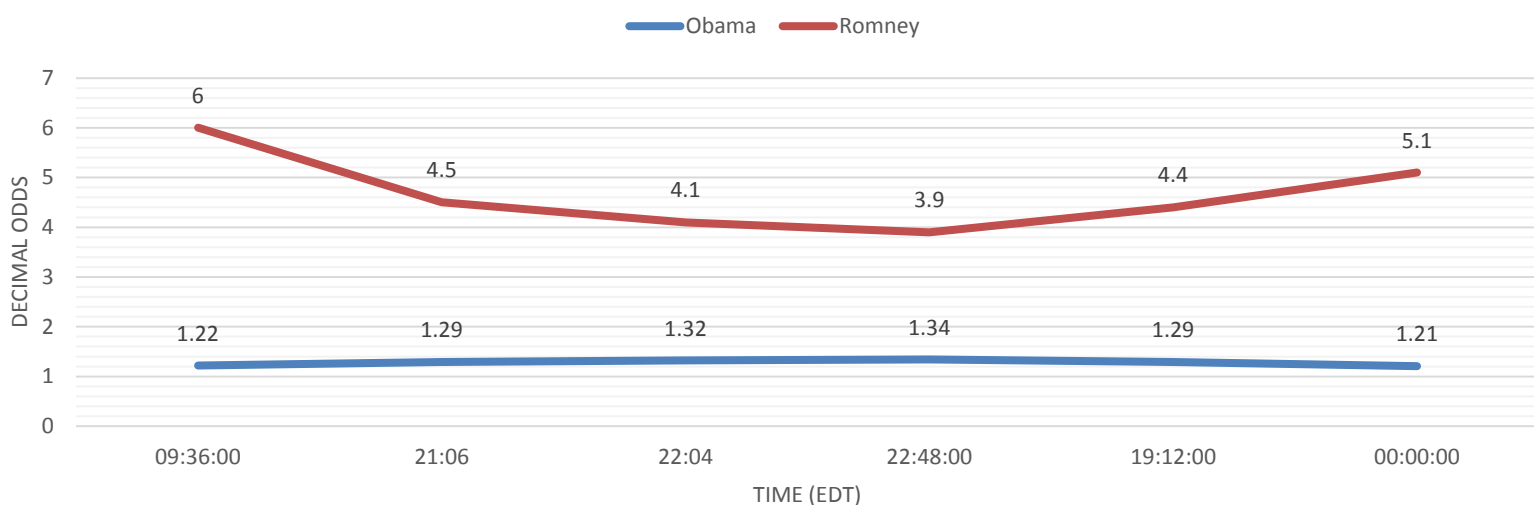
My next step was to again analyse the keywords and semantic sentiment score of the tweets, but this time with sensitivity to the point in time they were captured. The first bar chart below is a representation of the three Ohio datasets, and shows the changing frequency of keywords 'Obama' and 'Romney' in harvested Tweets as the election night progressed. The bar chart shows that Twitter conversations discussing Romney (the eventual loser in the presidential race) was initially higher than that of his competitor, however as the event progressed the topic of conversation reversed and Obama was the most Tweeted topic.

Frequency of Keyword in Tweets



The next bar chart is displaying data I gathered from the historical archives of Betfair, the world's largest betting exchange ^{Reference 13}. This data shows the fluctuation of decimal odds over the period that my Tweets were harvested. The further below 2 (or 'evens') a decimal odds value is, the more likely an event is deemed to occur. Conversely, odds higher than 2 are deemed unlikely to occur, with a rising value indicating a further decreasing probability.

Historical odds of Election betting



As you can see, over the first 13 hours of Election Day the odds for Romney to win decreased while Obama's increased, therefore indicating that the markets were gaining confidence in Romney's potential to win the election. However later on into the evening this odds trend reversed, and by midnight the markets were almost unchanged from where they had begun that morning. I believe that this information is useful in analysing why we saw the reversal in discussion topics between Romney and Obama, as I will explain below.

Collectively and as a result of the above analysis, I have concluded that my initial hypothesis was an oversimplification of the data, whereby I predicted that as there should statistically speaking be a higher proportion of Democratic (Obama) supporters on twitter, that there should also be more tweets containing the keywords Obama and Democrat. On the contrary, I found that in my sample of tweets there was actually more conversation about Romney, however interestingly the majority of this discussion was semantically marked as containing negative connotations.

This unexpected outcome made me eager to research the topic of Twitter user's posts globally, and whether on average all Tweets across Twitter were generally positive or negative, and to what degree. This led me to discover a number of recent academic journals on the subject, including most notably the paper 'Semantic Sentiment Analysis of Twitter', written by Hassan Saif, Yulan He & Harith Alani from the Open University ^{Reference 14}. They analysed much larger Tweet samples and came to the conclusion that generally Twitter users are more likely to Tweet about things they don't like than things they do, particularly when it comes to politics (interestingly, they conducted one of their research samples on the 2008 US Presidential election where they found that there was on average 1.5 times as many negative tweets as positive ones).

The combination of my own findings, which are consistent with other studies such as the Open University paper detailed above, has led me to the following overall conclusion:

My sample of Ohio US Election night tweets, which should be a strong indication of overall US sentiment due to the similarity of results there and across the nation, shows that Twitter users are more likely to discuss and comment on subjects or ideologies that they disagree with, rather than simply harmonizing with similarly-minded Twitter users on topics they do agree with. Additionally, in this example there also appeared to be a relationship between a section of society's political beliefs/ideologies, and their resulting activity on Twitter when it appeared to be coming under threat. This is suggested by the increase in negative Romney tweets as he was (according to the markets), making progress toward beating Obama, and an almost immediate reduction in the number of such Tweets once the markets (linked strongly to news coverage) had changed and were again predicting their own political/ideological leader's victory.

Medium-scale demonstration: Event Monitoring

To demonstrate the system's monitoring and live analysis capabilities on a real-world event, I decided to monitor proceedings occurring on Wednesday 17 April 2013 for Baroness Thatcher's funeral in London. I chose this event as I believed that there would likely be a large amount of Twitter conversation surrounding the event due to the strong divisive views that many of the public hold regarding her policies when in office. Additionally the event will be widely televised, and so any information that I gain should be verifiable by various reputable sources.

