

# Social Media Visualization Tools and Techniques

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



*Interim Report*

# Abstract

Localised factual information is freely available from a variety of sources for the general public's consumption. Similarly there is a large amount of geo-tagged social data from sources such as Twitter that can be harvested in an efficient and real-time manner. However, there are currently few applications that utilise and combine both types of information, encompassing factual, semantic and visual data, to help formulate a new set of unique outputs for a given geographical position. This project will attempt to amalgamate data from numerous sources into a single application, in order to provide a tangible and real-world outlook on the current conditions of any physical location, with the possibility that knowledge generated could be distributed to third parties who would act accordingly on this new information.

The primary focus of the project is on intelligent data harvesting, whereby given a set of criteria deemed important by an operator, any starting information gathered from a particular geographical area would be passed through a number of data gathering processes, in order to build upon the information provided from the initial data sample. Recording, displaying and storing this data in an efficient manner will be critical to the project's success, as will providing the user with a large degree of flexibility as what possibilities they have for any external post-processing. As such, all data recorded must be both efficient for a computer to process quickly, given the quantity of data being analysed, but also human-readable and extractable for other uses.

Given the relative lack of current applications in this area of computing and social science, I feel that the possibilities for this project are abundant. Opportunities for the media in automatically and immediately gathering real-time data and visualisations on areas of interest, before journalists arrive – or to begin scouting an area for particular interests to direct them specifically on arrival, are present. Similarly, there is potential for a slightly modified system to be used for market research or canvassing for businesses or political parties to discreetly gather data on the public, negating the need for expensive, time consuming and arguably less reliable data-gathering from humans. I feel the prospects for commercial use from this developing area of research are vast, and that particularly with the advent of emerging technologies such as Google Glasses, the data, and hence accuracy and usefulness of projects similar to this, can only increase.

# Acknowledgements

There are a number of people without whom this project would not be at the stage it is now. I would like to thank my supervisor Prof N J Avis, for his continual help and guidance throughout this term. I would also like to thank Omer 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, I would like to thank Weather Underground ([www.wunderground.com](http://www.wunderground.com)), 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.

# Table of Contents

<b>Abstract.....</b>	<b>01</b>
<b>Acknowledgements.....</b>	<b>02</b>
<b>Introduction .....</b>	<b>04</b>
<b>Background .....</b>	<b>05 - 06</b>
<b>Implementation.....</b>	<b>07 - 08</b>
Formatting pre-existing data .....	09 - 10
Refining data .....	11 - 12
Displaying Alert Popup Box.....	13 - 15
<b>Current System State &amp; Testing Methodologies .....</b>	<b>16 - 19</b>
<b>Future Development .....</b>	<b>20 - 21</b>
<b>Conclusions .....</b>	<b>22</b>
<b>Appendix.....</b>	<b>23</b>
Term 1 Gantt chart.....	23
Term 2 Gantt chart.....	23
<b>Glossary.....</b>	<b>24</b>
<b>References.....</b>	<b>25</b>

# Introduction

The timeline of this particular project varies from others, whereby usually the first term would be used to research and plan, and the second term to implement the software in the second term. For this project, as outlined in my initial plan and with the support of my supervisor Prof Nick Avis, I have also spent a considerable amount of time implementing large aspects of the software, including numerous data-gathering and visualisation elements, and am in a position whereby I can concentrate on which particular aspects of the project I wish to pursue in the second term. As a result, this interim report will have a strong emphasis on the implementation of the software thus far, and will naturally lead on to the final report which will contain details on how the system has been further refined.

The primary objective for the first semester of this project has been to find and create programmatic methods to collect and combine data from an abundance of sources, and amalgamate it into a single system. This includes both qualitative social sources such as Twitter Feeds and Flickr photos, as well as quantitative, respected factual data, such as that of the Weather Channel and the Met Office. The key underlying relationship between all of this information however is the geographical location data that can also be harvested in some cases.

A reoccurring theme present throughout the project is how multiple pieces of data can be combined and transformed into information, and how this information can in turn be used to provide actual useful knowledge to appropriate parties. To clarify, raw data such as temperature or wind speed could only be classified as useful information if a location for these respective values was also known. Additionally, having access to supplementary information such as historical weather or current environmental alerts could arguably transform this information into usable knowledge, if for example it provided local authorities with a prediction of how the current situation will change in the future, allowing them to act on this knowledge.

There are currently a large number of applications and web services that the public can use to find localised information such as weather alerts or traffic information relatively easily. However, given the quantity of publically-accessible location-tagged social information provided by services such as Twitter and Flickr, there are comparatively few services that focus on combining the two. It is my belief that in the coming years, as the number of devices capable of providing accurate location information increases, the demand for this type of amalgamation and analysis will grow rapidly.

In line with my initial plan, I have concentrated on the planning, design and implementation of the software program for this semester, and will focus on refining and fully analysing the results of my system in the spring semester. I am progressing through my initial plan outline at a quicker pace than first envisioned, and as such have included functionality beyond that of which was initially specified. These developments have been regularly discussed with Professor Nick J Avis, and should provide the project with a number of interesting and exciting possibilities for the next semester.

# Background

Data Mining describes the process of collecting, storing and organising large amounts of data, with the intent for it to be integrated into another system or subsystem at a later date. Its purpose is usually to produce either an overview of a problem by reviewing a large amount of data, or alternately to provide an accurate analysis on a particular dataset, often a user, in order to complete personalised post-processing to produce unique properties for each dataset. The task is usually completed automatically in order to bypass time-consuming human analysis, and the data is often stored in a database system in which the data can be queried and returned in predetermined groups in an efficient manner.

This project focuses on combining geographical data, or geo-location data, with other forms of data. The geo-data I am harvesting consists of a Latitude and Longitude coordinate value, accurate to seven decimal places – this equates to an accuracy of approximately 1.11cm. However, it should be noted that whilst the *accuracy* of the geo-data is exact, this does not necessarily mean that the data is *precise*. The accuracy rating of 1.11cm shows that given a Latitude and Longitude it is possible to find exactly where that is on a map, however it does not mean that the actual coordinates provided represent the true location – this is defined as the precision of the data. On the contrary, the usefulness of any given geo-data is dictated by a combination of both precision and accuracy, and as such an accuracy level provided can only be useful up to the corresponding precision level of the originating data source. It is widely reported and accepted that the average level of precision for a commercial GPS system is precise to between four and five decimal places – between 1.1 and 11 metres. Reference 1

Due to the large number of variants of global positioning systems that I harvest in the project, such as those of mobile phones, cameras and weather stations, it is not possible to give an exact value for the particular precision of a coordinate. However, due to the nature of the project it can be assumed that the majority of data harvested is largely useable, particularly for aspects of the analysis that focus on constructing information on larger areas of land.

A number of web services and Application Programming Interfaces I use return data in an XML format. XML (Extensible Markup Language) is a markup language similar in style to HTML, and is capable of displaying data in a format that is both human and machine readable. The Java programming language allows for the retrieval of elements of an XML document by searching for a specified node value – this is useful for retrieving data relating to multiple inputs from one XML documents, hence conserving bandwidth and increasing the performance of my application.

There are a number of APIs (Application Programming Interfaces) that my system makes use of which are listed below, accompanied by a brief description. Further information regarding the actual implementation of each API is present later in the report.

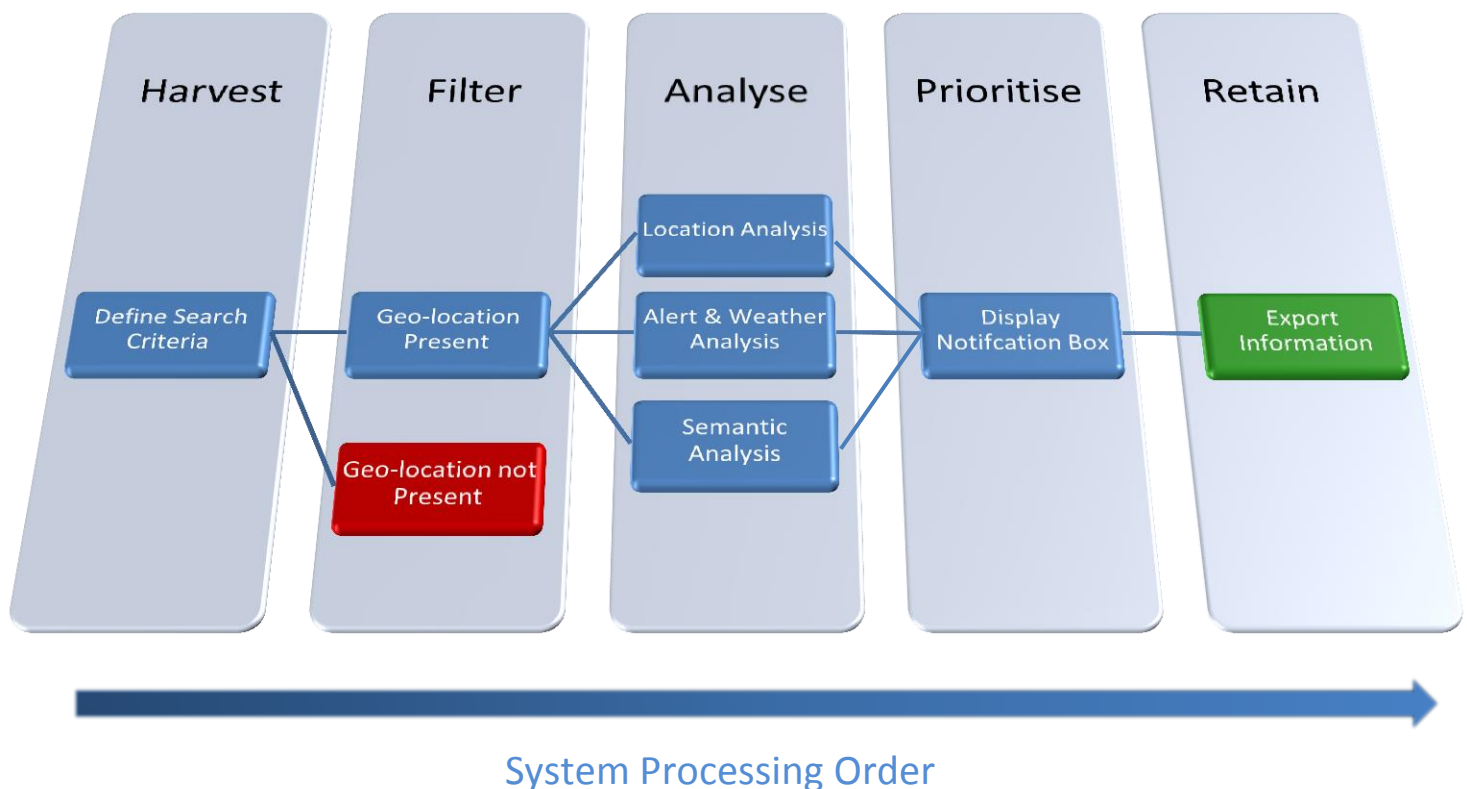
- Twitter Streaming API: [www.dev.twitter.com](http://www.dev.twitter.com)
  - This API provides real-time access to the global stream of Tweet data. It allows the developer to search for tweets by keyword or location, and returns any tweets that match these key(s) in JSON format – a format similar in many ways to XML.