I was not sure how successful the investigation would be, however with information such as live webcams, news articles, traffic reports, regular Flickr images and the geographical mapping of positive and negative tweets at my disposal, I believed that it would prove to be at the very least an interesting showcase of the system's real-time monitoring ability on a national event.

Background Information

Baroness Thatcher was the first female British prime minister, serving from 4 May 1979 – 28 November 1990, and passed away on the 8 April 2013, aged 87. She is widely regarded as a deeply divisive political figure, and her public ceremonial funeral that took place on 17 April 2013 was expected to be viewed by millions around the world. The event is due to be very controversial for a number of political reasons, and as a result of the Boston marathon bombings just three days before, security surrounding the event is expected to be very high.

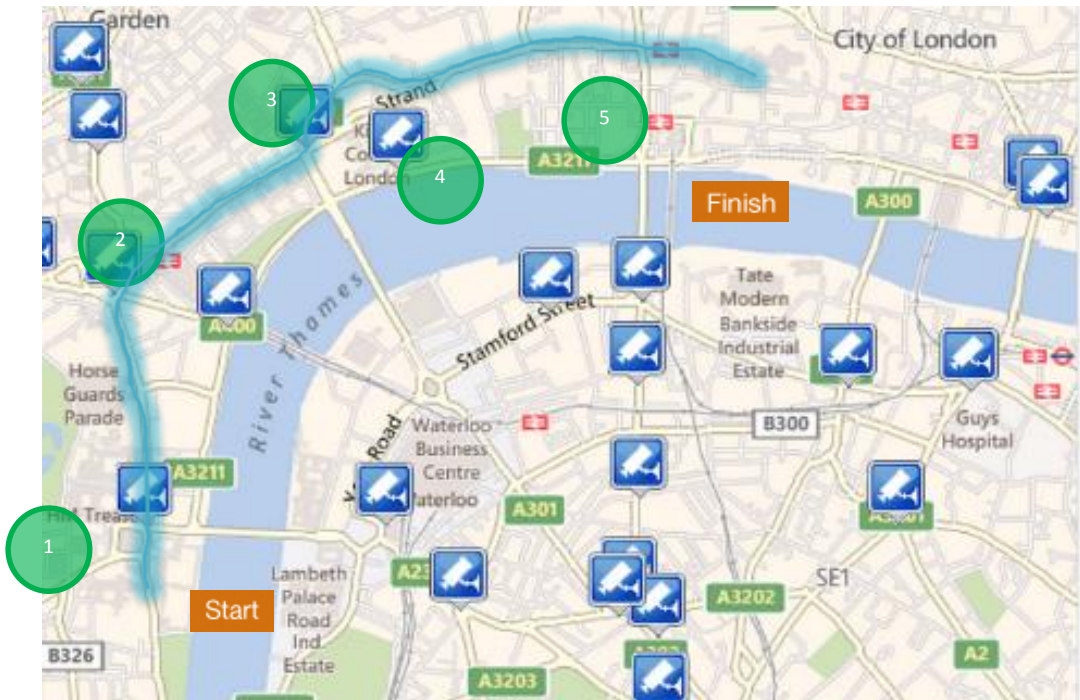
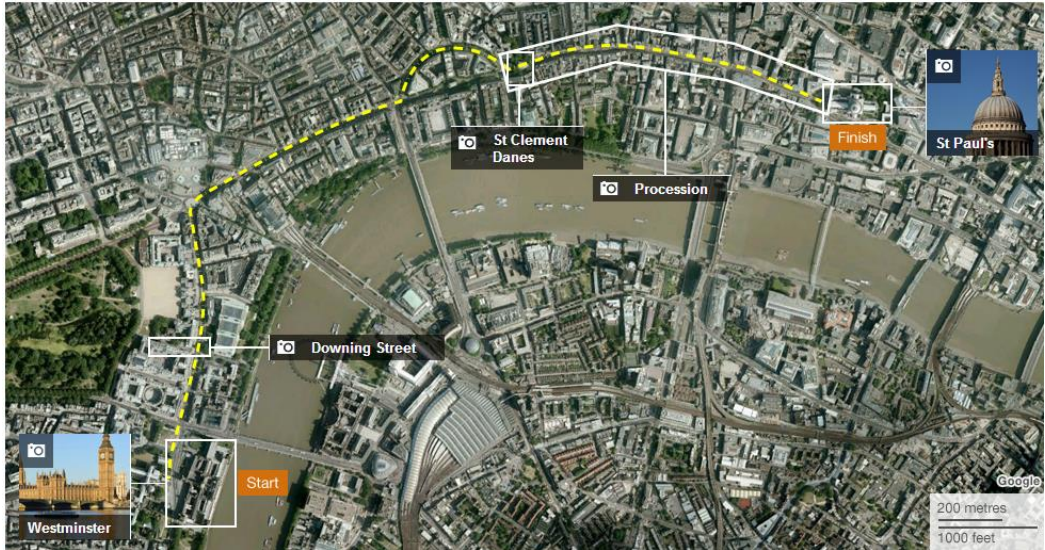
Preparation

As the procession through London was planned to last just 19 minutes I had to ensure that I was fully prepared for the analysis. My plan was to monitor the situation in London throughout the period, and use my system to alert me to any potentially interesting tweets. I achieved this by modifying the filter alert of the system from weather alerts to instead detect predefined tweet keywords. The keyword comments that have been rated as containing either strongly positive or negative connotations would then be displayed.

Keyword	Reasoning / Explanation
Thatcher	
Maggie	
Margaret	
Funeral	
Protest	
Respect	
Tory	Slang for Conservative
Witch	Protestors have been known to call Thatcher a witch
Riot	Small but nonetheless present potential for riot

Police	
Kettle	A police strategy of controlling protestors
Kettling	Verb of above
Strike	
Milk	Reports of protester plans to throw milk
Coal	Reports of protester plans to throw coal
Great	
Greatest	

Prior to the day of the funeral, I researched the exact roads that would be used and searched for any publically accessible webcams along the route. Following are two images - firstly of the official planned route, and secondly of a map displaying known webcam locations in the area.



In total there are five webcam locations along the route that should prove useful, with two webcams at Trafalgar Square (*location 2*). Whilst this isn't as many webcam feeds that I would have liked along the procession, I am aware that there may be areas of interest throughout other parts of London (from for example a protest group rally), and so I may be alerted to interesting tweets from other areas with additional cameras. In the analysis section I will refer to webcam locations by the ID numbers I have assigned above.

Analysis & Event Diary

On the morning of the funeral the first thing I did was proceed to check the availability of the above webcams. Unfortunately all of the above marked webcams, along with the vast majority of surrounding ones, were all unavailable and instead displaying the message shown to the right. This was a disappointing problem that was out of my control, as confirmed by the ITV news article '*Thatcher funeral security tight*' ^{Reference 14}, whereby information on security measures such as the monitoring of London cameras were detailed.

Camera 00001.06502
In use keeping
London moving

Camera 5
(Ludgate Hill)

Interestingly one of the images shown in the article was actually my above-marked '*Camera 5*', so whilst it was frustrating that I could not gain access to the camera video feeds, it was reassuring to know that the cameras I had chosen to monitor closely were the very same that the Metropolitan Police's '*Specialist Operations Room*' were monitoring.



My next step was to determine the weather outlook for the area, which I performed by loading into the system a generic tweet from the area and selecting the '*Animated Satellite Imagery*' option in my application. From this I discovered that there were large rain clouds over London, and so periodic rain was likely. This again was a disappointment as I believed that the turnout of onlookers - both those there to pay their last respects and those there to protest, would likely be comparatively lower than if it was a warm and dry morning.

The event officially began at 10:00am, and the slow military procession to St Paul's Cathedral was due to begin at 10:31 and arrive at 11:00. In total I made three separate tweet harvests, with the first collection occurring between 10:06 and 10:11. The twitter harvest filter was set to only receive tweets from within the area shown below, and no additional specified keywords were added to the filter at this stage.

In total 106 tweets were recorded in this five minute period, suggesting that there was a new geo-tagged tweet being created every 2.83 seconds. However, the fact that officially only 10.3% of Tweets are geo-enabled ^{Reference 15}, and that the Twitter live stream API will limit popular topic requests to a maximum of 1% of total Twitter conversation, I can estimate that the actual number of tweets in this period was around one thousand, therefore giving a true rate of around 3 tweets per second in the area.



The filter for my analysis application was set to display tweets containing messages of non-neutral sentiment. I will now describe some of the more interesting tweets that were brought to my attention via the program.

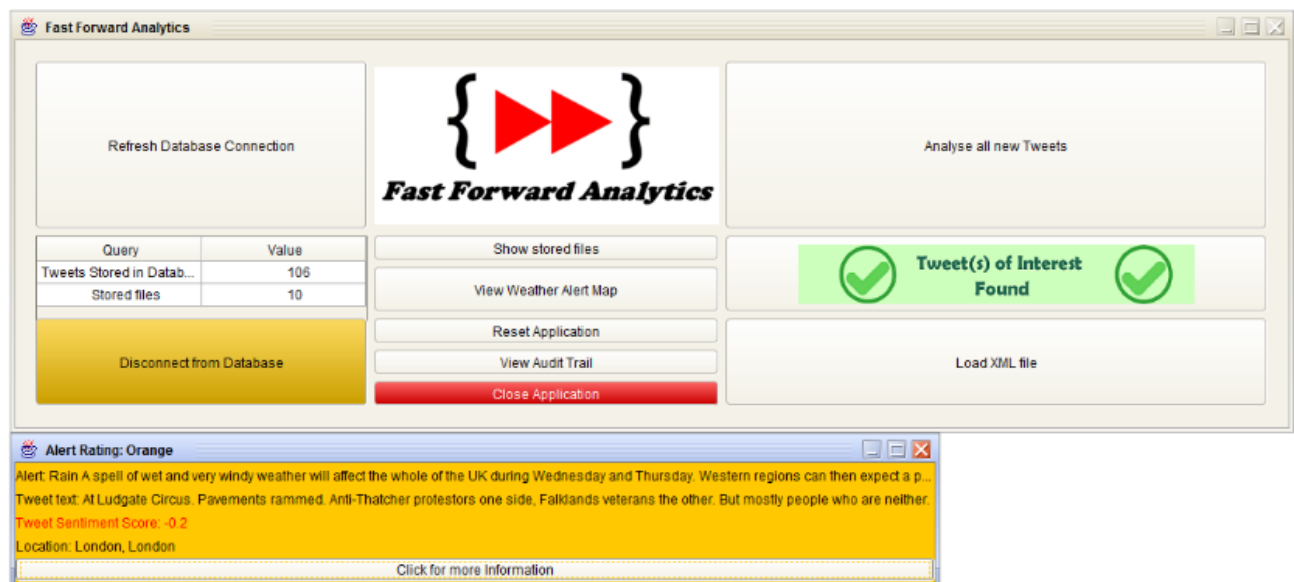
Tweet text: *"Mixed reception @ Ludgate Circus, protestors chant "what a waste of money", "get a job" comes reply from offices above"*

Time of tweet: **10:04**



Tweet text: *"At Ludgate Circus. Pavements rammed. Anti-Thatcher protestors one side, Falklands veterans the other. But mostly people who are neither."*

Time of tweet: **10:08**



These highlighted tweets of interest are promising, as my system detected them at 10:04 and 10:08, and only at 10:11 did the BBC report the following: ^{Reference 16}