- Wunderground API: [www.wunderground.com/weather/api/](http://www.wunderground.com/weather/api/)
  - This API is used extensively in the project at numerous points, and provides information relating to the weather and current conditions. Wunderground.com is a subsidiary of The Weather Channel, and they obtain their information from a combination of sources, including each country's national weather station (i.e. The Met Office), and European wide systems (i.e. EUMETNET). The following APIs are used in my project:
    - Geolookup – Provided with Latitude & Longitude coordinates, this returns the county, nearest settlement and nearest weather station.
    - Forecast – Provided with Latitude & Longitude coordinates, this returns the current weather conditions for the last hour. Data includes High/Low Temperature, Wind Speed and current conditions (e.g. Thunderstorms).
    - Alerts – Provided with a weather station, this returns true if there are any weather alerts for the area, and the corresponding information including alert rating, description and expiry date. This data is gathered from EUMETNET MeteoAlarm.
    - Satellite/Radar – Provided with a weather station, this returns an animated weather satellite/radar image of the location.
- Microsoft Bing Maps: [www.microsoft.com/maps/developers/](http://www.microsoft.com/maps/developers/)
  - This API returns graphical maps of the world when variables such as coordinates, map type and map-markers are provided.
- Google Maps: [www.developers.google.com/maps/](http://www.developers.google.com/maps/)
  - This API also returns maps, and some locations on Google Maps have more detailed satellite imagery than Bing Maps. However, there are a number of legal usage restrictions on the using Google Maps that Bing maps do not have.
- Yahoo! Flickr: [www.developer.yahoo.com/flickr/](http://www.developer.yahoo.com/flickr/)
  - This API returns any photos uploaded to the Flickr photo-sharing service within a provided radius of a set of coordinates. Geo-tagged images are also available for harvesting.
- Feedzilla News: [www.code.google.com/p/feedzilla-api/](http://www.code.google.com/p/feedzilla-api/)
  - This API is used to gather news stories, given a specific geo-location.
- Full Contact: [www.fullcontact.com/developer/](http://www.fullcontact.com/developer/)
  - This API returns the population of any given settlement.
- Webcams.travel: [www.webcams.travel/developers/](http://www.webcams.travel/developers/)
  - This API returns any active webcams within a specified radius, for any given location.

# Implementation

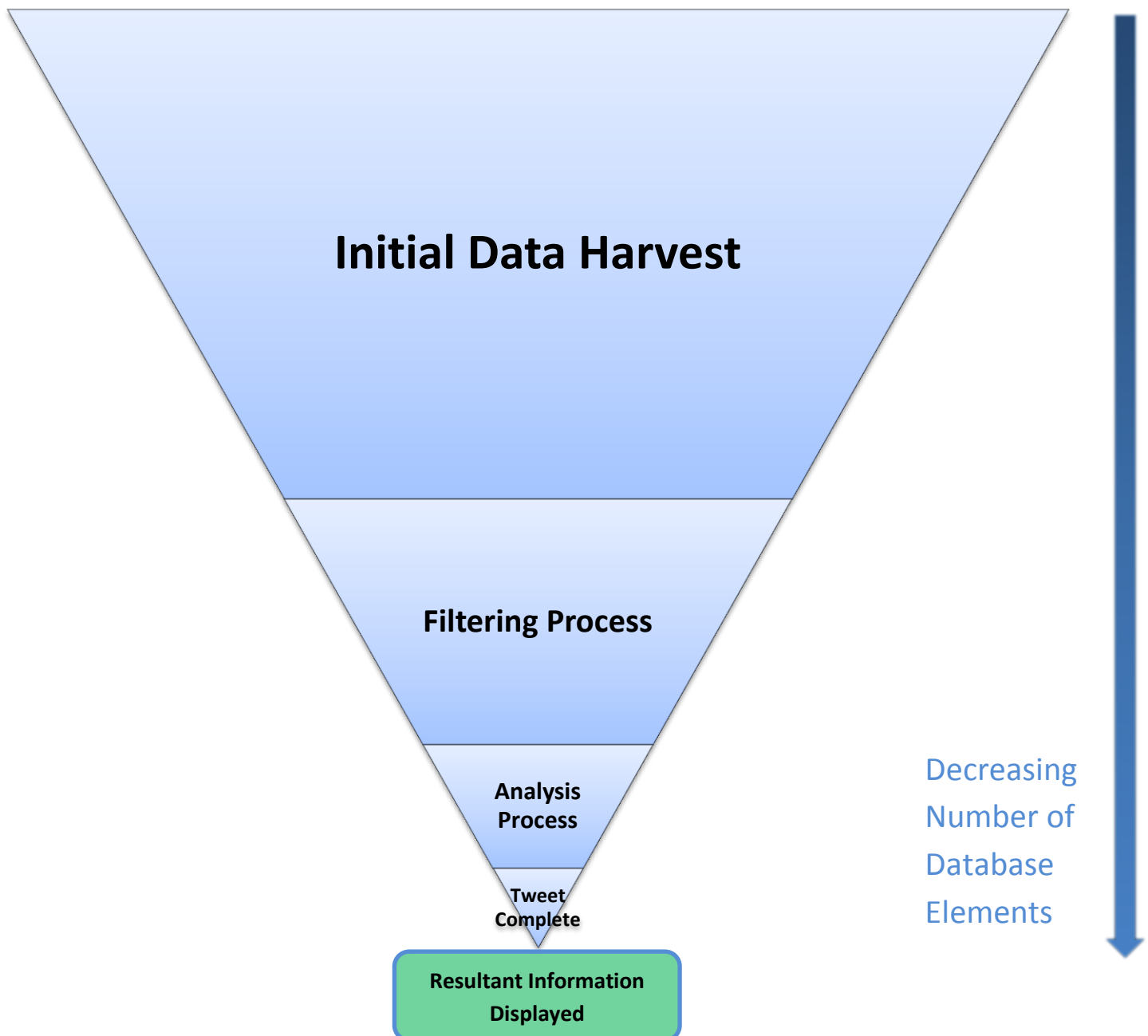
A good starting point for the project was suggested at my initial meeting with Prof Omer Rana on March 22<sup>nd</sup> 2012, whereby it was suggested that I utilise the Twitter collection software created by previous Cardiff University student Nicholas Horne. This software proved to be a useful base for my project, as it allowed me to focus on the post-collection analysis and visualisation aspects of the project, without having to recreate data-harvesting software that already existed. I have kept my project code completely separate from this pre-existing program, and use data provided by Nick Horne's program by redirecting the database outputs to my own database. I also made a number of small modifications to the original twitter-collection Java program to better suit my system's needs, by adding a database value for each entry specifying explicitly whether the Tweet has a geo-location or not.