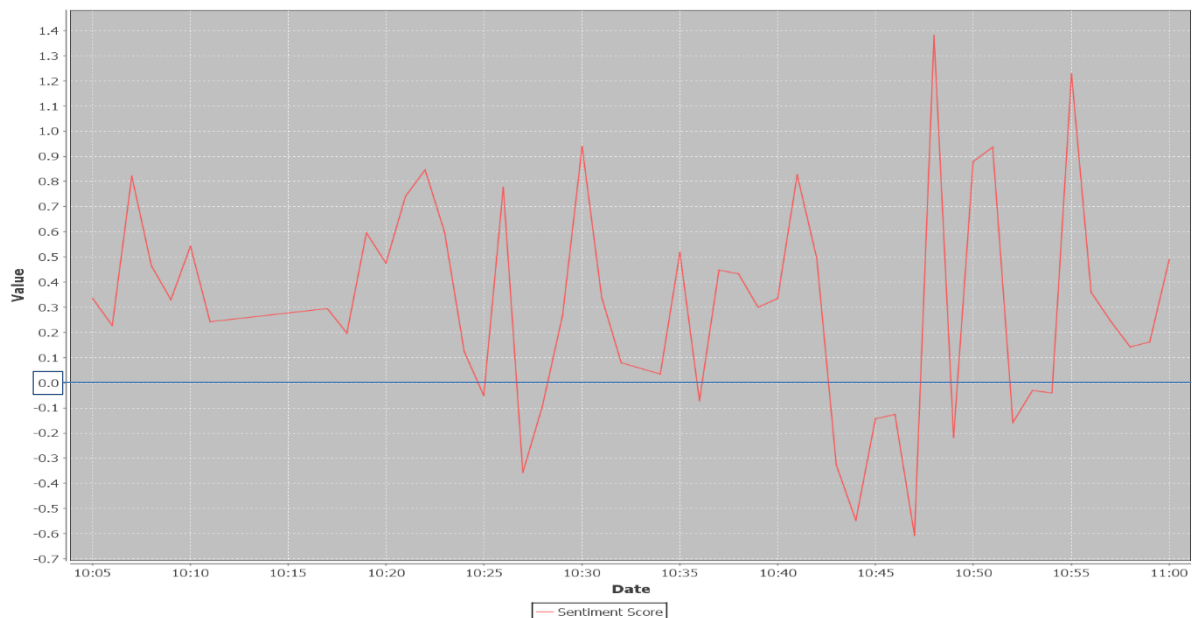
1011: The BBC's Home Affairs Correspondent Dominic Casciani says he has just heard the day's first protest at Ludgate Circus.

"As a military band marched up Fleet Street in all its pomp, one side of the crowd clapped them enthusiastically - but the other side booed loudly.

"Protesters chanted 'waste of money' as the band marched past."

If the system had full access to the webcams at Ludgate Circus as it normally would, I could have immediately and visually verified this information up to seven minutes before the BBC reported it, and without having a correspondent 'on the ground'.

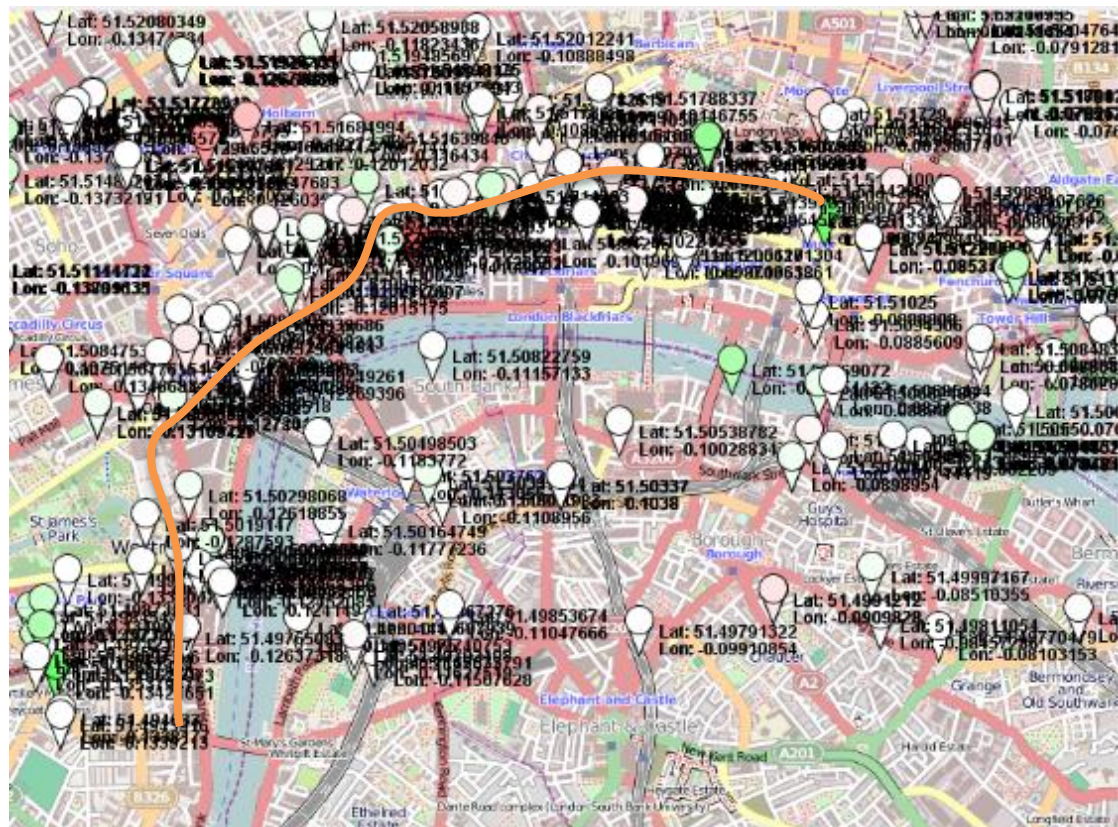
From 10:18 – 11:00 I harvested Tweets with the same filter of being within the predefined area, and had the system alert me to tweets with positive or negative sentiment. In this period I recorded 870 tweets, giving a geo-tagged tweet per second rate of 2.89 per second – this is almost identical to the first harvest’s rate. The figure below shows the sentiment timeline of these 870 tweets for the 42 minute period.



The overall average sentiment score of Tweets for the funeral was overwhelmingly positive, with the only period of consistent overall negative sentiment being between 10:43 – 10:47am. This result intrigued me, and after watching a recording of the funeral back, I realised that this happened to be the four minute period of the event where the Queen arrived and entered into St Paul’s cathedral, and so I decided to analyse further to discover if there was a connection.

I analysed the content of negative Tweets in this period, however I found that they did not appear to make comment on the attendance of the monarch more so than any other section of the hour. In actuality, I discovered that the reason for the recorded negative sentiment was not due to an increase in negative tweets, but rather a decrease in positive ones. My theory is that perhaps this could be linked to mourners, observers and other non-protestors wanting to savour the moment of seeing the Queen instead of tweeting, however I have no conclusive evidence to prove or disprove this observation.

The map below is a visual representation of where each of the 870 London-area tweets originated from. As you can see the most densely populated tweet areas are along the funeral route (represented by the orange line), as would be expected.



Just after the procession finished at 11:02, I recorded a further 2000 tweets with the keywords outlined in the table on pages 50-51. I did not perform real-time analysis on these tweets as they originated from across the globe and were harvested regardless of if they had geo-data present or not. The most distant tweet from Britain came from user EugeneH67 from Brisbane, Australia, who said '*#auspol Baroness Thatcher dedicated her life to serving Britain whilst Juliar dedicates hers to serving herself!!*', likely referring to the Australian Prime Minister Julie Gillard.

I have included a number of screenshot images on the following pages that I captured whilst performing the analysis, which illustrates how I believe the system would be put to best use for monitoring events such as this.



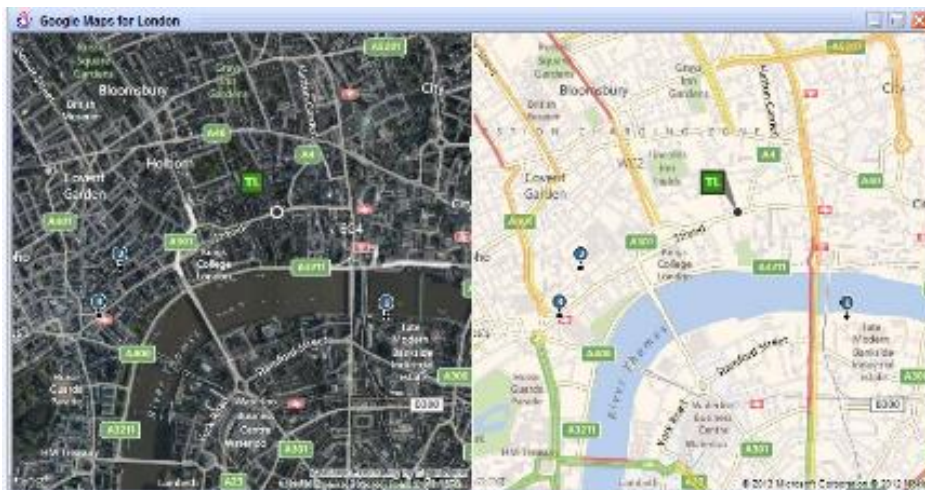
I was alerted to the tweet on the right of page, and was at this point making a note in the comment box (visible at the top left of the screen), stating 'A few boos at 10:37, overall very positive though'. The tweet text was 'Saw just the one protestor around Parliament St earlier. Polite applause as #Thatcher hearse passed. Crowds four or five deep.'



I did manage to very briefly gain access to the two Trafalgar square cameras at 10:16am and 10:26am, after being alerted to a tweet reading 'more police than public at Traf Square!'. As you can see from the rightmost image there was indeed a large police presence, and whilst these still images do not convey it clearly, there was no sense of protest or any other unexpected activity at the time the webcams were live.



In this screen capture taken at 11:09 I was searching for any locally uploaded Twitter photos, and the image of St Paul's Cathedral was returned.



These are two google map images that I generated at 10:50 showing the location of webcams in both images, and additionally a live traffic overlay on the right image.

Real-world Usage

Throughout the summer of 2012 (before starting the project), I worked for the Member of Parliament for Kingswood, Chris Skidmore MP, and during this time I discussed at length with him my plans for the project. Whilst the project has evolved considerably from these initial discussions, he was nonetheless intrigued in finding out how he could make use of the software to increase his team's efficiency in campaigning and other related activities. I presented the system to him over the Christmas period and he was very interested in its capabilities. Since then he has informed me that his team have used the prototype system that I supplied them with a number of times in order to gauge public opinion at events that Mr Skidmore has spoken at.

He plans to have his team use the program again at the upcoming political local elections on 2 May 2013, and so I am looking forward to hearing of how he utilises the updated program this time. Whilst it will unfortunately be too late to include information on his feedback in this report, I am nonetheless proud that I have managed to develop a system with a real tangible use.

Future Work

I believe that there is a very large number of potential features that could be added to the application, particularly when considering the modular nature of the filter stage where changes that modify the entire search algorithms of the database could be changed. However, I have also outlined a number of features that I feel could add value to the system, some of which I began to research and implement myself, but could not complete due to the time constraints of the project.

- Radio API Implementation:

I spent a considerable amount of time researching possibilities of how I could develop a feature that would immediately and reliably begin playing (or at least provide a user with a list of and links to) nearby radio stations to the tweet location. This would provide the user with another form of information gathering which could be listened to whilst performing other analysis activities as an additional and concurrent sensory information source. For example, if a user was interested in finding out why there was an unusually large number of people tweeting about being stuck in Cardiff, then a quick connection to a local radio station detailing traffic information could quicken the user's decision to check webcams along a particular motorway to find out the cause of the delay.

I found that there was very little support for development of location-based automated radio stations, which I was initially surprised at. I did begin to develop a rudimentary solution which involved querying the site www.radiofeeds.com, and modifying the request URL as appropriate for a location (for example, <http://www.radiofeeds.co.uk/query.asp?feedme=bristol>). From here my solution would then search through all text on the page for the string 'Listen Live', and attempt to access a feed this way. This was of course a very inelegant, inefficient and unreliable process, so I decided not to include it in the final code of the project. I have very recently discovered an Android Application titled 'Scanner Radio Pro' ^{Reference 17}, which appears to offer similar functionality to that which I require. Whilst there is no documentation on how the developers gain access to these radio streams, it is nonetheless promising and suggests that with further research and development, a solution may be possible.

- Betfair API implementation:

Betfair (the world's largest betting exchange) has a sophisticated API that can be used to monitor a large variety of betting markets, ranging from traditional sports to political events across the world, such as the US election where I demonstrated the potential uses of Betfair's historical data. I believe that having real-time access to information such as this could help provide further analytical opportunities to gain a clearer outlook on a particular tweet of interest, and could provide supporting evidence for a range of investigations which can be measured quantitatively with precise timing information.

- Google Maps Street View Implementation:

This feature would be relatively easy to implement, as it would just require taking the respective coordinates from a Tweet, and inserting them into a string that a button press would take as a link and direct the user to via their browser. The format would appear as follows:

```
https://maps.google.com/maps?f=q&q="latitude",-  
"longitude&t=k&ie=UTF8&ll="latitude",-  
"longitude&spn=0.001627,0.004849&z=18&layer=c&cbll="latitude,-  
"longitude"&panoid=n-BlkcbnAChIU3KO5yI_Qg&cbp=12,152.93,,1,6
```

This would then take the user directly into Google Street View mode at the exact location (or as close as possible) to the Tweet origin location. The only reason this has not been implemented is due to time constraints, and a focus on attempting to implement other features such as the effective saving & loading mechanism.