The diagram below represents the basic structure of the software. After populating the database with Tweets by supplying an appropriate search criterion, the Tweet collection is automatically filtered to remove unsuitable entries, explained in further detail later in this section. All remaining Tweets are analysed for data such as current weather information, and then marked as complete. Once a dataset marked as complete is detected, it is displayed to the user for human inspection. If, after human analysis the information is deemed important, it can be saved and exported to an external file.





As the diagram below shows, the number of elements to be analysed or displayed decreases as the system progresses through its computations. The full database after an initial data harvest is first filtered to remove any elements that do not have geo-locations. The remaining Tweets are then checked for compliance with my predefined search criteria, and if they pass these tests their geo-coordinates are used as keys for use in querying a number of other resources. Tweets that have passed all tests are then passed to the GUI Notification Class, whereby the information gathered is displayed to a user for human-inspection.



All code summarisations on the following pages describe the code that I produced for the project, and are edited to only display notable portions of code – a full copy of code is included in the Appendix. The programming language used for this project is Java – I made this decision based on two factors: firstly, it has been the language taught within the School the most, and secondly because I believed it would allow easier integration with certain aspects of the existing project, also written in Java.

## Formatting pre-existing data

My Java class **formatting.java** is the first class ran when analysing tweets, and it performs the following actions:

- Access the text file outputTweets.txt – this is a text file containing key data about all tweets and is generated by my modified section of the previous Tweet Collection program. Data utilised by this class are the unique database ID value, Tweet text and Latitude & Longitude values.

```
String filename1 = "C:\\Users\\Steven\\Documents\\University\\Year 3\\Project  
Code\\projectworkspace\\javavis\\bin\\outputTweets.txt";  
  
try {  
  
    FileInputStream input_file = new FileInputStream(filename1);  
    Scanner input = new Scanner(input_file);  
  
    while (input.hasNextLine()) {  
  
        counter = counter + 1;  
  
        String line = input.nextLine();  
  
        input.findInLine("\\\"id\\\" :");  
  
        String twitterID = input.next();
```

- Tweets without latitude & longitude values do not receive any further processing from this point forward.

- The class then makes a URL request to the Wunderground.com API geolookup and forecast services for each latitude & longitude combination found, sequentially. I have an extended API key for this resource, courtesy of The Weather Channel, and am authorised to make up to 100 calls per minute, with a maximum of 5000 requests per day. This plan would usually have a value of \$300 per month, and could be further increased to 10,000/minute and 1,000,000/day if necessary. The API returns XML data for high and low temperatures, average wind speed, current conditions (e.g. snow), and the ID number of the nearest weather station.

```

URL WundergroundWeatherXml = new
URL("http://api.wunderground.com/api/205f2bbxy5cf8536/forecast/geolookup/q/" +
latitude + "," + longitude + ".xml");

URLConnection uc = WundergroundWeatherXml.openConnection();
uc.connect();

...

NodeList high = forecastdayElement.getElementsByTagName("high");

for (int f = 0; f < high.getLength(); f++) {

    Node highNode = high.item(f);

    if (highNode.getNodeType() == Node.ELEMENT_NODE) {

        Element highElement = (Element) highNode;
        NodeList celsius = highElement.getElementsByTagName("celsius");
        Element cel = (Element) celsius.item(0);

        if(x == 1){

            x = x + 1;

            System.out.println("Highest Temperature: " + cel.getTextContent() +
"°C" + eol );

        }

    }

}

```

The *latitude* and *longitude* variables are updated with each Tweet's respective latitude and longitude values for each pass of this section of code.

This section parses the returned XML document for appropriate node values, and assigns the *cel* Element with the textual temperature data for the given coordinates.

## Refining data

My Java class **refine.java** is the second class ran when analysing tweets, and executes when every tweet has been passed through **formatting.java**, and performs the following actions:

- Gathers tweets from the previous outputTweetsWeather text file, and processes them to create a human-readable text file, unique to each Tweet using a numerical marking system (i.e. Tweet1, Tweet2). These are only the Tweets that return true when queried for possessing a geo-location. The aforementioned necessity for having a human-readable format that the information can be presented in is the primary reason for this class's existence.

```
while (input.hasNextLine()) {
    PrintStream out = new PrintStream(new
    FileOutputStream("C:\\Users\\Steven\\Documents\\University\\Year 3\\Project
    Code\\Tests\\temporaryFiles\\outputTweetsWeatherAlertsRefined" + unique + ".txt",
    true));

    System.setOut(out);

    String line = input.nextLine();
    if (line.startsWith("+t")) {
        String databaseID = input.next();
        System.out.println("----- NEW TWEET -----");
        System.out.println(" ");
        System.out.println("Database ID: " + databaseID);
    }
}
```

Printing each Tweet  
dataset to a unique  
the *unique* variable  
incremented for ea  
new Tweet

Printing each Tweet dataset to a unique file - the *unique* variable is incremented for each new Tweet

- Tweets without a valid weather station location do not receive any further processing from this point forward.
- For Tweets with a valid weather station location, the Wunderground.com API is queried, returning information on any weather alerts issued for the area in XML format. This information is appended to the unique Tweet text file.

```
URL AlertXML = new URL("http://api.wunderground.com/api/205f2bbxy5cf8536/alerts" +  
location + ".xml");  
URLConnection uc = AlertXML.openConnection();  
uc.connect();  
  
NodeList alerts = doc.getElementsByTagName("alerts");  
for (int v = 0; v < alerts.getLength(); v++) {  
    Node alertsNode = alerts.item(v);  
    System.out.println(" ");  
    System.out.println("\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\ ALERTS \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\");  
    System.out.println(" ");
```

My unique  
Wunderground  
API key.

My unique  
Wunderground.com  
API key.

To ensure that the system can be used in multiple regions around the world, variations of the code shown below used to access Weather Alerts are needed. This is because the format that data gathered for Weather Alerts is in varies by country, and so the system needs to deduce which format the data is before parsing the information.

```
if(uscell1 == null){  
  
...  
}else if(  
    counterForpopupBox = counterForpopupBox + 1;  
    counterForreadFile = counterForreadFile + 1;  
    System.out.println("Alert: " + uscell1.getTextContent());  
    NodeList message = alertsElement.getElementsByTagName("message");  
    Element us2 = (Element) message.item(0);  
    System.out.println("Alert Description: " + us2.getTextContent());  
  
    ...  
  
    readFile.main(args, counterForreadFile);  
    Toolkit.getDefaultToolkit().beep();  
    unique = unique + 1;  
  
} else{  
...  
}
```

Integer variables to be sent to the popupBox.java and readFile.java classes.

- Once a Tweet that has both a geo-location and a valid current weather alert has been found, the class readFile.java is executed. This class extracts the unique Database ID value of the Tweet, and appends it to a list stored in 'forpopup1.txt'. This is stored as a text file so that a record of the order in which the Popup alerts are displayed is recorded in an easily-accessible format, which myself and the supervisor feel would be useful after Prof Nick Avis discussed with members of the Metropolitan Police on 28/11/12 the need for a clear and accurate audit trail of actions taken in projects like this.
- When all Tweets have been analysed and categorised correctly, the program initiates the popupBox.java class, if the file forpopup1.txt exists. This file will only exist if the readFile.java class has been executed, which occurs when a tweet with both geo-location data and a valid alert was previously found.

```
File file = new File("C:\\Users\\Steven\\Documents\\University\\Year 3\\Project  
Code\\Tests\\temporaryFiles\\forpopup1.txt");  
boolean exists = file.exists();  
if (!exists) {  
    // Do nothing. Later implement an appropriate notification for the user.  
}else{  
    popupBox notify = new popupBox(counterForpopupBox, 400, 100, 1);  
}
```

## Displaying Alert Popup Box

The popupBox.java class is the largest java file, and 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 can be multiple Tweets being displayed simultaneously, but if the user clicks to view detailed 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 procedures carried out are summarised below:

- The class will calculate the number of entries in the forpopUp1.txt file, and create a for loop based on this result – the resulting number is the number of Tweet entries stored in the database that are of interest to us (in the program's current state this is Tweets satisfying the condition that they have both a geo-location and an active weather alert for their area).

```
BufferedReader reader = new BufferedReader(new
FileReader("C:\\Users\\Steven\\Documents\\University\\Year 3\\Project
Code\\Tests\\temporaryFiles\\forpopUp1.txt"));
int lines = 0;
while (reader.readLine() != null) lines++;
reader.close();
int unique = 0;
for(int i = 0; i < lines; i++ ) {
    unique = unique + 1;
    ...
}
```

- Each Tweet's respective unique Database ID value is then gathered, and is used to query the Mongo Database to return all information stored about each Tweet.

```
String databaseID = input.next();
System.out.println("Database ID: " + databaseID);
long idOfInterest = Long.parseLong(databaseID);

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

BasicDBObject searchQuery = new BasicDBObject();
searchQuery.put("id", idOfInterest);

DBCursor cursor = coll.find(searchQuery);
System.out.println(cursor.next());
```

Java code to connect to the appropriate Mongo Database via the local host on port 27017. There is potential capability for the database to be remotely accessed in future.

Searches the database for ID value specified

- Analysis of the resultant output is now performed, and variables that are in a form suitable for use in later computations are created:

```
...
final String location = input5.nextLine();
...
int startPosition4 = tweetText1.indexOf("fullName") + "\" : e : \" : \".length();
int endPosition4 = tweetText1.indexOf("\"} , \"hasLocation\", startPosition4);
String locationText = tweetText1.substring(startPosition4, endPosition4);
final String locationForNews = locationText.substring(locationText.lastIndexOf("
")+1);
```

- Once these variables have been created, the GUI parameters are defined and displayed. A small portion of the Java Swing GUI code is displayed below:

```
final JFrame frame2 = new JFrame();

JButton satelliteButton = new JButton("Display animated satellite imagery");

frame2.setTitle(alertRatingLine);
frame2.setLayout(new GridLayout(10, 1));

frame2.add(new JLabel(alert));
frame2.add(new JLabel(alertIssued));
frame2.add(new JLabel(alertExpires));
frame2.add(new JLabel("Tweet text: " + tweetText));
frame2.add(new JLabel("Tweet Sentiment Score: " + sentimentScore));
frame2.add(new JLabel("Location: " + locationText));
frame2.add(new JLabel(currentConditions));
frame2.add(satelliteButton);
...
frame2.getContentPane();

int frameWidth = 250;
int frameHeight = 120;
```

These parameters define the style of the Java Swing User Interface. I have used the grid layout as it was initially simple to implement and manage – I will consider changing to another layout, such as a Flow style, when I have finalised what information will be displayed.

- Action Listeners for the JButtons are now created, containing appropriate code:

```
satelliteButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e)
    {
        JFrame frame4 = new JFrame();
        frame4.setTitle("Satellite Images for Tweet Location");
        frame4.setLayout(new GridLayout(1, 1));
        try {
            URL satelliteURL = new
            URL("http://api.wunderground.com/api/205f2bbxy5cf8536/animatedsatellite" +
            location +
            ".gif?basemap=1&timelabel=1&timelabel.y=10&num=10&delay=50&width=640&heigh
            t=480");
            ImageIcon image = new ImageIcon(satelliteURL);
            frame4.add(new JLabel(image));
            frame4.getContentPane();
            frame4.pack();
            frame4.setLocation(600, 630);
            frame4.setVisible(true);
        }
    }
});
```

Variables for satellite API:

- 10 second total animation time
- 10 animation updates
- 5 second end pause

- Finally, the initial Popup Box GUI is set visible, along with the Alert Rating's corresponding alert rating colour. Additionally, the Java code finds information regarding the user's visual display unit:

```
...
else if (alertRatingLine.equals("Alert Rating: Red") ||
alertRatingLine.equals("Benachrichtigen Rating: Rot") ||
alertRatingLine.equals("Estatus alerta: Roja"))
{
    frame.getContentPane().setBackground(Color.RED);
    frame2.getContentPane().setBackground(Color.RED);
    Dimension screenSize= Toolkit.getDefaultToolkit().getScreenSize();
    frame.pack();
    frame2.pack();
    frame.setVisible(true);
}

Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
frame.setBounds((int) screenSize.getWidth() - frameWidth, 0, frameWidth, frameHeight);
frame.setLocation(frameLocation, frameLocationHeight);
frame.setVisible(true);
frameLocation = frameLocation + 240;
```

Multiple languages must be supported in order to support data gathering from other countries. In this shortened example the alert rating text has been translated to German and Spanish, following an 'OR' statement. Many other modifications are also necessary in order to support multiple country data sources.




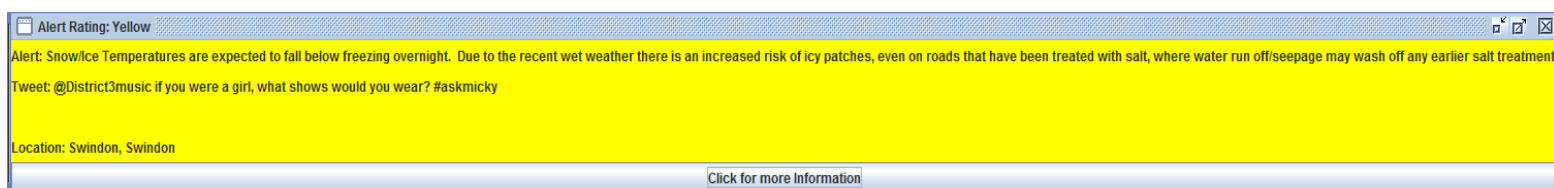
## Current System State & Testing Methodologies

Currently, upon launch the system presents the user with a basic Java Swing GUI with two buttons – one allowing the user to perform analysis on Tweets already stored in the database, and the other closing the connection to the database and the application itself. After clicking on the ‘Analyse all Tweets’ button, the system will perform the actions outlined in the previous section for each Tweet stored in the Mongo Database. A separate Java Swing Alert box will appear for each ‘Tweet of Interest’ found, displaying a summary of information gathered about the Tweet. This Interface is shown below:



- The colour of the box indicates the Weather Alert Rating.
- The ‘Alert:’ text is a description of the Alert, as defined by The Met Office (or the equivalent organisation in another country), via MeteoAlarm.
- The ‘Tweet:’ text is the Tweet message that the user broadcasted.
- The ‘Location’ field contains information on the nearest settlement and town: Swindon, Swindon indicates that the Tweet originated from the town of Swindon, whereas Mendip, Somerset shows the message came from the Mendips, which are located in the Somerset area.

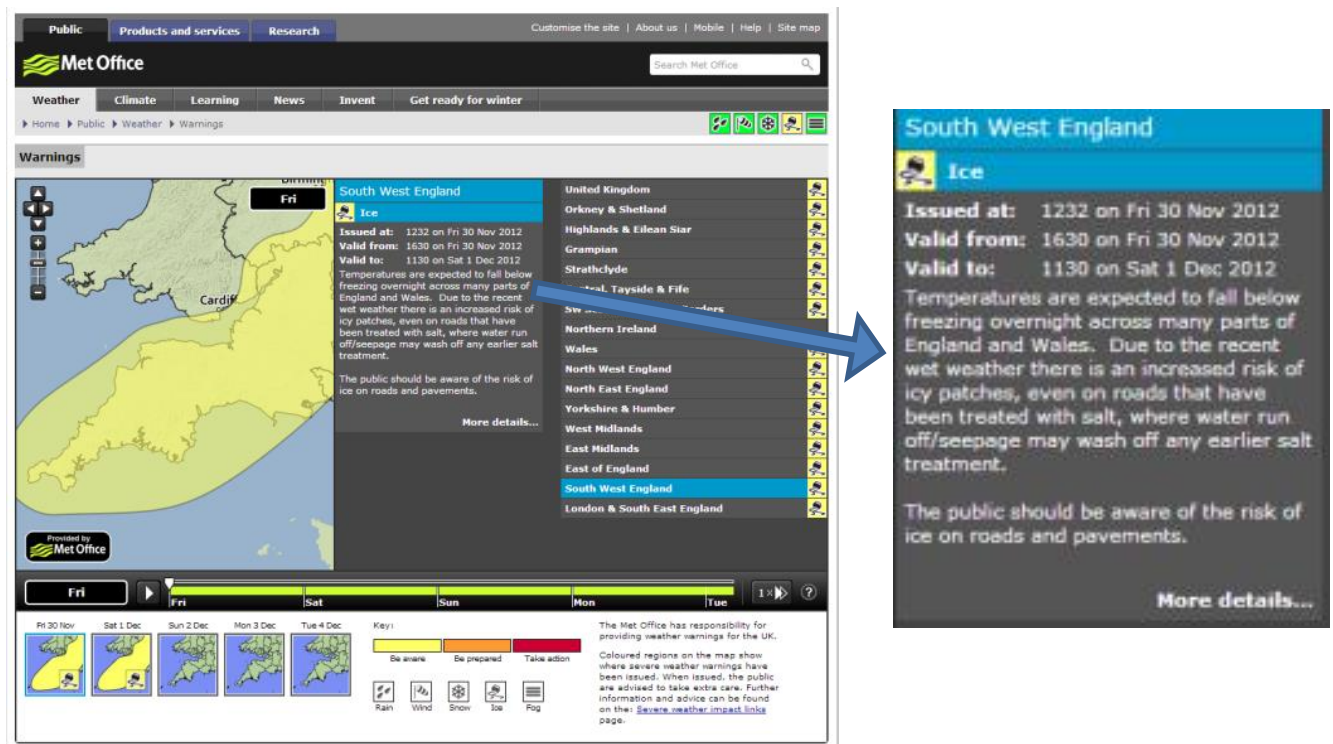
Maximising the size of the Swindon window by clicking on the  button will show the full Tweet message and Alert description:



The Alert description reads as follows:

‘Alert: Snow/Ice Temperatures are expected to fall below freezing overnight. Due to the recent wet weather there is an increased risk of ice patches, even on roads that have been treated with salt, where water run off/seepage may wash off any earlier salt treatment’.