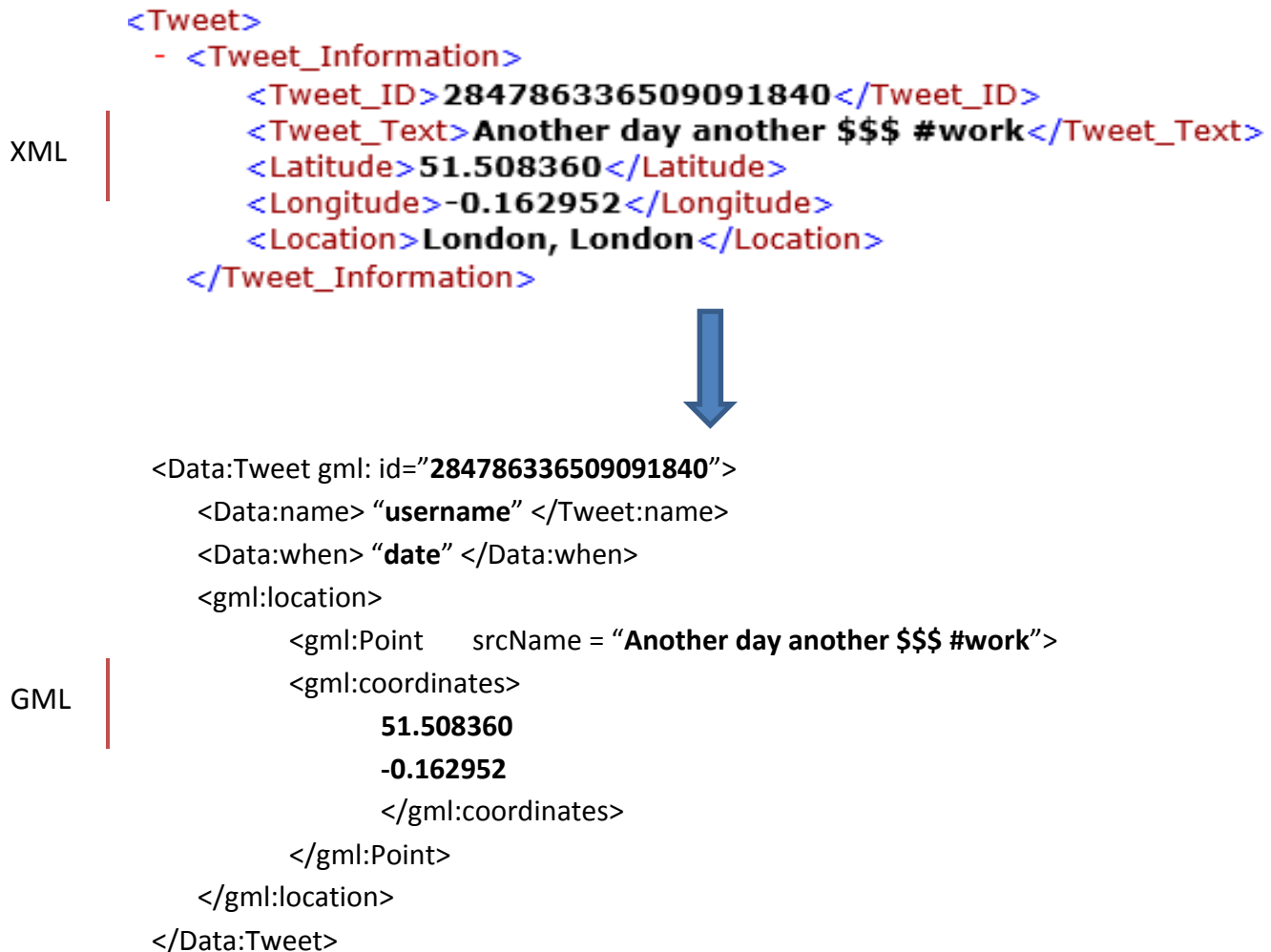
- Remote database access:

Due to Mongo Database's interface with Java, it should be relatively easy to access a non-locally-stored database on a host machine that numerous clients could access. This would be achieved by modifying the current database connection Java properties in the code shown below to appropriate values.

```
Mongo m = new Mongo("localhost", 27017);  
DB db = m.getDB("test");  
DBCollection coll = db.getCollection("tweets");
```

- Geographical Markup Language Support:

I believe that it would be both a useful and relatively simplistic task to modify the XML saving feature to enable a GML-supported (Geographical Markup Language) saving option. This would enable interoperability with numerous other software services such as KML mapping software, and CityGML (used in creating mapping 3D models) as GML is a universally-recognised markup language ^{Reference 19}. GML's format is similar to that of XML, and for clarity and material for any future developer, I have created a GML adaptation of an example of the system's current XML output, that I feel would be appropriate to be outputted from the system.



- HTTP Compression:

I have conducted a small amount of research into HTTP Compression when searching for ways to further optimise the program, and believe that if implemented correctly it could have a noticeable impact on system performance when performing network queries.

Conclusions

Overall I feel that the project has evolved and progressed at a significant rate since both the initial plan and the interim report. All tasks outlined in the interim report have been completed, in addition to a large number of further features and improvements, which I believe demonstrates the multitude of use-cases in which the system could be applied to - in many cases with only minor modifications necessary.

The meeting and presentation that was held between myself, my supervisor and four members of the Cardiff Social Science Department on January 30th 2013 was a very useful experience. It provided me with a number of ideas and concepts that lead to a combination of direct new code implementation into the software, the 'future work' section, or modified the overruling theme behind the project by deliberating new and interesting use-case possibilities.

The two demonstrations that I tested the system with were both very successful and gave quantifiable outlooks on events which could have otherwise only had qualitative opinions made about them. Due to the large scope of the project it would have been interesting to analyse other events, however it was not feasible to do so within the time constraints of the module.

There is one limitation of the system, however this is not a constriction that can currently be remedied whilst the product is in the development stage. This constraint is the arbitrary restriction that Twitter applies to its live stream API, whereby the interface will only return roughly 1% of all Tweets. Additionally, as previously mentioned official statistics show that only 10.3% of Twitter users have their 'share geo-location' option enabled – as such, the actual usable tweet output that the system can geographically analyse is realistically around 0.1% of total tweets. The percentage of tweets sharing geo-location is however increasing on a regular basis, and the advent of new technologies such as Google Glass and other integrated and location-aware network devices should help to further accelerate the location-sharing tweet ratio.

Reflections on Learning

Before I began developing the project I considered using a number of programming languages other than Java, due to previously finding some of the Java syntax difficult to fully grasp. However I am very pleased that I eventually decided to put the extra effort in to really understand the language, as the skills learnt will unquestionably be useful in my future career due to my graduate workplace making extensive use of Java.

I feel that the initial open-ended nature of the project ensured that I could tailor it to my particular interests and ensure that the system was something which genuinely excited me with its' potential future uses. The interest from third parties again encouraged me to strive to be as creative as possible, trying to cater for multiple use-case scenarios. However, the creation of a more formal set of required specifications during meetings with my supervisor in the second semester also helped me with the inevitable task of eventually narrowing down the project's focus to ensure I could present a usable product within the timeline of the module.

I believe that whilst my understanding of the practical areas of the project such as Java programming has improved considerably throughout the lifecycle of the project, I feel that the more adaptable scientific approach to problem solving that I have gained will provide me with a solid foundation for many future areas of my working life.

Glossary

API – Application Programming Interface

A protocol used as an interface by an application to communicate with another external application or service.

DES – Data Encryption Standard

An algorithm used in the encryption of data. More secure successors (such as Triple DES) have since been developed.

EXIF – Exchangeable image File Format

This image file format stores metadata about an associated image, and is capable of storing location-data such as GPS coordinates.

GML: Geographical Markup Language

A universally recognised format used for storing geographical information, and often also for geographical mapping purposes. Similar to XML in its layout and readability traits.

GPS: Global Positioning System

A global system of orbital satellites used to pinpoint a given location onto a map or chart.

GUI: Graphical User Interface

An interface that allows a user to interact with a computer system through images and options presented by the system to the user.

HTML – HyperText Markup Language

A standardised web-programming language used to present information to a user via a web browser.

IDE – Integrated Development Environment

An application such as Eclipse or Netbeans that is designed to increase productivity when developing software applications.

JSON – JavaScript Object Notation

A text-based data format used to represent a document's hierarchy and data structure in a human-readable form.

MongoDB – Mongo Database

A 'NoSQL' database allowing for efficient integration with Java.

XML – Extensible Markup Language

A markup language with similar uses to that of JSON. Data can be stored in nodes, with child and parent nodes used to show document hierarchies. Can be used to show data structures in a human-readable form, and parsed using the DOM method.

(XML) DOM – (Extensible Markup Language) Document Object Model

An API form used to parse data from an XML document by accessing node values.

References

1. Preece, P. A. (2012, November). Data Mining Processes. *Lecture 6, Slide 14*. Cardiff University, UK.
2. MeteoAlarm. (n.d.). *Alerting Europe for extreme weather*. Retrieved from: <http://www.meteoalarm.eu/>
3. Met Office. (2007, March 23). *Meteoalarm*. Retrieved from Met Office: <http://www.metoffice.gov.uk/weather/europe/meteoalarm/>
4. W3C. (2008, December 11). *W3C Guidelines for colour readability*. Retrieved from W3C: <http://www.w3.org/TR/WCAG/>
5. Google. (n.d.). *Google Maps*. Retrieved from <https://www.maps.google.co.uk/>
6. Microsoft. (n.d.). *Bing Maps*. Retrieved from Bing: <http://www.bing.com/maps/>
7. Feedzilla. (n.d.). *Feedzilla News*. Retrieved from <http://news.feedzilla.com/>
8. JTattoo. (2012). JTattoo GUI LookAndFeel. Wanfried, Germany.
9. Jar2Exe. (2012, October). Jar2Exe Application. Retrieved from <http://www.jar2exe.com/>
10. Flickr. (n.d.). *The App garden*. Retrieved from Flickr Developer API Services: <http://www.flickr.com/services/api/flickr.photos.getWithGeoData.html>
11. Oracle. (n.d.). *MessageDigest*. Retrieved from Java Documentation: <http://docs.oracle.com/javase/6/docs/api/java/security/MessageDigest.html#getInstance%28java.lang.String%29>
12. Ohio Secretary of State. (2012, November 8). *Ohio Secretary of State*. Retrieved from Ohio Secretary of State: <http://www.sos.state.oh.us/sos/upload/elections/2012/gen/FinalResults.xlsx>
13. beevolve. (2012, October 10). *An Exhaustive Study of Twitter Users Across the World*. Retrieved from beevolve: <http://www.beevolve.com/twitter-statistics/#e1>
14. Betfair. (2012, November 6). *Betfair Historical Data*. Retrieved from Betfair: <http://data.betfair.com/> *
15. ITV. (2013, April 17). *Massive police surveillance operation underway*. Retrieved from ITV London: <http://www.itv.com/news/london/story/2013-04-17/thatcher-funeral-security-tight/>

16. Semiocast. (2012). *Geolocation analysis of Twitter accounts and tweets*. Paris, France: Retrieved from Semiocast SAS:
http://semiocast.com/en/publications/2012_07_30_Twitter_reaches_half_a_billion_accounts_140m_in_the_US
17. BBC. (2013, April 17). *Baroness Thatcher funeral as it happened*. . Retrieved from BBC News: <http://www.bbc.co.uk/news/uk-22151589>
18. Edwards, G. (2013, March 11). Scanner Radio Pro. Retrieved from
https://play.google.com/store/apps/details?id=com.scannerradio_pro&hl=en
19. Jones, P. C. (2013, April 16). Advanced Database Topics. Cardiff University, UK.

* The raw data used to create the Presidential Election historical odds timeline diagram on page 48 can be found in the additional submission document Election Raw Data.xlsx, on lines 552, 533, 542, 543, 624, 722, 731 and 743.