To test and confirm that this alert is correct, I manually checked the Met office website. The two images below are screenshots of their Weather Warning's tool for the South West of England, where Swindon is located.



Clicking on the 'More Information' button on the generated Alert Popup Box expands the User Interface to show additional information, as shown below:



This larger notification box shows more detailed information for the weather alert, and additional local weather details.

- 'Alert Issued:' displays a timestamp of when the Weather Alert was issued.
- 'Alert Expires:' displays a timestamp of when the Weather Alert will expire, according to the Met office.

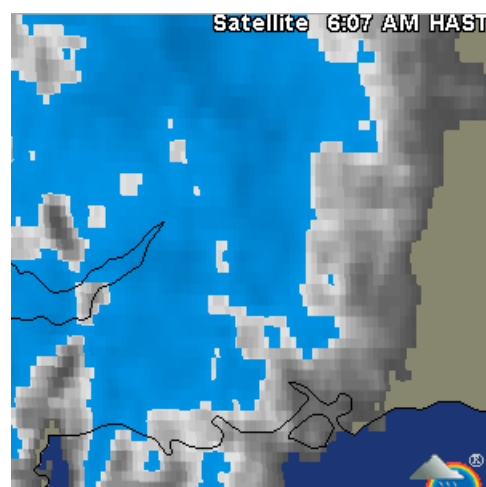
- 'Current Conditions:' displays categories such as 'Chance of Rain', 'Snow' or 'Thunderstorms'.
- 'Highest/Lowest Temperature:' gives details of the corresponding high/low temperatures for that particular area.
- 'Wind Speed:' displays the current wind speed for the area, as recorded from the nearest weather station to the initial Tweet location.

Clicking on the 'Display nearby webcams:' button queries the webcams.travel API, and returns the user with images of the three nearest webcams to the Tweet location. The output for the Swindon Tweet is shown below:



Information detailing the location and timestamps of the webcams are usually present on the webcam image – this metadata is independent of my API call, and are returned attached to the images. Additionally, information such as Wind Speed/Direction at that particular webcam's location is sometimes displayed, as shown in the second webcam image above.

Finally, selecting the 'Display animated satellite imagery' button will display an animated image such as the example below:

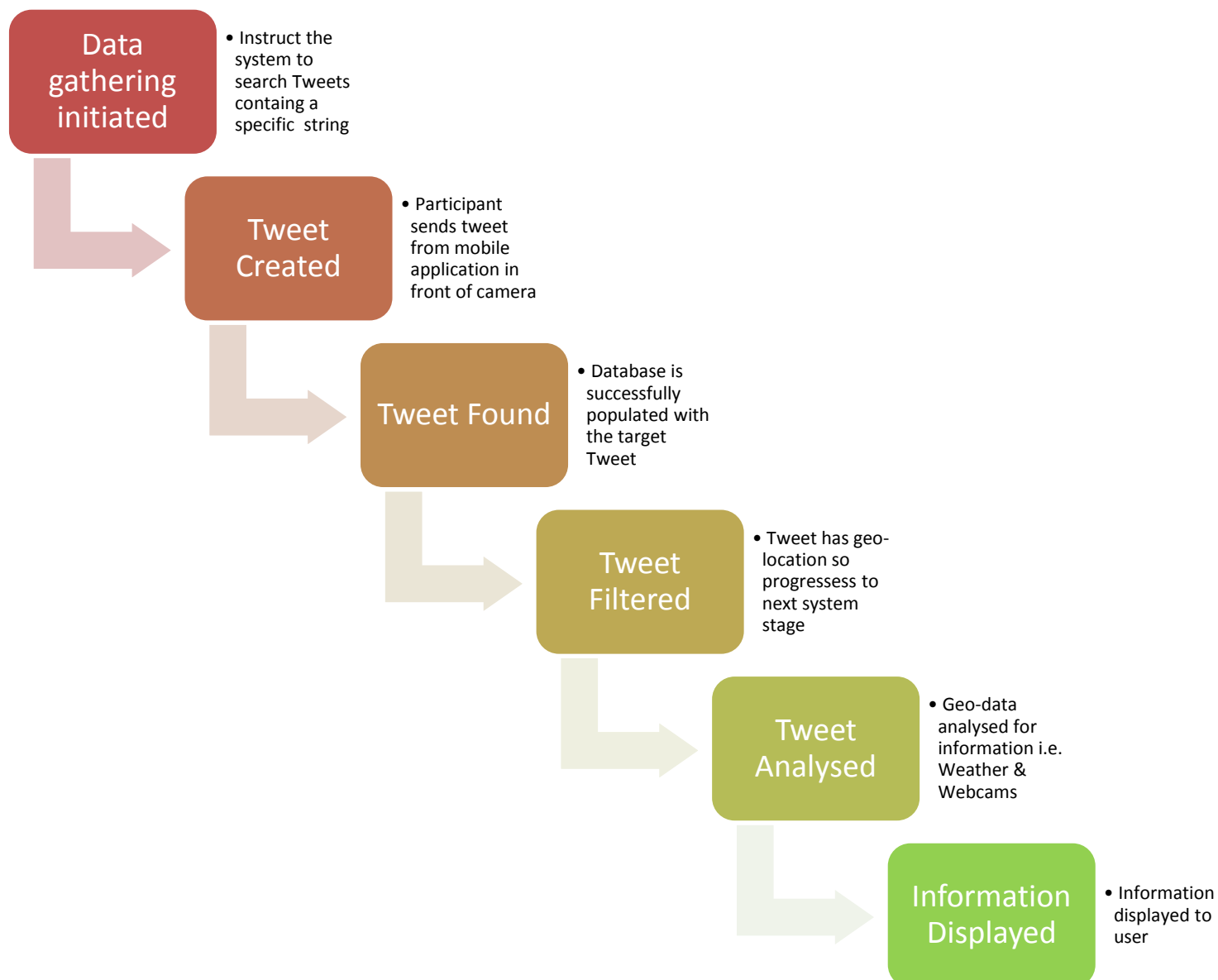


This image updates within the Java Frame, showing how the simulated satellite imagery for rainfall and cloud movement has changed over the last ten hours.

After confirming that all aspects of the project were working correctly using locations within the UK, I performed a full system test on how the software manages with European data. A number of modifications were necessary, such as modifying the manner in which XML DOM parsing took place, and adding 'OR' statements to include different language versions of 'Yellow' or 'Red' etc. for the alert rating colour scheme.

Once I was confident that the system was working, I contacted a friend living in Barcelona to stand in view of a webcam that I had specified, and to send a particular Tweet. I then performed a Europe-wide search for Tweets containing this text, effectively using the Tweet message as a Key. The Tweet was found and the correct notification box appeared, specifying the location as Barcelona. I deemed the test as a complete success, as after selecting 'Display nearby Webcams', my friend was in the returned image looking at the webcam.

The diagram below shows the processes employed by the system to achieve the described output from the experiment. In order to perform this test, I had to have a known webcam location to instruct the participant where to stand, and I had to supply a textual 'key' for them to tweet using the Twitter mobile application.



## 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), I have a meeting with the Cardiff University Social Science department on 30<sup>th</sup> 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.

## Conclusions

I feel very encouraged by the progress made thus far on the project, and the support from third parties such as The Weather Channel, in addition to the interest from areas such as Law Enforcement and Political Canvassing in projects similar to mine has reassured me that this project is very much part of an evolving area of research, and has a significant number of potential future uses. I believe that the number of location-aware applications will continue to increase, especially as the quantity and accuracy of the data improves. Research in this area is gaining considerable financial support from large organisations, such as the NOAA (National Oceanic and Atmospheric Administration), who have recently (August 27, 2012) awarded grants of \$879,000 for four projects focusing on social media analysis for weather analysis and prediction. <sup>Reference 2</sup>

A study by SemioCast has placed the percentage of publically-accessible Tweets with accompanying geo-location metadata at 0.77% (July 2012). Whilst this may seem a small number, when put into context with statistics such as there being 1.058 billion Tweets broadcasted between June 1<sup>st</sup> and June 31<sup>st</sup> 2012 alone, and with over half a billion registered accounts, 0.77% equates to just over 8 million coordinate-tagged Tweets created per month. The advent of ever-increasing variations of location-enabled devices such as smart phones, tablets and cameras, will inevitably lead to this number increasing, especially as developing nations begin to enter the market. <sup>Reference 3</sup>

I feel that all of the aforementioned objectives I have outlined are achievable, with the only limiting factor being time. Other development opportunities will likely present themselves whilst working on the project as they have done in the first term, and prioritising which features should be added and which left for possible future implementation will have to be considered and discussed with my supervisor throughout the year. I am excited by the prospects of what this system is capable of achieving, and also of actually analysing and reporting on the findings of the system towards the end of term.

# Appendix

## 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 existing data with Weather APIs											
Submit Initial 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 ☆

The Gantt chart above was the plan presented in my initial report. All objectives outlined in this plan have been achieved, and a number of additional tasks not shown have also been completed.

## 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											

As shown in the above chart, during the first half of the second term I will continue to develop and refine the software I have developed thus far. I have a meeting with the Cardiff University Social Science department on 30<sup>th</sup> January 2013 where I will present the system and discuss the future direction of the project. This fixed deadline will ensure that sufficient development work is completed early in the term, leaving time to analyse the results of the software later in the year. The final report will contain a full breakdown of each section of the system, explaining how precisely the program filters, prioritises and displays harvested data. It will also contain information on my post-processing analysis methods, and any conclusions I find that relate to the practicality of this kind of software solution.



# Glossary

## **API – Application Programming Interface**

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

## **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.*

## **JSON – JavaScript Object Notation**

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

## **XML – Extensible Markup Language**

*A markup language with similar uses to that of JSON - data can be stored in nodes, with child 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**

*A form of API used to parse data from an XML document by accessing node values*

## References

1. Simon Thompson. (2011, December 6). *International Geospatial Geocoding Conference*. Retrieved November 03, 2012, from GeoCoding Conference:  
[http://geocodingconference.com/proceedings/pdfs/2011\\_iggc\\_geocoding\\_locationprecision.pdf](http://geocodingconference.com/proceedings/pdfs/2011_iggc_geocoding_locationprecision.pdf)
2. 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.noaaews.noaa.gov/stories2012/20120827\\_oarsocalscienceawards.html](http://www.noaaews.noaa.gov/stories2012/20120827_oarsocalscienceawards.html)
3. Semiocast. (2012). *Geolocation analysis of Twitter accounts and tweets*. Paris, France: Semiocast SAS.

Goetz, B., Peuerls, T., Bloch, J., Bowbeer, J., Holmes, D., & Lea, D. (2006). *Java Concurrency in Practise*. Addison Wesley.

Doan, S., Ho Vo, B.-K., & Collier, N. (2011, May 31). *An analysis of Twitter messages in the 2011 Tohoku Earthquake*. Retrieved December 06, 12, from National Institute of Informatics:  
<http://arxiv.org/ftp/arxiv/papers/1109/1109.1618.pdf>

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>

Tindal, S. (n.d.). *The University of Edinburgh*. Retrieved October 23, 2012, from CPC - Centre for Population Change:  
[http://cpc.geodata.soton.ac.uk/resources/downloads/Tindal\\_social\\_network\\_analysis.pdf](http://cpc.geodata.soton.ac.uk/resources/downloads/Tindal_social_network_analysis.pdf)