Bibliography

- BBC. (2013, April 16). *Thatcher funeral: Guide to the day*. Retrieved from BBC News - Politics: <http://www.bbc.co.uk/news/uk-politics-22096613>
- beevolve. (2012, October 10). *An Exhaustive Study of Twitter Users Across the World*. Retrieved from beevolve: <http://www.beevolve.com/twitter-statistics/#e1>
- Daily Mail. (2013, January 17). *Airports and motorways closed as blizzard moves in from the west bringing huge blanket of snow set to cover Britain by this evening* . Retrieved from Mail Online: <http://www.dailymail.co.uk/news/article-2263837/UK-snow-forecast-Airports-motorways-closed-blizzard-moves-west.html>
- Doan, S., Ho Vo, B.-K., & Collier, N. (2011, May 31). *National Institute of Informatics*. Retrieved 12 06, 12, from National Institute of Informatics: <http://arxiv.org/ftp/arxiv/papers/1109/1109.1618.pdf>
- Fracsoft. (2013). *Electronic Market Trade and Analysis*. Retrieved from <http://www.fracsoft.com/>
- Goetz, B., Peuerls, T., Bloch, J., Bowbeer, J., Holmes, D., & Lea, D. (2006). *Java Concurrency in Practise*. Addison Wesley.
- Hamilton, G. (2004, October 19). *Multithreaded toolkits: A failed dream?* Retrieved from Java.net: http://weblogs.java.net/blog/kgh/archive/2004/10/multithreaded_t.html
- National Geographic. (2006, October 19). *Disaster Prediction, Social Networking Boosted by Geo-Data Feeds*. Retrieved November 3, 2012, from National Geographic News: <http://news.nationalgeographic.co.uk/news/2006/10/061019-tsunami-maps.html>
- NOAA - National Oceanic and Atmospheric Administration. (2012, August 27). *New NOAA awards to fund studies of weather warnings, social media, Internet tools and public response*. Retrieved November 3, 2012, from National Oceanic and Atmospheric Administration: http://www.noaanews.noaa.gov/stories2012/20120827_oarsocalscienceawards.htm
|
- Oracle. (n.d.). *The Java Tutorials*. Retrieved from Java Documentation: <http://docs.oracle.com/>
- Pew Research Center. (2012, June 4). *Partisan Polarization Surges in Bush, Obama Years*. Washington DC, United States of America. Retrieved from <http://www.people-press.org/2012/06/04/partisan-polarization-surges-in-bush-obama-years/>

- Simon Thompson. (2011, December 6). *International Geospatial Geocoding Conference*. Retrieved 11 22, 2012, from GeoCoding Conference:
http://geocodingconference.com/proceedings/pdfs/2011_iggc_geocoding_locationprecision.pdf
- Stack Overflow. (n.d.). Java Forums. Retrieved from stackoverflow:
<http://stackoverflow.com>
- The Guardian. (2013, April 15). *Margaret Thatcher funeral dress rehearsal takes place*. Retrieved from <http://www.guardian.co.uk/politics/2013/apr/15/margaret-thatcher-funeral-dress-rehearsal>
- Tindal, S. (n.d.). *The University of Edinburgh*. Retrieved 10 23, 2012, from CPC - Centre for Population Change:
http://cpc.geodata.soton.ac.uk/resources/downloads/Tindal_social_network_analysis.pdf
- Webcams.travel. (n.d.). *Webcam Map*. Retrieved from Webcams Worldwide:
<http://www.webcams.travel/map/>

Appendix

Figure 1: Interim Report Future Development

“I have a number of aspirations for how the project will develop in the second term, and have designed the system thus far with these in mind. Following is a description of the various features I feel would be possible, and that I plan to incorporate.

- Advanced plotting of elements such as the originating Tweet location and Webcam coordinates onto a mapping platform such as Google Maps or Bing Maps. Allowing options such as selecting satellite or aerial imagery, or overlaying additional data such as traffic information, will be considered.*
- I will attempt to harvest other types of data given specific coordinate values. This could be in the form of photo images uploaded to a photo-sharing service such as Flickr, provided metadata such as geographical location is publically available.*
- Returning news stories alongside this information may be useful, as it could provide further context for the overall Tweet alert, especially for overseas Tweet analysis which my project now permits.*
- The ability to store and export collected information in a clear and human-readable format, accessible independently of the program running, is crucial if the project is to be viable for a multitude of third-party uses.*
- Semantic Analysis was a large part of Nick Horne’s original project, so I believe it may be possible to integrate some of this data into my project without too much difficulty.*
- It has been suggested that incorporating an accurate and secure audit trail for what actions have been taken by an operator would be useful, particularly if the system was to be adopted and customised by any kind of law-enforcement or analysis institution. Considering how I could hash or encrypt an audit trail will be necessary, and provided time constraints are not a problem, this should be feasible.*

- *Extensive testing of the software in multiple countries, languages and even continents is necessary. Currently certain aspects of the system work fully with the United States of America's Weather Service Alert System (The National Oceanic And Atmospheric Administration), however further refinement is needed to ensure full compatibility.*
- *'Stress testing' the software to ensure it can cope with real-world situations is required. I anticipate that a significant amount of optimization will be necessary to ensure the system can maintain its real-time full functionality when dealing with potentially thousands of tweets per minute. I have however ensured that throughout the development this far I have commented my code effectively to allow any future optimization process to proceed relatively easily.*
- *User testing of the GUI aspects of the software will be important towards the end of the project, to ensure that it is both aesthetically pleasing and usable to an outsider without explicit knowledge of the system.*
- *As shown in the Term 2 Gantt chart (see Appendix figure 2), I have a meeting with the Cardiff University Social Science department on 30th January 2013. This will act as both a milestone/deadline for my software to ensure it works correctly, and also act as a form of guidance as to what the focus of the project should be for the remainder of the year."*

Figure 2: Term 1 & 2 Gantt Charts

Term 1 Gantt chart

Task	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8	Week 9	Week 10	Week 11
Initial Meeting + Planning											
Research Existing Data Mining Solutions											
Begin integrating existintg data with Weather APIs											
Submit Intial Plan			☆ Milestone ☆								
Create a draft GUI + popup box notifications											
Add visual data to notifications											
Weather Integration complete								☆ Milestone ☆			
Research other APIs											
Implement new API into existing program											
Produce Interim Report											
Submit Interim Report											☆ Milestone ☆

Term 2 Gantt chart

Task	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8	Week 9	Week 10	Week 11
Implement advanced mapping features											
Research & Implement photo-gathering features											
Add News API service for Tweet location											
Meeting with Social Science Department			☆ Milestone ☆								
Finalise storage & saving functionality											
Create effective & robust audit trail functionality											
Testing + Buffer space											
Implementation Complete								☆ Milestone ☆			
User GUI & functionality testing											
Analysis of collected data											
Production of Final Report											

The two Gantt charts above were presented in my initial plan and interim report respectively. All objectives outlined in both plans have been achieved, along with a number of additional tasks that were conceived throughout the development process.

Figure 3: Interim Report System Overview

